programmez.com

[Programmez!]

Le magazine des développeurs

224 DÉCEMBRE 2018

PYTHON

LE LANGAGE SUPERSTAR

M 04319 - 224S - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

Printed in EU - Imprimé en UE - BELGIQUE 7 e - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 e - TOM 1020 XPF - MAROC 55 DH

Rejoignez
notre cordée !

elcimaï LE GROUPE

INFORMATIQUE

LA PERFORMANCE
NE DOIT RIEN AU HASARD

Melun

Paris

Elcimaï, éditeur informatique en plein développement
recrute de forts potentiels sur Melun et Paris

Vous êtes :

- **Ingénieur études et développement .Net C#** H/F
- **Ingénieur support applicatif** H/F
- **Ingénieur d'études C / C++ Unix** H/F
- **Ingénieur études et développement Java** H/F
- **Ingénieur études et développement Sharepoint** H/F
- **Consultant AMOA banque** H/F
- **Consultant AMOA assurance** H/F

Retrouvez toutes nos opportunités de carrières sur
www.elcimai.com - E-mail : **candidature@elcimai.com**

Vous interviendrez sur :
Expertise Technique (MOE) C / C++ ;
C#, .Net, JAVA, J2EE, sharepoint…
Expertise et assistance Métier/ Fonctionnelle
(MOA et AMOA) : banque et assurance

EDITO

Soyons badass !

Ce qui est bien avec le monde technologique, c'est que de temps en temps, une annonce surprend. J'aurai pu discourir sur le rachat de Red Hat par IBM, pour la modique somme de 34 milliards… A côté, le rachat de GitHub par Microsoft paraît presque ridicule… Non, une des annonces les plus surprenantes eut lieu mi-novembre : Amazon Web Services (tu sais la partie cloud d'Amazon) se veut comme le sauveur des développeurs Java avec Corretto !

La théorie est très simple : proposer un OpenJDK aux utilisateurs AWS, gratuit et open source. AWS part du constat que beaucoup d'utilisateurs et de développeurs utilisent Java sur AWS, donc pourquoi ne pas avoir sa propre distribution Java ? Et pour faire comme Oracle, Corretto sera même LTS, tu sais le support long terme. Et comme Amazon est sympa avec toi, Amazon te promet des fixes de sécurité, des améliorations de performances, tous les trimestres et si y'a une urgence véridique, promis, on te balance un patch ipso facto ! Et tout ça en utilisant OpenJDK 8, d'où le nom de Corretto 8. Et le tout sera disponible maintenant, enfin non, au 1er trimestre 2019… Et comme ça, tu pourrais remplacer toutes les versions de Java SE… Téléchargeable en préversion depuis AWS et disponible sur Amazon Linux 2, Windows, macOS et Docker.

Et comme on t'aime beaucoup (si, si) : on t'annonce aussi Corretto 11, pour août 2024.

Gratuit et LTS, n'est-ce pas ce que Oracle propose désormais pour Java ? Bah si, mais en payant. AWS est théoriquement un sérieux concurrent à Oracle Java. En tout cas, 2019 promet d'être agité côté Java.

Ce mois-ci, Programmez! vous propose une belle brochette de technologies et de langages :

- Tu veux coder ton propre Space Invaders ? alors fais-toi plaisir… en C++.
- Python est LE langage incontournable. Nous vous proposons un petit dossier très sympa. On y parle de la migration 2.x vers 3.x, Python dans les calculatrices et les boards de prototypages, et bien d'autres choses.
- Un projet robotique avec le TJBOT.
- Un retour terrain très intéressant sur comment mettre en place une architecture micro-services.
- On parle aussi de hack hardware, de refactoring, de IFTTT, de .Net Core, du framework Blazor, de Vue.js ou encore de programmation Amiga.

Bon code et avec un peu d'avance, bonnes fêtes !
(Programmez! est une bonne idée cadeau)

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !
Programmez! #225, dès le 2 janvier 2019

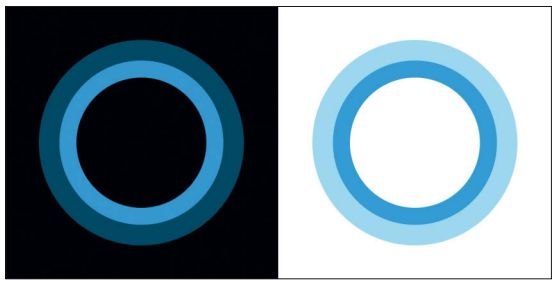
Angular 6 : *les nouveautés*

2019 : *quels changements pour les développeurs ?*

Développement mobile : *comment choisir la bonne techno pour le bon développement ?*

Supercalculateurs : les États Unis reprennent la main

Dans le monde merveilleux des supercalculateurs, les États-Unis peuvent se vanter d'avoir repris du poil de la bête. Deux nouvelles machines américaines viennent en effet de rafler les deux premières places dans le classement Top 500 de novembre, qui établit le palmarès des superordinateurs les plus puissants au monde. « Summit » d'IBM a été le premier à percer en raflant la première place du classement au mois de juin 2018, affichant une puissance de calcul de 122 petaflops. Dans l'édition de novembre, sa puissance de calcul a bondi pour atteindre 143 petaflops et le supercalculateur a été rejoint en tête de classement par « Sierra », un autre superordinateur américain qui dépasse le Sunway Tahulight chinois d'une courte tête en affichant une puissance de calcul de 94 petaflops. Les machines chinoises monopolisaient la première place du classement depuis 2012.



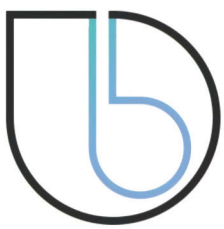
Le directeur de Cortana quitte le navire

Javier Soltero, dirigeant de l'activité dédiée à l'assistant vocal Cortana chez Microsoft, est sur le départ. Celui-ci profite d'une réorganisation interne de Microsoft, qui a décidé de transférer la gestion de l'assistant vocal de la branche AI+Research à la branche Experiences&Devices sous la houlette de Rajesh Jha. Javier Soltero avait pris la direction du développement de Cortana au mois de mars dernier.

Google

lance Visbug, une extension Chrome pour l'édition de site web

Google a présenté un nouvel outil de développement baptisé Visbug. Celui-ci simplifie la modification d'une page Web à l'aide d'une interface simple WYSIWYG. Google compare l'outil à Firebug, une extension Firefox qui permettait déjà de procéder à des retouches mineures sur un site. Via Visbug, l'utilisateur pourra ainsi modifier l'apparence d'un site directement depuis le navigateur sans avoir besoin de se plonger dans les CSS du site. Google conçoit Visbug comme un outil qui pourrait être utilisé en complément d'outils de développement Web plus complets, afin de procéder à des retouches rapides lors d'une démonstration par exemple.



Bixby s'ouvre aux développeurs tiers

Samsung a annoncé le lancement de son « Bixby Developer Center » à l'occasion de sa conférence développeur. Au sein de celui-ci, Samsung entend regrouper des guides, références et API à l'intention des développeurs tiers qui souhaiteraient développer des applications destinées à son assistant vocal Bixby. Samsung tente de combler la distance qui le sépare de la concurrence (Alexa, Siri et Google Assistant) dont les écosystèmes sont déjà bien installés.

MICROSOFT PROPOSE UNE NOUVELLE DISTRIBUTION LINUX SUR SON MICROSOFT STORE

Microsoft accueille la nouvelle distribution WLinux basée sur Debian sur le Microsoft Store. Fait notable, il s'agit de la première distribution payante disponible pour WLS pour les utilisateurs. Celle-ci est facturée pour le moment 9,99 $, mais son prix est normalement de 19,99 $.
Développée par Whitewater Foundry, elle est présentée comme un "environnement de terminal Linux rapide pour les développeurs" utilisant WSL pour Windows 10. Outre cette évolution, Windows 10 1809 corrige également le manque de support historique de Windows Notepad pour les fins de ligne Linux et Unix.
Notepad utilise désormais Windows line ending (CRLF) par défaut, mais les développeurs peuvent afficher, modifier et imprimer les fichiers existants sans perdre le format actuel du fichier.

Sept nouvelles variantes de Spectre et Meltdown découvertes

Un groupe de chercheurs a publié un article détaillant sept nouvelles variantes de Spectre et Meltdown affectant les processeurs. Deux des sept nouvelles attaques sont des variantes de l'attaque Meltdown, tandis que les cinq autres sont des variantes de l'attaque Spectre.
L'équipe de recherche a déclaré avoir communiqué toutes ses conclusions aux trois fournisseurs de processeurs dont elle avait analysé les processeurs, mais que seuls ARM et Intel avaient reconnu leurs conclusions.
En outre, l'équipe de recherche a également découvert que certaines mesures d'atténuation déjà mises en place par les constructeurs n'étaient pas parvenues à arrêter les nouvelles attaques.

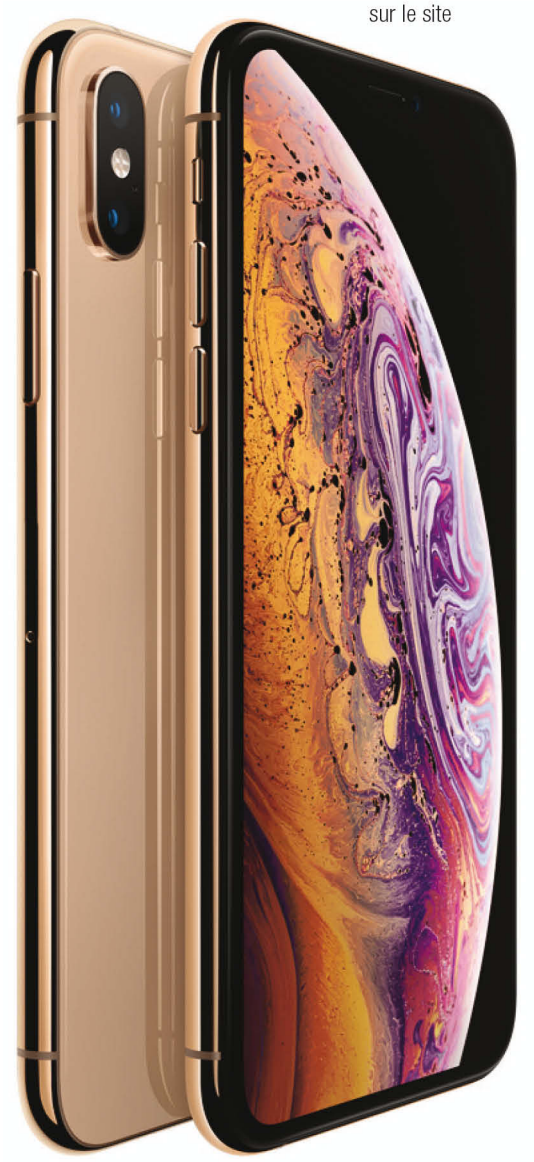
PROLONGATION JUSQU'AU 21 DÉCEMBRE

COMMANDEZ WINDEV 24

OU WEBDEV 24 OU WINDEV MOBILE 24

ET RECEVEZ LE NOUVEL iPhone X S

Choix de la couleur sur le site



iPhone X S

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 24 (ou WINDEV 24, ou WEBDEV 24) chez PC SOFT au tarif catalogue avant le 21 Décembre 2018. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. **Voir tous les détails sur : WWW.PCSOFT.FR ou appelez-nous au 04.67.032.032**
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.980,00 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels. Tarifs modifiables sans préavis.

CHOISISSEZ :

- **iPhone XS** 64GB
- **iPhone XR** 256GB
- **MacBook Air 13 pouces** 128GB
- **lot de 2 iPad Wi-Fi 128GB** 9,7"
- **lot de 2 Apple Watch** 44 mm

(Détails et **autres matériels** sur www.pcsoft.fr)

Atelier de Génie Logiciel Professionnel cross-plateformes

iPhone X S

WWW.PCSOFT.FR

COMMUNAUTÉS

4/12 : meetup Drupal / communauté de Marseille
5/12 : React Native & IoT / DCMA Nice
5/12 : l'après JQuery vers VueJS & React / BrestJS
5/12 : journée Agile Tour / Paris
6/12 : sécurité dans Azure / AzugFR
6/12 : soirée C++ / C++ User Group de Sophia Antipolis
7/12 : A 41 ans, je suis devenu Data Scientist / Meetup Machine Learning Pau
10/12 : RUST embarqué / communauté embarqué & Android de Nantes
11/12 : soirée langages pour tous / JUG Paris
11/12 : soirée Low Code / communauté Low Code Paris
12/12 : soirée multiplateforme avec Kotlin & Flutter / GDG Mobile de Toulouse
13/12 : Dev & UI / JUG Nantes
15/12 : le MUG Lyon organise un meetup. Pas de précision.
8/1/19 : young blood VI / JUG Paris
15/1/19 : soirée annoncée / JUG Nantes

Le JUG Tours nous annonce :

19/12 : développeurs en marche vers plus de leadership
23/01/19 : au-delà des brokers, Kafka

Le GDG Tours organise 2 meetups :

12/12 : introduction à Android Things
15/01/19 : Développez un capteur IoT connecté à Reddis et repartez avec !

Paris Open Source Summit 2018

5 & 6 décembre : Dock de Paris

C'est LE rendez-vous du monde open source et du logiciel libre. Cette nouvelle édition proposera des dizaines de conférences et notamment un track développeur qui ne pourra que vous plaire ! Programmez! sera sur place durant tout l'évènement.
Site : https://www.opensourcesummit.paris

blague

F & F' sont sur un bateau.

- F tombe à l'eau, que fait F' ?
- Je ne sais pas
- Il dérive

:-)

Girls Can Code !

Les stages Girls Can Code! c'est une semaine d'initiation à la programmation pour les collégiennes et lycéennes. On y apprend les bases de la programmation avec le langage Python, et de quoi se lancer dans des petits projets de son choix ! Les stages sont gratuits, entièrement organisés par des bénévoles de l'association Prologin.
Au programme : apprentissage du code, ateliers pratiques, réseaux. Une occasion de découvrir le métier de développeur et le monde de l'informatique. Une excellente initiative que nous pouvons que soutenir ! Pour en savoir plus : https://gcc.prologin.org/

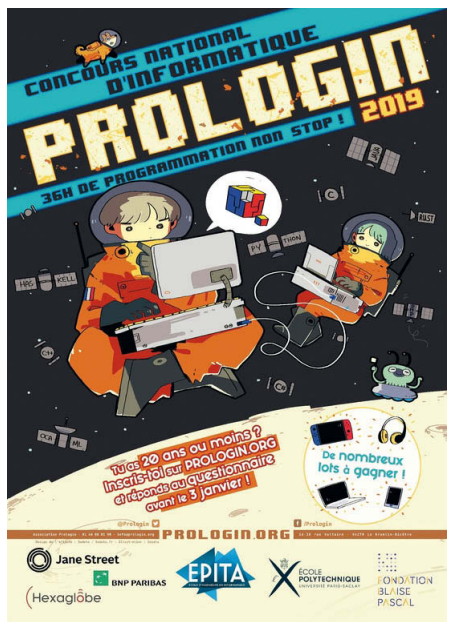
La Nuit de l'info 2018

6 & 7 décembre : dans toute la France !

La Nuit se déroule tous les ans, le premier jeudi du mois de décembre, du coucher du soleil, jusqu'au lever du soleil le lendemain matin.
La Nuit de l'Info est une compétition nationale qui réunit étudiants, enseignants et entreprises pour travailler ensemble sur le développement d'une application web. Les participants ont la durée d'une nuit pour proposer, implémenter et packager une application Web 2.0.
Durant cette nuit, des partenaires lancent des défis (par exemple : interface web la plus ergonomique, meilleure architecture du système, meilleure collaboration, etc.) aux équipes participantes, et proposent des prix pour les équipes ayant le mieux réussi. La nuit est aussi l'occasion de rencontres et discussions avec les entreprises, les ingénieurs et les chefs d'entreprises qui viennent soutenir les étudiants, voire leur donner quelques conseils pour mieux relever les défis.C'est l'évènement étudiant de l'année : +120 écoles, +50 sites, des milliers d'étudiants !
Information : http://www.nuitdelinfo.com

CONCOURS PROLOGIN

Inscrivez-vous avant le 3 janvier 2019 !



Le concours national d'informatique Prologin est ouvert à tous les jeunes de 20 ans et moins. Ce concours entièrement gratuit est l'occasion pour les jeunes passionnés d'informatique de démontrer leurs talents et de rencontrer d'autres passionnés d'informatique. Prologin est organisé par des étudiants de l'EPITA, de l'École Normale Supérieure, ainsi que de l'École Polytechnique. La grande finale se déroulera en mai prochain à l'EPITA Paris : 36 heures de code pour les 100 meilleurs candidats !
Pour participer, il suffit de remplir le questionnaire sur https://prologin.org avant le 3 janvier 2019.

DevCon #7

13/Décembre/2018

INFRASTRUCTURE AS CODE

Pourquoi coder son infrastructure ?

- 2 KEYNOTES
- 4 SESSIONS TECHNIQUES
- 2 QUICKIES
- 1 PIZZA PARTY

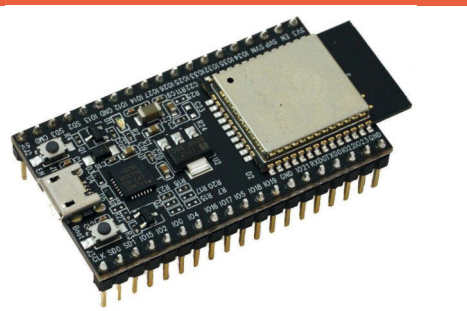
Conférence organisée par

[Programmez!] Le magazine des développeurs

Où ?

42

INFORMATIONS & INSCRIPTION SUR WWW.PROGRAMMEZ.COM



La bombe ESP32

L'ESP32 se présente comme le successeur des ESP8266. L'avantage de l'ESP est son prix et sa capacité WiFi par défaut. Les ESP32 ne sont pas totalement nouveaux sur le marché, mais les prix dépassaient parfois largement 15-20 €. Mais depuis quelques mois, les prix tombent et l'ESP32 devient enfin intéressant.

L'ESP32 a été développé par Espressif Systems. Il s'agit d'un SoC intégrant le CPU 32 bits à 1 ou 2 cœurs, le WiFi et le Bluetooth. Généralement, le CPU tourne à 160 ou 240 MHz. On dispose des SPI, I2C, I2S, UART, des capteurs tactiles, capteur de température, capteur effet hall, un secure boot. Il existe plusieurs versions du SoC. Actuellement, les modèles ESP32-WROOM et WROVER sont les plus communs. La série WROOM intègre le SoC ESP32, de la mémoire flash, des capteurs, des antennes optimisées sur le PCB. Généralement, on bénéficie de 4 Mo de mémoire flash. Sur la partie sécurité, la plateforme supporte par défaut : AES, SHA-2 RSA, ECC et RNG. Avantage aussi de l'ESP32 : la plupart des broches sont PWM, mais, la tension reste à 3,3V ce qui peut être un inconvénient pour certains capteurs.

ESP-32 d'AZ-Delivery

D'aspect, la carte est « massive », une Raspberry Pi Zero réduite, les headers sont soudés. La fabrication est propre, les soudures nettes. Les bords des headers sont parfois à la limite des trous de fixation. La board est livrée nue.

Cette carte utilise un ESP-WROOM à 160 MHz, 512 Ko de mémoire vive et 16 mo de stockage flash ! On dispose de 32 I/O (digital uniquement) + 6 pins analogiques, 3 UART, 2 SPI, 2 I2C, WiFi 802.11 et Bluetooth 4.2 + BLE. Plus précisément, le modèle pris est un ESP32 DEVKITV1.

Le modèle de développement est large : compatibilité avec Arduino, micropython, LUA. Côté outils, le choix est large : Arduino IDE, Espressif Tools, IDE Python, Espressif IoT. La carte fonctionne avec Linux, macOS et Windows. Il faut installer les pilotes CP210x (Silicon Labs).

On flashe le firmware

Certaines cartes sont flashées par défaut en micropython, mais la plupart non. Plusieurs solutions pour flasher le firmware :

- En ligne de commande (via esptool) ;
- Flash Download Tool (Espressive) ;
- uPyCraft.

Attention : sur Windows, avec uPyCraft, vous aurez peut-être une erreur VCOMP100.DLL au lancement. Il faudra installer (ou réinstaller) VC++ Redistributable.

Python doit être installé sur votre machine, en v3.x. (la version dépend de l'outil). D'autres librairies peuvent être demandées comme PyQt.

L'avantage d'utiliser un outil comme uPyCraft est que l'on dispose d'un IDE et surtout des fonctions firmware (micropython) pour faire le burning. L'IDE supporte les ESP8266 et ESP32. Certains modèles peuvent ne pas fonctionner avec l'IDE. Flash Download Tool (Espressive) permet uniquement de flasher le firmware (n'importe quel firmware et pas uniquement du micropython). Il est léger et très rapide.

ESP32 & Arduino IDE

Les ESP sont compatibles avec Arduino IDE. Pour rajouter les ESP32 dans l'IDE, le plus simple, et surtout le plus rapide, est de passer par le gestionnaire des cartes. Au préalable, dans les préférences, indiquer l'URL suivante dans Additional Boards Manager URL : https://dl.espressif.com/dl/package_esp32_index.json puis cliquer sur OK.

Dans Tools -> Board -> Boards Manager, saisissez ESP32, sélectionnez la carte affichée (esp32 by Espressif Systems) puis cliquez sur Install. Le process prend quelques minutes.

Attention : Python et esptool.py doivent être installés sur votre système. Le chargement du code depuis l'IDE utilise esptool. Si la carte n'est pas trouvée par l'IDE, vérifiez bien qu'un firmware est installé sur la carte ou changer de câble USB. La programmation est identique à l'Arduino.

WiFi

Comme la carte est WiFi par défaut, vous pouvez la transformer en mini serveur web ! La partie WiFi est d'une facilité déconcertante. Les éléments du code seront :

```
// on charge la librairie WiFi
#include <WiFi.h>

// on renseigne les connexions au réseau
const char* ssid    = "TOTO";
const char* password = "1234";

// on attribue le port 80 au serveur
WiFiServer server(80);
```

Dans la partie setup :

```
// on se connecte au réseau WiFi avec les ssid et password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
// on récupère l'adresse IP de la board et on lance le  serveur
Serial.println(WiFi.localIP());
server.begin();
```

Dans la partie loop, on code la disponibilité ou non du client web, puis on code la page HTML qui va servir d'interface par exemple pour afficher des données de capteur ou activer / désactiver des capteurs.

Une fois le code téléversé, on affiche le moniteur série pour récupérer l'adresse IP. On lance le navigateur, on renseigne l'IP et là, on accède à la page HTML liée à notre projet.

ESP32 vs la concurrence

L'ESP32 remplacera avantageusement vos ESP8266 et Arduino. Attention : la migration 8266 vers 32 n'est pas forcément simple : outre le code qu'il faut vérifier, il y aura sans doute un remappage des pins à faire. Mais difficile de la comparer à une Raspberry Pi et équivalent. La comparaison peut se faire sur la Pi Zero W et les cartes du même genre. L'avantage de l'ESP est qu'elle est facilement trouvable.

L'ESP32 aura l'avantage de sa taille et de sa consommation. On me dira : OK, mais la Pi Zero W propose plus de ports et un vrai système. Certes, mais dans un IoT, pour moi, ce ne sont pas des arguments décisifs. Personnellement, je regarde la compacité de la board, la consommation, les GPIO.

•

Les +

- Puissance brute
- GPIO
- Capteurs intégrés
- WiFi + Bluetooth
- Tarif attrayant
- Documentation
- Variété des firmwares
- MicroPython

Les –

- Beaucoup de modèles
- Les subtilités entre les modèles pas toujours claires
- Pins 8266 – 32 pas identiques
- Code ESP8266 à adapter (un peu)
- Prix au-dessus des ESP8266

A PARAITRE EN DÉCEMBRE 2018

LE CADEAU GEEK IDÉAL

Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007

9,99 €
(+ 3 € de frais postaux)

116 pages - Format magazine A4

☐ **Découvrez l'âge d'or des micro-ordinateurs de 1973 à 2007** : 9,90 € (+3 € de frais postaux) = 12,90 €

PROG 224

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

CRYOPDP
an Air Liquide company

CRYOPDP crée son ERP avec WEBDEV

(GROUPE AIR LIQUIDE)

CRYOPDP est leader mondial dans le transport et la logistique en température dirigée (produits cryogéniques). La DSI a réalisé son ERP en un temps record grâce à l'AGL WEBDEV.

En tant que leader de la logistique du froid, CRYOPDP fournit des produits cryogéniques et propose des solutions logistiques sur mesure pour acheminer les échantillons et produits des industries pharmaceutiques et agroalimentaires tout en répondant aux exigences de la chaîne du froid.

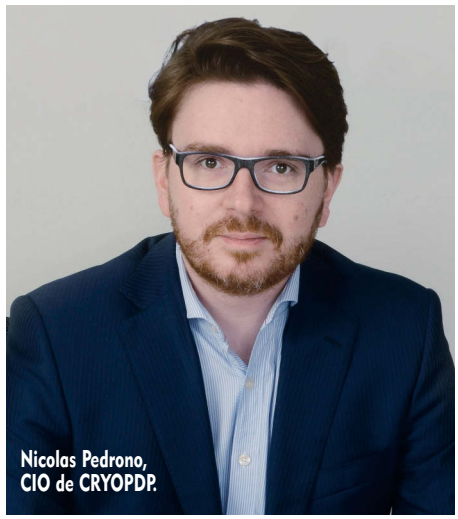
DÉVELOPPER UN ERP EN MOINS DE 9 MOIS ? DÉFI RELEVÉ AVEC WEBDEV !

Dans ce contexte, la DSI a eu pour objectif initial de rationaliser tous les outils informatiques hétérogènes. Sa mission a ainsi été de proposer une solution innovante gérant tout le cycle de gestion d'un ERP et ainsi optimiser la satisfaction de ses clients.

« Pour la réalisation de notre ERP, nous avons choisi WEBDEV, car c'est un AGL basé sur les mêmes qualités de productivité que WINDEV (célèbre AGL de PC SOFT) que nous connaissions déjà », déclare Nicolas Pedrono, CIO de CRYOPDP. Le projet a été réalisé en collaboration étroite avec le service Qualité, ce qui a permis de le développer intégralement avec une vision client respectant les normalisations ISO et GxP imposées par CRYOPDP.

Et d'ajouter : « Nous avons abordé le projet avec une méthodologie Agile, précisément Scrum. WEBDEV nous a permis grâce à sa conception très facile et Wysiwyg des pages de présenter rapidement une solution opérationnelle. Des premiers POC à la mise à disposition de l'ERP, il s'est effectivement écoulé moins de 9 mois».

L'ERP maison intègre notamment une gestion


Nicolas Pedrono,
CIO de CRYOPDP.

commerciale, une gestion de production, un système de suivi d'exploitation ou encore une gestion des stocks.

WEBSERVICES CRÉÉS AVEC WEBDEV ET HÉBERGÉS CHEZ AMAZON

Dans le cadre de ce projet, la DSI a défini comme priorité la centralisation des données pour répondre aux attentes des clients avec une traçabilité totale. Et pour répondre à ce besoin, le choix de SQL server a été retenu dans un environnement Cloud. « D'un point de vue architectural, nous utilisons une solution Cloud hybride hébergée chez Amazon. Nous avons ainsi mis en place un système basé sur des Webservices Rest créés en WLangage depuis WEBDEV. Ils permettent de communiquer avec l'ensemble des prestataires ou fournisseurs». Aujourd'hui cela représente plus de 100.000 consommations de Webservices par jour pour le tracking temps réel des colis (« via un lien natif avec Google map », précise le CIO) et la relation avec les principaux transporteurs tels Chronopost, TNT, DHL ou Fedex. CRYOPDP met également à disposition une solution mobile native sous iOS et Android pour le tracking des colis. Le tout a été développé avec l'AGL WINDEV Mobile. « Nous avons capitalisé un maximum avec un code unique, toujours en WLangage», se réjouit le Nicolas Pedrono.

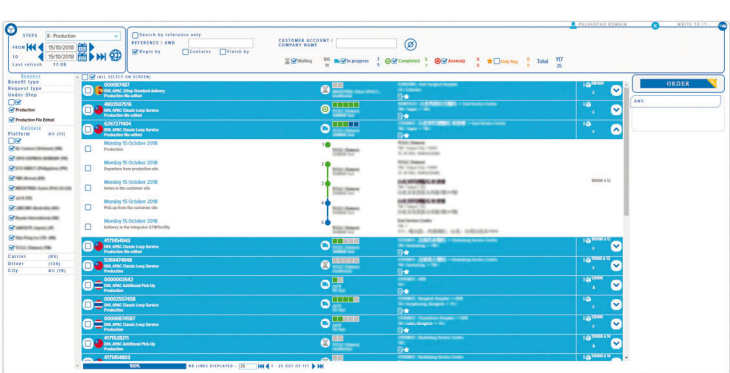
UML, DAO & POO : WEBDEV AU TOP DES MÉTHOLOGIES

Au niveau méthodologie de développement, l'équipe s'est appuyée sur des modèles UML, une conception DAO et la programmation orientée objet en WLangage. « Programmer avec ce L5G est un atout précieux en terme de productivité et de flexibilité », confie Nicolas Pedrono. Et de poursuivre : « A de rares moments, nous avons fait appel à des ressources JavaScript comme jQuery. WEBDEV est top car c'est un AGL ouvert qui permet de s'affranchir de certaines technologies du Web souvent complexes. Et le code WLangage est tellement plus facile à maintenir».

DÉPLOIEMENT INTERNATIONAL SUR 25 SITES

Aujourd'hui, la filiale de Air Liquide a déployé son ERP sur 25 sites à travers le monde (Europe, Etats-Unis, Asie) en s'appuyant sur les fonctions de multilinguisme intégrées à WEBDEV.

« Jamais, nous n'aurions pu relever ce challenge de développer un ERP en moins de 9 mois sans WEBDEV », conclut fièrement le CIO de CRYOPDP.



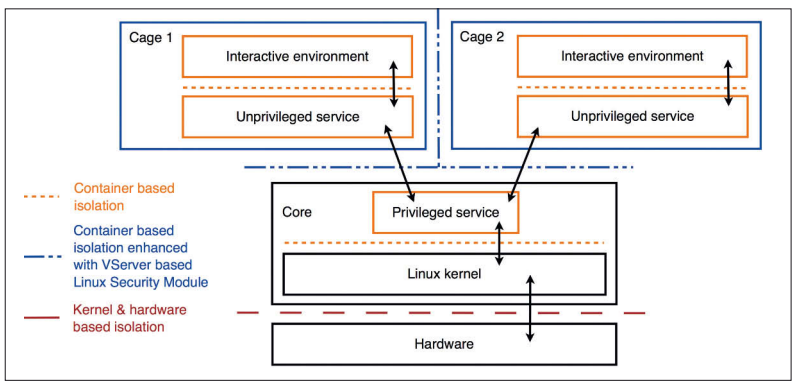
François Tonic

Quelques remarques sur CLIP OS

L'ANSSI a annoncé il y a quelques semaines une première version publique de son OS sécurisé. Il se base sur une fondation Linux et plus précisément sur un Gento Hardened. Sur le principe, les concepteurs se sont inspirés de Chromium et Yocto. CLIP OS s'articule autour d'un core et d'un ensemble de conteneurs. Il fonctionne uniquement sur des machines x86-64 avec un boot sécurisé UEFI, et une plateforme matérielle de confiance. La version alpha 5 disponible publiquement était fonctionnellement limitée à son lancement et de nouvelles fonctions devraient être rajoutées au fur et à mesure. La version dite stable n'est pas encore en développement. L'ANSSI précise que CLIP OS n'est pas un OS souverain. Olivier Cros nous donne son avis.

Version stand-alone

Il n'existe pas de version stand-alone directement utilisable de Clip OS, mais plutôt un vaste ensemble de briques logicielles contribuant à faire de Clip une distribution très orientée sécu. L'ouverture à la communauté des développeurs par le portage du projet en OpenSource peut constituer un vecteur de développement intéressant. Il est toutefois clair qu'il n'y a pas (pour le moment) une priorité de l'ANSSI de travailler dans cette direction.



Quelques particularités

- Au niveau OS, les processus liés à chaque VM sont séparés.
- Via un module noyau custom (lsm), Clip permet de définir des utilisateurs (et administrateurs) différents pour chaque VM. Il n'y a donc pas de compte root commun aux droits partagés entre les deux couches.
- Racine montée en read-only, contrôle de l'exécution des fichiers : pas d'écriture + exécution (mutuellement exclusifs) en WXORX (Write XOR Exec). Intérêt : éviter l'import d'outils extérieurs par un attaquant potentiel.
- Téléchargement des MAJ chiffrée (via IPsec).

LES PLUS

- Cloisonnement fort : la séparation entre couche haute et couche basse, assurée au niveau logiciel ET physique, permet de faire se comporter la machine comme une forme de switch KVM. Même si, du point de vue utilisateur, il s'agit d'un simple changement de bureau, l'OS permet réellement de réaliser un switch entre deux VM isolées.
  De même, les périphériques physiques branchés sur la machine sont spécifiquement attribués à une VM précise, et ne peuvent donc pas servir de pont. Idem pour les environnements graphiques, isolés via un serveur VNC (chaque environnement est en réalité un client VNC spécifique).
- Double validation des paquets et MAJ automatiques : l'utilisateur n'intervient pas sur le processus de MAJ. Les MAJ sont déclenchées automatiquement et gérées en tâche de fond pour permettre une correction de failles automatisée et régulière (exception faite de certaines MAJ cruciales nécessitant un reboot). Les paquets sont doublement signés : développeur package + validateur.
  L'ouverture en OpenSource d'un tel projet est une démarche politique globale d'ouverture des ressources numériques, et non pas une volonté spécifiquement imprégnée dans la démarche de conception de l'outil. Pour autant, cela peut représenter une source intéressante d'évolution, à terme, pour proposer des variantes de CLIP OS adaptées à différents contextes. Ceci étant dit, la publication en open source étant récente, il est encore tôt pour se prononcer sur l'ouverture d'un tel OS à des contextes variés.

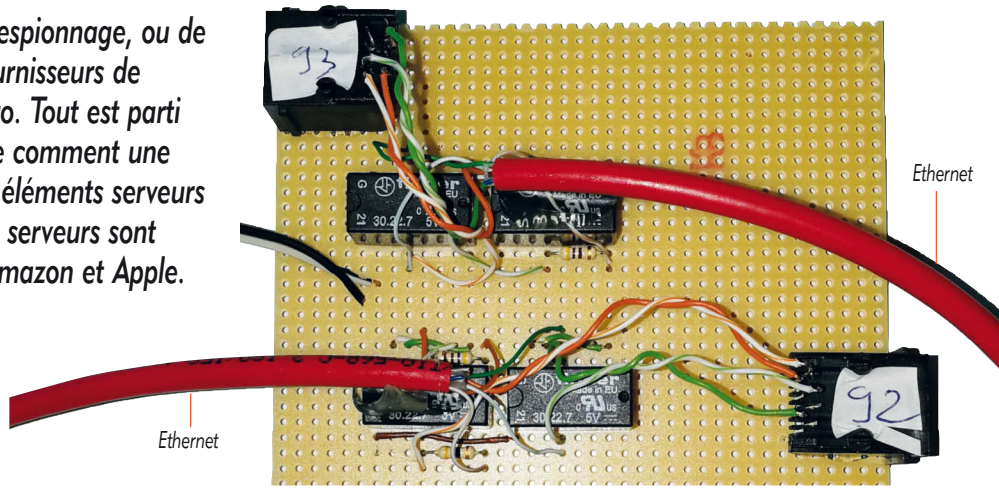
LES MOINS

- L'outil a été rendu Open Source et possède une doc technique solide. De fait, le développement et la participation à la communauté sont possibles et même encouragés. Pour autant, il est difficile de distinguer si le plus pertinent est d'adapter Clip à un besoin précis, ou bien repartir d'une distribution Linux plus classique puis la re-sécuriser (quitte à réutiliser certains modules de Clip).
- Très orienté bureautique et déplacement : les tâches de base sont coûteuses en temps et pour autant la distribution est très orientée utilisateur final (types postes nomades de commerciaux ou d'experts en déplacement). De fait, l'outil est clairement fait pour sécuriser des postes pros, dans le cadre d'une PSSI stricte imposée par les entreprises. Monter Clip sur un ordi perso n'est donc pas très pertinent (bien que possible) si ce n'est pas mis en lien avec une démarche de sécurité forte.
- L'outil a principalement été conçu pour un développement avec support (par l'ANSSI ou autre partenaire). De fait, il y a une grosse phase d'auto-formation nécessaire avant de pouvoir l'utiliser dans un contexte grand public.

François Tonic

Le big hack : pas encore là mais bien présent

Depuis plusieurs semaines, une affaire d'espionnage, ou de hacking c'est selon, secoue les grands fournisseurs de technologies et le constructeur Supermicro. Tout est parti d'une enquête de Bloomberg qui explique comment une minuscule puce est/était installée sur des éléments serveurs construits en Chine pour Supermicro. Ces serveurs sont destinés à plus de 30 fournisseurs dont Amazon et Apple.



Cette affaire réveille de nombreuses inquiétudes. Apple a été rapide à démentir fermement les propos de l'article. Ce dernier repose sur des témoignages et non des éléments définitifs. Qui dit vrai ? Qui dit faux ? Même les autorités américaines ont réagi pour soutenir les fournisseurs technologiques. L'enjeu est colossal.

Cela rappelle des informations qui avaient circulé début 2016 : Apple lançait un projet interne pour désigner ses propres serveurs et éviter toute puce étrangère. Peut-être était-ce motivé par un article d'Ars technica montrant des photos de routeurs Cisco ouverts par la NSA pour y ajouter quelques fonctions maisons. Le matériel était ensuite soigneusement remballé et remis dans le circuit de livraison(1).

Le hack matériel est difficile à détecter et à bloquer

Le hacking matériel (hardware) n'est pas une nouveauté dans le monde de la sécurité et du hack. Le premier objectif est de comprendre le fonctionnement du matériel, avec les schémas électroniques quand ils sont disponibles, ou d'en décrypter le fonctionnement en en analysant le comportement, en décompilant les couches logicielles, etc. C'est le principe de la rétro-ingénierie. Avec les complexités des systèmes, l'intégration du matériel, il faut parfois de la déduction et de bons instruments.

Quand on fouille un peu le sujet, on constate que le hack matériel n'est ni exceptionnel, ni insurmontable. Il existe des kits complets qui peuvent être détournés de leur usage premier. Prenons l'exemple de la carte Bus Pirate. Ce module permet d'analyser des signaux, de découvrir le fonctionnement d'une carte / device inconnue, etc. Son intérêt est son prix (25-30 € sur internet) et son support multi-protocoles (wire, I2C, SPI, UART…). Cette carte est devenue un outil pour faire du hacking matériel, du reverse-engineering. Il va être capable de sniffer les flux des protocoles et vous pourrez repérer les pins nécessaires.

Ce type de hack nécessite, la plupart du temps, un accès aux matériels, surtout quand on utilise une carte Bus Pirate. Quelques (petits) cas d'utilisation de la Bus Pirate sur un routeur :

http://blog.isecurion.com/2017/07/06/dumping-the-firmware-from-the-device-using-buspirate/

http://konukoii.com/blog/2018/02/13/lifting-firmware-with-the-bus-pirate/

Le hacker va préalablement se servir de plusieurs sources pour déterminer le fonctionnement : les schémas électroniques des constructeurs, la documentation et datasheet des puces et composants, observer des PCB et des puces pour repérer les indications (et elles sont parfois nombreuses). Ce travail de recherches préalable est important car on peut gagner de nombreuses heures. On peut aussi regarder les vidéos et tutoriels de démontages qui peuvent apporter des éléments intéressants.

Les frontières poreuses en pentest

Les outils et plateformes évoqués plus haut servent à faire du pentest, c'est-à-dire des tests de pénétration / d'intrusion, avec l'accord du fabricant ou du propriétaire. On force l'accès en simulant une attaque ou en exploitation des failles. Le pentest va permettre d'éprouver son matériel, son logiciel. Il va servir d'audit de sécurité quand ce test est dans cette optique. Bien entendu, tout pentest ayant un but frauduleux est interdit. Le pentest doit se réaliser dans un cadre légal.

Basiquement, le pentest peut aider à :

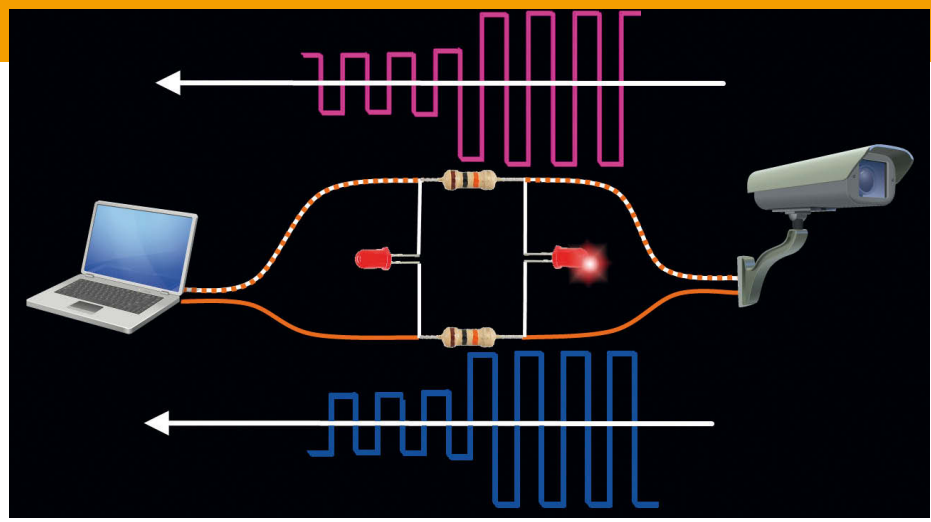
- tester les défenses des environnements ;
- mettre en évidence les failles ;
- savoir si des données critiques / sensibles sont accessibles.

15 minutes, 200 € de matériel : silent wire hacking, une démonstration qui en dit long !

L'événement Hack in Paris est l'une des conférences de sécurité les plus reconnues en France. L'édition 2018 s'est déroulée à Paris au printemps dernier. Nous avons été particulièrement impressionnés par la session de Erwan Broquaire et Pierre-Yves Tanniou. L'intitulé est un programme en soi : silent wire hacking, que l'on pourrait traduire par : le hacking silencieux de câbles.

Ce hack est l'illustration d'un exploit MITM, man in the middle ou attaque de l'homme du milieu. Pour faire simple, il s'agit d'intercepter les communications entre deux parties sans être détecté. En MITM, il est possible de lire, de modifier et d'injecter des messages, des données. Par cet exploit, on peut faire de l'ARP spoofing, du DNS poisoning, de l'analyse de trafic ou encore un déni de service.

(1) https://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/



Comme nous l'ont précisé les speakers, « le hack matériel est faisable à moindre coût avec une solution facilement reproductible, même si cela a été plus compliqué que ce que l'on pensait : par exemple, il faut ruser pour couper la ligne rapidement et sans interruption électrique ». Faut-il des compétences en électronique et en électricité ? Finalement peu. « Entrainez-vous simplement sur du matériel que vous maîtrisez avant de passer au terrain ».

L'idée de base est de se connecter à un câble Ethernet 100 Mb/s, de mettre en place le MITM, sans que cela ne provoque la moindre alerte sur les outils de supervision. « Il n'y pas de réel contre-mesure. Aussi, on préconise d'avoir un chiffrement de bout en bout. » poursuivent les deux speakers. Cette attaque, vise notamment des équipements déployés en extérieur, tels que des caméras de vidéosurveillance. Par contre, cet exploit sera plus difficile à réaliser sur un réseau interne, une salle serveur. Là, les réseaux sont en Ethernet 1 Gb/s, que nous ne traitons pas.

En modèle POE, cet exploit est possible même si sa mise en place est plus complexe : il faut séparer les flux (signal et alimentation), appliquer l'exploit sur la partie signal et redonner de la puissance au signal en sortie via un injecteur POE.

«Lors de notre démonstration, nous repérons le côté émetteur et le côté récepteur pour chaque paire de brins. Pour cela nous utilisons deux diodes et deux résistances. Mais il n'est pas toujours possible d'ajouter une résistance sur la ligne si celle-ci est déjà proche de sa longueur maximale (100m). Dans ce cas, il faut adapter notre dispositif et se servir du câble lui-même en lieu et place de la résistance. » poursuivent les deux experts.

« Aujourd'hui, le protocole 100base-T est encore largement utilisé. Or il existe depuis plus de vingt ans, une époque où on ne se posait pas (moins) de questions sur la sécurité. Au final, nous présentons aujourd'hui une vulnérabilité qui est en fait présente sur les réseaux depuis plus de 20 ans ! » Cette attaque vise un protocole possédant des faiblesses par design et rejoint la famille des exploits tels que l'ARPspoofing, qui existent depuis toujours et ne sont pourtant pas vraiment corrigés (un outil spécifique est nécessaire pour s'en prémunir).

L'attaque consiste donc à faire un MITM sans être détecté par les switches ni les outils de supervisions. Bien que l'attaque ajoute des matériels, elle ne déclenche pas de changement de topologie qui pourrait éveiller les soupçons d'une présence illégale. Elle implique néanmoins des parasites électriques. Mais ils sont minimisés, et le système pourra les considérer comme des parasites réseaux normaux.

Pour réaliser la démonstration, des Raspberry Pi, des relais et de switchs manageables sont nécessaires. « Pour nous, il s'agissait de montrer que cette attaque est reproductible » précisent les experts. Et c'est bien cela qui doit interpeler. Plus largement, la smartcity et les IoT multiplient les risques potentiels et proposent de nouvelles surfaces d'attaque. En IoT, les réglementations ne sont pas toujours claires ou disponibles, les protocoles sont parfois perfectibles.

Etape 1 : on repère et on ouvre un câble (prévoir 30cm environ). On sépare les différents fils (sans les endommager). On place des connecteurs téléphoniques CAD type LSA (parfois appelés « prise vampire »).

Etape 2 : on collecte les informations nécessaires (MAC, IP, vitesse, protocoles utilisés).

N'oubliez pas qu'un câble Ethernet se compose de plusieurs paires de fils torsadées pour émettre (TX) et pour recevoir (RX). En réseau, on ne se soucie plus de câbles droits et croisés. La fonction auto-MDIx permet de détecter automatiquement si le switch doit croiser ou non les flux, selon la topologie réseau. Mais pour le hack le sens de transmission est important ; aussi pour l'identification RX / TX, des diodes sont installées sur les fils de part et d'autre d'une résistance, celle qui s'allume indique le côté émetteur Tx.

Etape 3 : installation et connexion de l'appareillage. Pour injecter du trafic, il faut du matériel supplémentaire : deux switchs manageables (pas nécessaires dans une simple écoute). Il faut connecter la solution sur les différences fils. On branche, on coupe.

Etape 4 : on bascule en position de MITM.

Quand la solution est totalement déployée, les deux entités (par exemple : une caméra et un écran de surveillance) ne sont plus connectées directement. La connexion passe par les deux switchs et tout autre matériel intermédiaire déployé par l'attaquant.

Le temps de commutation des relais (5ms) pourrait être détecté, et nécessite l'ajout d'un signal atténué par deux autres résistances, qui ne sera pas coupé pendant la commutation et considéré comme un simple bruit en fonctionnement normal.

L'usage de Mosfets au-lieu de relais permettrait une commutation plus rapide, mais nécessiterait un montage de commande un peu plus élaboré : *« Nous voulions un montage qui reste très simple et faisable par tous pour être totalement démonstratif.»*

Quelle réaction après la session ? *« Nous avons eu peu de retours d'entreprises, quelques académiques après notre session au HIP18. Nous avons fait également une présentation au congrès Sthack à Bordeaux. Là nous avons eu plus de retours dont une personne qui déployait des caméras le long de clôtures et qui se sentait particulièrement concernée.* » concluent les deux speakers. •

Lien des présentations :

Les slides

https://hackinparis.com/data/slides/2018/talks/HIP2018_Pierre-Yves_tanniou_Silent_Wire_Hacking.pdf

La vidéo

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwjDmvKnwYreAhWIJ8AKHSVJBs4QwqsBMAB6BAgFEAQ&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D3h96ZEE02YQ&usg=AOvVaw0nJz1dmhwMVu1M4bApK_fX

Lien pour l'arpspoofing :

https://www.information-security.fr/attaque-man-in-the-middle-via-arp-spoofing/

Merci à Erwan Broquaire et Pierre-Yves Tanniou

François Tonic

Le Département Maine-et-Loire remplace Google Maps par Openstreetmap et GeoPortail.

Les dernières annonces de Google sur les tarifs et les quotas de Maps ont donné des angoisses à de nombreux développeurs et entreprises. Le Maine–et-Loire ne s'est pas posé de questions. La décision est radicale : on remplace purement et simplement Google Maps…

« Le changement de politique de Google sur les services Maps nous a incité à sauter le pas. Nous avions déjà réfléchi à des alternatives pour les recherches d'itinéraire et les cartes interactives affichées sur nos différents sites. L'annonce a été un déclencheur », David Gatard, développeur.

« Nous n'avons pas réalisé d'étude sur l'impact des coûts. Honnêtement, je ne pense pas que l'on aurait payé plus cher que les quotas gratuits, exceptés 1 ou 2 mois peut-être pour le site inforoute quand il y a des événements majeurs en hiver », précise Céline Pottier, responsable du service Usages numériques et internet.

> // LES TECHNOLOGIES NE SONT PAS RÉVOLUTIONNAIRES MAIS ELLES SONT FIABLES //

Ce n'est pas tant les tarifs que le changement de politique sur Maps opéré par Google qui a déclenché cette migration. Les équipes techniques du Département cherchaient à reprendre le contrôle des données, se rapprocher des principes de l'open source et finalement, ne pas dépendre d'un éditeur.

Plusieurs sites web ont été concernés par cette migration, notamment :

- Le site inforoutes49.fr
- Le site maine-et-loire.fr
- le site assistantsmaternels49.fr

Un double choix mais aussi des contraintes

En réalité, deux technologies ont été choisies : OpenStreetMap et GéoPortail (IGN). La partie IGN est utilisée pour le site de recherche des assistants maternels alors qu'OpenStreetMap l'est pour le site Inforoute et le site web du Département.

Le périmètre fonctionnel a été évalué pour définir les besoins exacts. Sur la plupart des points, les solutions choisies répondent aux besoins et remplacent, à niveau équivalent, Google Maps. « La question s'est tout de même posée sur quelques points précis et particulièrement l'utilisation du format KML qui est utilisé par Google », remarque Grégoire Rondouin, développeur. KML est un format de fichiers permettant d'afficher des données géographiques comme sur Google Earth. Il repose sur des tags et une structure XML. Si à l'origine ce format vient de Google, il dépend aujourd'hui de l'OGC et est un format ouvert. « Comme nous utilisons Leaflet, nous pouvons réutiliser les fichiers KML notamment pour afficher les données sur Inforoute », indique Grégoire.

Leaflet est une librairie open source JavaScript pour créer et afficher des cartes interactives. Elle est très légère et son apprentissage est plutôt rapide.

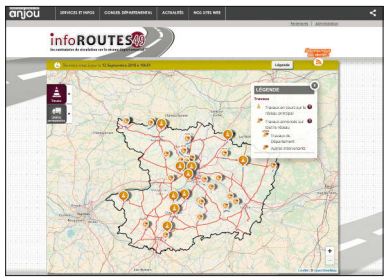
Un des atouts de Google Maps est le calcul des itinéraires et leurs affichages. En comparaison, les technologies utilisées n'offrent pas de résultats aussi pertinents.

Sur le fond de carte, OpenStreetMap ne propose qu'un unique style mais de nombreux providers l'utilisent et le personnalisent. Par exemple, on peut utiliser les cartes de cartoDB pour afficher les cartes.

Une migration courte

Globalement, la migration depuis Google Maps a été plutôt rapide. Il faut à peine quelques heures pour générer une carte avec un marqueur et l'intégrer sur l'un des sites. Le travail sur Inforoute a été plus long car les données sont plus nombreuses et diversifiées, et outre la partie visible par le grand public, il y a également un usage de la cartographie lors de la saisie des informations dans l'administration du site. Là, le projet a duré environ 2 semaines.

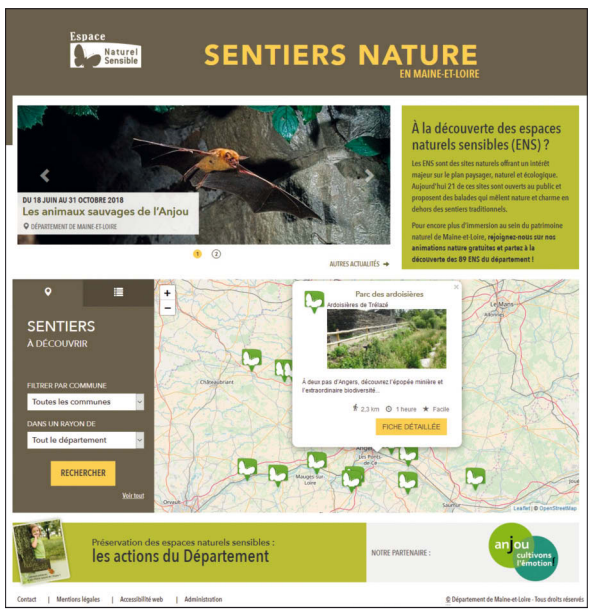
Pour générer et inclure les marqueurs, les développeurs n'ont pas eu d'énormes difficultés. La documentation est propre avec

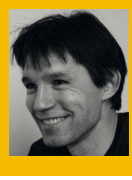


de nombreux exemples. « Finalement, on passe plus de temps à lire la doc qu'à coder », plaisante David. Bien entendu des tests ont été réalisés pour valider les nouvelles cartes. Firefox, Chrome, et IE ont été les principaux navigateurs éprouvés.

Au final

Il reste quelques cartes GMaps sur le site institutionnel du Département, une partie qui n'a pas encore été modifiée car elle intègre le calcul d'itinéraire. Mais à terme, ces cartes seront elles aussi migrées. Au final, les développeurs estiment qu'il a fallu moins de code avec OpenStreetMap et GeoPortal qu'avec Google Maps. Avantage : plus simple à maintenir et code plus lisible. •





Denis Duplan,
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Comment fonctionne la configuration de PhpStorm 2017

PhpStorm est un des nombreux IDE développés par JetBrains, éditeur, entre autres, des excellents WebStorm (développement Web client) et de PyCharm (développement Python).

Pour le développeur, la version 2017 de PhpStorm présente notamment l'intérêt d'être éminemment configurable (Figure 1). Les possibilités offertes touchent à tous les aspects ou presque de cet IDE, tout particulièrement l'interface utilisateur (UI) qui peut être très finement ajustée. Pour autant, la manière de s'y prendre a de quoi dérouter. En effet, à tout instant, il semble possible de modifier un paramètre par plusieurs chemins.

C'est une erreur de conception majeure, sans doute le fruit d'une histoire qu'on espère que JetBrains soldera bientôt. Dans l'immédiat, une exploration des possibilités offertes permet finalement de s'y retrouver et d'adopter PhpStorm en toute sérénité. 1

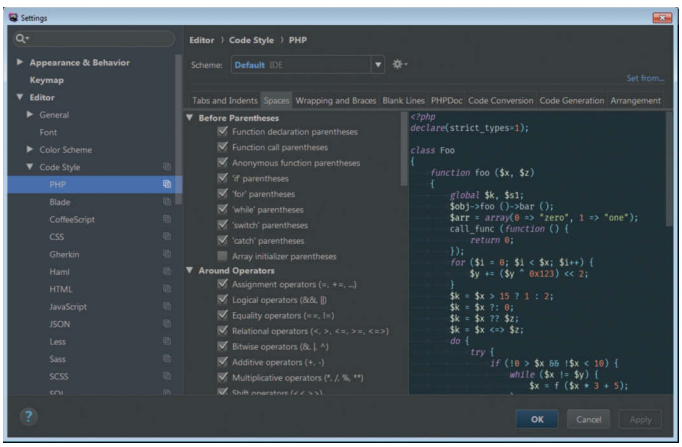


Figure 1 : Page de configuration du code PHP dans PhpStorm 2017.3.

De multiples chemins d'accès à la configuration

Commençons par clarifier la notion de paramètre (setting). Ils sont de deux types :

- ceux qui s'appliquent à l'IDE ;
- ceux qui s'appliquent aux projets.

Par exemple, les paramètres de `Editor / Font` s'appliquent à l'IDE, et les paramètres de `Editor / Code Style` s'appliquent à un projet (on le devine d'après la petite icône se trouvant à sa droite) (Figure 2).

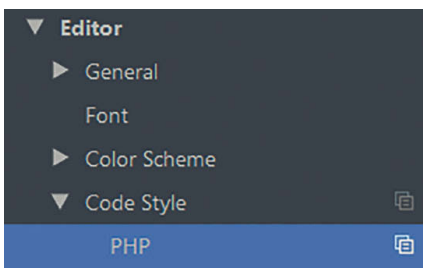


Figure 2 : L'icône signalant un paramètre de projet.

Dans la version 2017.3 de PhpStorm, les paramètres s'appliquant aux projets sont regroupés dans diverses catégories, dont `Appearance and Behaviour`, `Editor` et `Version Control`.

Pourvu qu'un projet soit créé, il n'existe pas moins de trois chemins d'accès pour modifier la configuration de PhpStorm :

- dans l'écran d'accueil, le bouton `Configuration`, puis `Settings` ;
- dans l'éditeur d'un projet, le menu `File`, puis `Settings` ;
- dans l'éditeur d'un projet, le menu File, puis `Default Settings`.

Et encore faut-il comprendre pourquoi certains paramètres sont rattachés à un « profile » ou un « scheme », et d'autres pas.

Pour comprendre comment tout cela fonctionne, rien ne vaut que de procéder à quelques tests. En effet, visualiser les manipulations opérées sur les fichiers de configuration rend beaucoup plus intelligible le fonctionnement de la configuration qui s'appuie dessus.

Quelques tests pour comprendre

Démarrons PhpStorm pour la première fois (la manœuvre peut être rééditée après une installation en supprimant le contenu de `C:\Users\<utilisateur>\.PhpStorm2017.3`). Adoptons le style « Dracula » pour ne pas nous user les yeux et enchaînons directement sur une configuration par défaut. Sur l'écran d'accueil, cliquons sur `Configure` et sélectionnons `Settings` (Figure 3).

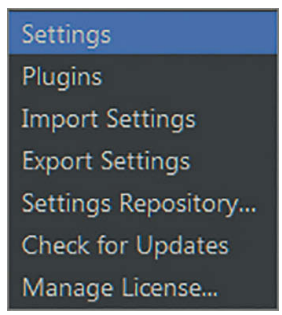


Figure 3 : Accès à la configuration depuis l'écran d'accueil.

Rendons-nous dans `Editor / Code Style / PHP`. Le schéma sélectionné étant `Default` (Figure 4).

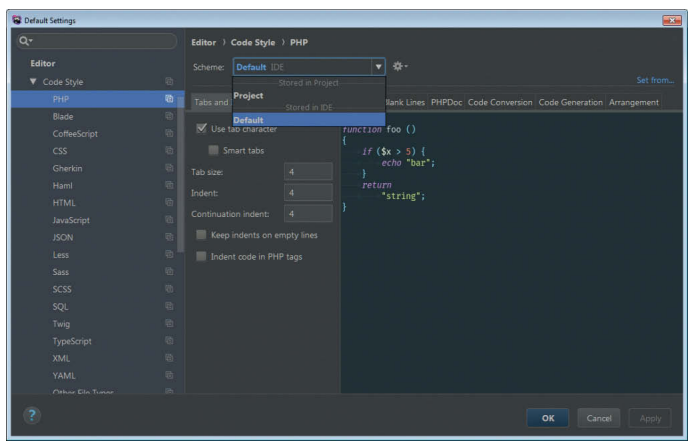


Figure 4 : Configuration de la tabulation pour PHP.

Cochons Use tab character et passons Tab size à 8.
Dans C:\Users\<utilisateur>\.PhpStorm2017.3\config, cela entraîne la création de deux répertoires :
Un répertoire codestyles contenant un fichier Default.xml. Comme on peut s'y attendre, ce fichier contient notamment l'état des paramètres :

```
<option name="TAB_SIZE" value="8" />
<option name="USE_TAB_CHARACTER" value="true" />
```

Un répertoire options contenant toute une série de fichiers, dont un fichier code.style.schemes qui contient notamment ceci :

```
<option name="PREFERRED_PROJECT_CODE_STYLE" value="Default" />
```

Retournons dans Editor / Code Style / PHP. Sélectionnons le schéma Project (Figure 5).

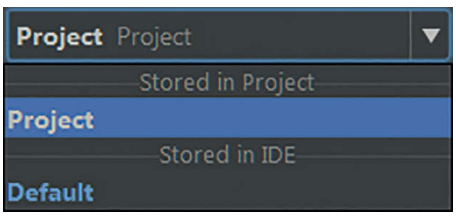


Figure 5 : Sélection du schéma « Project ».

Il apparaît que Use tab character n'est pas coché et que Tab size est à 4.
Cochons Use tab character et passons Tab size à 16.
Dans C:\Users\<utilisateur>\.PhpStorm2017.3\config\options, cela entraîne la modification du fichier code.styles.schemes :

```
<option name="PER_PROJECT_SETTINGS">
  <value version="173">
    <codeStyleSettings language="PHP">
      <indentOptions>
        <option name="TAB_SIZE" value="16" />
        <option name="USE_TAB_CHARACTER" value="true" />
      </indentOptions>
    </codeStyleSettings>
  </value>
</option>

<option name="USE_PER_PROJECT_SETTINGS" value="true" />
```

Retournons dans Editor / Code Style / PHP et sélectionnons simplement le schéma Default. Le fichier code.styles.schemes est de nouveau modifié pour refléter le schéma adopté :

```
<option name="PREFERRED_PROJECT_CODE_STYLE" value="Default" />
```

Là-dessus, créons un nouveau projet à partir d'un répertoire vide et rajoutons-lui un nouveau fichier PHP.
Dans l'éditeur, une tabulation produit bien un caractère tabulation de 8 caractères. C'est donc le schéma Default qui a été pris en compte, comme attendu.
Il apparaît que le répertoire du projet contient un répertoire .idea. Dans ce dernier, on trouve un répertoire codeStyles qui contient deux fichiers :

- codeStyleConfig.xml, qui mentionne que pour les settings de la catégorie code Styles, c'est le schéma Default qu'il faut utiliser :

```
<option name="PREFERRED_PROJECT_CODE_STYLE" value="Default" />
```

Project.xml qui reprend ce qui a déjà été vu dans code.styles.schemes :

```
<codeStyleSettings language="PHP">
  <indentOptions>
    <option name="TAB_SIZE" value="16" />
    <option name="USE_TAB_CHARACTER" value="true" />
  </indentOptions>
</codeStyleSettings>
```

L'IDE donne accès à deux nouvelles possibilités pour modifier les settings : File / Settings (Ctrl + Alt + S) et File / Default Settings.
Sélectionnons File / Settings, et rendons-nous dans Editor / Code Style / PHP (ce qui ne sera plus indiqué par la suite, pour ne pas surcharger cette rédaction). Le schéma sélectionné étant toujours Default, Passons Tab size à 2. Seul le fichier Default.xml est modifié pour refléter l'état des paramètres :

```
<option name="TAB_SIZE" value="2" />
<option name="USE_TAB_CHARACTER" value="true" />
```

Dans l'éditeur, la tabulation occupe désormais 2 caractères.
Sélectionnons de nouveau File / Settings. Cette fois, sélectionnons le schéma Project. Use tab character est cochée, et Tab size est à 16. Contentons-nous de valider. Seul le fichier codeStyleConfig.xml est modifié pour indiquer que c'est le schéma Project qu'il faut désormais utiliser :

```
<option name="USE_PER_PROJECT_SETTINGS" value="true" />
```

De fait, dans l'éditeur, la tabulation occupe désormais 16 caractères.
Sélectionnons de nouveau File / Settings. Le schéma Project étant toujours sélectionné, passons Tab size à 6. Seul le fichier Project.xml est modifié :

```
<option name="TAB_SIZE" value="6" />
```

Dans l'éditeur, la tabulation occupe désormais 6 caractères.
Sélectionnons maintenant File / Default Settings. Il apparaît que le schéma Default est sélectionné, que Use tab character est coché et que Tab size est à 2. Sélectionnons le schéma Project. Il apparaît que Use tab character est coché et que Tab size est à 16.

Lumière sur le fonctionnement de la configuration

Que retenir de tout cela ? Pour ce qui concerne un paramètre qui s'applique aux projets (pour rappel, un paramètre s'applique à l'IDE ou aux projets) :

- modifier le schéma Default via File / Default Settings revient à le modifier via File / Settings ou, depuis l'écran d'accueil, via Configuration / Settings ;
- modifier le schéma Project via File / Default Settings ne revient pas à le modifier via File / Settings, mais revient à le modifier, depuis l'écran d'accueil, via Configuration / Settings.

Lorsqu'un projet est créé, il est doté d'une configuration qui est une copie de la configuration par défaut d'un projet telle que définie via l'écran d'accueil (et éventuellement modifiée via File / Default Settings depuis n'importe quel projet). Cette configuration décrit l'état de paramètres s'appliquant aux projets uniquement. Or il existe deux catégories de paramètres de cette espèce-là :

- **Des paramètres dont la valeur est toujours spécifique au projet.** A partir du moment où ce dernier est créé, toute modification du paramètre ne peut concerner que ce projet spécifiquement, étant décentralisée dans son répertoire .idea\codeStyles. Par exemple, c'est le cas des paramètres de la catégorie Editor / Copyright / Copyright Profiles, comme on peut le deviner en notant la mention For current project en haut de sa page de configuration (Figure 6).

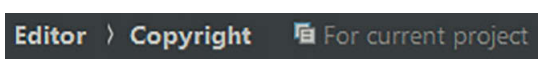


Figure 6 : Signalement d'un paramètre par projet uniquement.

Depuis l'écran d'accueil, il est possible de modifier ce paramètre après avoir créé un projet, mais cela n'affectera jamais la configuration du projet en question. Cette dernière ne pourra jamais être modifiée que depuis le projet, via `File / Settings`.

- **Des paramètres dont la valeur peut être soit spécifique au projet, soit partagée entre tous les projets.** Par exemple, c'est le cas des paramètres de la catégorie `Editor / Code Style / PHP`. Dans la page de configuration, il est possible de sélectionner un schéma (cette notion peut être désignée autrement selon la catégorie considérée ; on parle de profil dans le cas de `Editor / Inspections`, par exemple) pour indiquer si le projet doit utiliser, pour cette catégorie de paramètres et celle-là uniquement, la configuration centralisée au niveau de l'IDE (le répertoire `C:\Users\<utilisateur>\.PhpStorm2017.3\config`) ou celle décentralisée au niveau du projet (son répertoire `.idea\codeStyles`).

D'où le rôle des fichiers dans l'exemple qui a été donné.

Tout d'abord, dans le répertoire `C:\Users\<utilisateur>\.PhpStorm2017.3\config\options` de l'IDE :

- Dans le répertoire `codestyles`, le fichier `Default.xml` contient les valeurs des paramètres du schéma `Default` :

```
<code_scheme name="Default" version="173">
...
</code_scheme>
```

Dans le répertoire options, `code.style.schemes` contient les valeurs des paramètres du schéma `Project` … :

```
<option name="PER_PROJECT_SETTINGS">
...
</option>
```

…et indique si les valeurs des paramètres à utiliser pour tout nouveau projet sont celles-ci ou celles du schéma Default figurant dans le fichier précédent :

```
<option name="USE_PER_PROJECT_SETTINGS" value="true" />
```

ou :

```
<option name="PREFERRED_PROJECT_CODE_STYLE" value="Default" />
```

Ensuite, dans le répertoire `.idea\codeStyles` d'un projet : `codeStyleConfig.xml` indique si les valeurs des paramètres sont celles du schéma `Project` ou du schéma `Default` :

```
<option name="USE_PER_PROJECT_SETTINGS" value="true" />
```

ou :

```
<option name="PREFERRED_PROJECT_CODE_STYLE" value="Default" />
```

`Project.xml` contient les valeurs des paramètres du schéma `Project`, définies lors de la création du projet à partir du fichier `code.style.schemes` et éventuellement modifiées au niveau du projet par la suite (via `File / Settings`, lorsqu'il est ouvert) :

```
<code_scheme name="Project" version="173">
...
</code_scheme>
```

Dans ces conditions, comment modifier la configuration après avoir créé des projets, et faire en sorte que cette modification impacte tous les projets en question ? Eh bien…Ce n'est possible que pour les paramètres dont la catégorie permet une référence à la configuration centralisée au niveau de l'IDE :

- Editor / Code Style (via un schéma) ;
- Editor / Inspections (via un profil) ;
- Editor / File and Code Templates (via un schéma).

On peut le tester. Le projet étant ouvert, sélectionnons `File / Settings` et sélectionnons le schéma `Default` (`Use tab character` coché, `Tab size` à 2). Fermons le projet. Dans l'écran d'accueil, cliquons sur `Configure` puis sélectionnons `Settings`. Le schéma `Default` étant sélectionné, passons `Tab size` à 7 `File / Settings`. Ouvrons de nouveau le projet. Dans l'éditeur, la tabulation occupe 7 caractères.

Cette limitation de la référence à une configuration centralisée a pu susciter l'ire de certains sur le site du support technique de JetBrains (https://intellij-support.jetbrains.com). Toutefois, comme les ingénieurs de JetBrains le rappellent (le support technique est très réactif), il est toujours possible d'exporter une configuration (décentralisée au niveau d'un projet depuis le projet via `File / Export Settings…` ou centralisée au niveau de l'IDE via `Export Settings`) et de l'importer au niveau d'un projet (via `File / Import Settings…`) ou de l'IDE (via `Import Settings`). Cela concerne les catégories de paramètres suivants :

- Code Style ;
- Code Style (schémas) ;
- Debugger ;
- Default Project ;
- Editor Colors ;
- File Templates (schémas) ;
- Look and Feel.

Enfin, il faut noter que lors d'une « session » de modification de paramètres, la modification d'un paramètre dans une page entraîne deux choses :

- un `Reset` apparaît en haut à gauche de cette page pour rétablir les paramètres qu'elle contient dans leur état du début de la « session » (Figure 7) ;

Figure 7 : Réinitialisation d'une page de paramètres.

- les catégories de paramètres dont un paramètre au moins a été modifié sont affichées en bleu dans la liste des catégories (Figure 8).

Figure 8 : Signalement qu'un paramètre a été modifié.

Ces petits plus sont bien pratiques pour qui commence à triturer la configuration de PhpStorm afin de l'adapter à ses besoins.

Un problème de conception qu'on espère bientôt réglé

Somme toute, le problème, c'est que PhpStorm 2017 n'offre pas de point d'entrée sur la configuration qui soit déterminé en fonction des besoins de l'utilisateur.

Comment modifier un paramètre et répercuter cette modification sur un ensemble de projets ? Comment modifier un paramètre et que cela ne vaille que pour tous les projets à venir ? A l'utilisateur de deviner comment s'y prendre pour trouver les réponses à ces questions pourtant élémentaires.

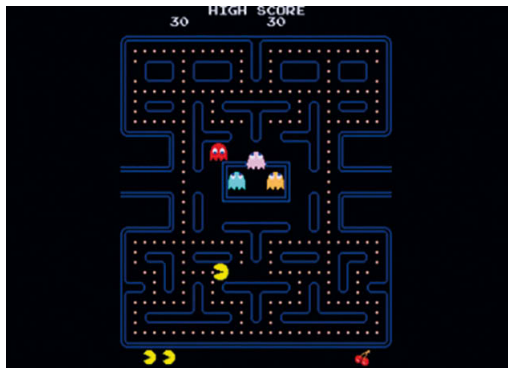
Il faut donc espérer que l'IDE fera l'objet d'un nettoyage pour simplifier sa configuration. En attendant, ce qui vient d'être présenté devrait permettre à n'importe qui de mieux s'y retrouver. PhpStorm est un excellent IDE, et il serait dommage de se laisser rebuter par cette petite difficulté. •

Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com |
www.windowscpp.com

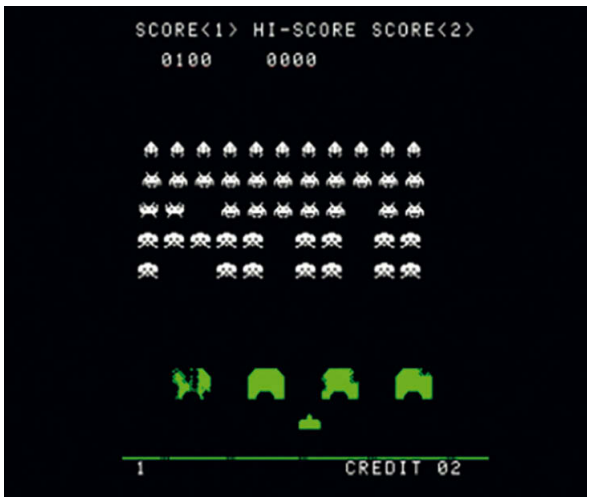
Space Invaders 1978 en C/C++ avec SFML

Ce mois-ci le gaming est à l'honneur. C'est un autre domaine dans lequel les hits sont écrits en C/C++. Nous allons nous intéresser à une bibliothèque qui se nomme SFML (Simple and Fast Multimedia Library). Il existe des bindings pour les autres langages comme Java, C#, Python, etc.

niveau 300



Le jeu d'arcade est un must. Qui n'a jamais tourné de l'œil devant un OutRun, Pacman, AfterBurner ou Space Invaders ?
Parlons de ce dernier, sorti en 1978. Et si nous le recodions pour le fun sous Windows ?



Alors ? Prêt ?

SFML, la boîte à outils

Tout d'abord, il nous faut une bibliothèque qui supporte les objets graphiques animés ou *Sprites*, le chargement de textures ou images, etc. On pourrait faire ça avec les primitives Win32 du système d'exploitation mais SFML encapsule tout ça dans des classes C++ et c'est très bien fait.
SFML est une bibliothèque multimédia, ce qui veut dire quelle fournit une couche entre vous et le hardware. Elle est composée de 5 modules :

- System : c'est le module principal. Il propose les classes de vecteur 2D, 3D, les horloges, les threads, les strings, etc.
- Window : le module Window permet de créer des fenêtres, de gérer l'interaction utilisateur, le clavier et la souris.
- Graphics : ce module fournit les fonctionnalités 2D comme le texte, les images, les formes et les couleurs.
- Audio : SFML propose un module pour la gestion du son.
- Network : ce module permet de créer des jeux en réseau et sur Internet et gère aussi HTTP ou FTP.

Commencer le codage du jeu

Avant tout il faut définir les fichiers d'entêtes nécessaires à utiliser. Nous les obtenons en téléchargeant SFML : https://www.sfml-dev.org/download/sfml/2.5.1/
Ensuite, créons une application console dans Visual Studio. Dans le fichier d'entêtes précompilées, ajoutons ça :

```
#include <stdio.h>
#include <tchar.h>
#include <string>
#include <sstream>
#include <memory>
#include <vector>

#include <SFML/Graphics.hpp>
#include <SFML/OpenGL.hpp>

#pragma comment(lib, "sfml-audio-d.lib")
#pragma comment(lib, "sfml-graphics-d.lib")
#pragma comment(lib, "sfml-main-d.lib")
#pragma comment(lib, "sfml-system-d.lib")
#pragma comment(lib, "sfml-window-d.lib")
#pragma comment(lib, "opengl32.lib")
```

On y inclut Graphics.hpp et c'est OK. On y inclut aussi OpenGL.hpp car le socle technique c'est OpenGL. FSML est multi-plateforme. Les directives pragma comment lib indiquent au linker la liste des librairies qu'il faut lier à l'application. On y retrouve les

5 modules de SFML et openGL. Commençons par écrire le squelette de l'application : une routine main et une boucle principale.

```
int main()
{
    Game game;
    game.run();
}
```

Voyons le constructeur de la classe Game et les variables statiques :

```
const float Game::PlayerSpeed = 100.f;
const sf::Time Game::TimePerFrame = sf::seconds(1.f / 60.f);

Game::Game()
    : m_Window(sf::VideoMode(840, 600), "Space Invaders 1978", sf::Style::Close)
    , m_Texture()
    , m_Player()
    , m_Font()
    , m_StatisticsText()
    , m_StatisticsUpdateTime()
    , m_StatisticsNumFrames(0)
    , m_IsMovingUp(false)
    , m_IsMovingDown(false)
    , m_IsMovingRight(false)
    , m_IsMovingLeft(false)
{
    m_Window.setFramerateLimit(160);

    BuildGameElements();
}
```

Le constructeur positionne la taille de la fenêtre principale.
La méthode BuildGameElements() construit le vaisseau du joueur, les blocs, les ennemis.

```
void Game::BuildGameElements()
{
    m_Texture.loadFromFile("Media/Textures/SI_Player.png");
    m_Player.setTexture(m_Texture);
    m_Player.setPosition(100.f, 500.f);

    std::shared_ptr<Entity> player = std::make_shared<Entity>();
    player->m_sprite = m_Player;
    player->m_type = EntityType::player;
    player->m_size = m_Texture.getSize();
    player->m_position = m_Player.getPosition();
    EntityManager::m_Entities.push_back(player);

// Construction of enemies
    m_TextureEnemy.loadFromFile("Media/Textures/SI_Enemy.png");
    for (int i = 0; i < SPRITE_COUNT_X; i++)
    {
        for (int j = 0; j < SPRITE_COUNT_Y; j++)
        {
            m_Enemy[i][j].setTexture(m_TextureEnemy);
            m_Enemy[i][j].setPosition(
100.f + 50.f * (i + 1),
0.f + 50.f * (j + 1));

            std::shared_ptr<Entity> se = std::make_shared<Entity>();
            se->m_sprite = m_Enemy[i][j];
            se->m_type = EntityType::enemy;
            se->m_size = m_TextureEnemy.getSize();
            se->m_position = m_Enemy[i][j].getPosition();
            EntityManager::m_Entities.push_back(se);
        }
    }

// Construction of blocks
    m_TextureBlock.loadFromFile("Media/Textures/SI_Block.png");
    for (int i = 0; i < BLOCK_COUNT; i++)
    {
        m_Block[i].setTexture(m_TextureBlock);
        m_Block[i].setPosition(0.f + 150.f * (i + 1), 350.f);

        std::shared_ptr<Entity> sb = std::make_shared<Entity>();
        sb->m_sprite = m_Block[i];
        sb->m_type = EntityType::block;
        sb->m_size = m_TextureBlock.getSize();
        sb->m_position = m_Block[i].getPosition();
        EntityManager::m_Entities.push_back(sb);
    }

    m_Font.loadFromFile("Media/Sansation.ttf");
    m_StatisticsText.setFont(m_Font);
    m_StatisticsText.setPosition(5.f, 5.f);
    m_StatisticsText.setCharacterSize(10);
}
```

Chaque objet est une Entity et possède un type :

```
enum EntityType
{
    player,
    weapon,
    enemy,
    block
};
```

Voici la class Entity :

```
class Entity
{
public:
    Entity() { };
    ~Entity() { };

public:
    sf::Sprite m_sprite;
    sf::Vector2u m_size;
    sf::Vector2f m_position;
    EntityType m_type;
    bool m_enabled = true;
```

```
    // Enemy only
    bool m_bLeftToRight = true;
    int m_times = 0;
};
```

Un objet graphique est un Sprite et possède une texture, une taille et une position. La classe Game encapsule le jeu et la méthode Run fait tout. Inspectons cette méthode Run() :

```
void Game::run()
{
    sf::Clock clock;
    sf::Time timeSinceLastUpdate = sf::Time::Zero;
    while (m_Window.isOpen())
    {
        sf::Time elapsedTime = clock.restart();
        timeSinceLastUpdate += elapsedTime;
        while (timeSinceLastUpdate > TimePerFrame)
        {
            timeSinceLastUpdate -= TimePerFrame;

            ProcessEvents();
            Update(TimePerFrame);
        }

        UpdateStatistics(elapsedTime);
        Render();
    }
}
```

En fait la routine principale est toujours la même. Pendant une durée déterminée, on gère les commandes utilisateurs et on anime les objets. De plus, la routine principale gère l'horloge du jeu.
La méthode ProcessEvents gère les touches du clavier ; mouvement du joueur à droite, à gauche ou tir de missile :

```
void Game::ProcessEvents()
{
    sf::Event event;
    while (m_Window.pollEvent(event))
    {
        switch (event.type)
        {
        case sf::Event::KeyPressed:
            HandlePlayerInput(event.key.code, true);
            break;

        case sf::Event::KeyReleased:
            HandlePlayerInput(event.key.code, false);
            break;

        case sf::Event::Closed:
            m_Window.close();
            break;
        }
    }
}
```

Pour bien comprendre la gestion du tir, regardons la méthode HandlePlayerInput() qui est appelée sur la pression/relâche d'une touche :

```
void Game::HandlePlayerInput(sf::Keyboard::Key key, bool isPressed)
{
    if (key == sf::Keyboard::Up)
        m_IsMovingUp = isPressed;
    else if (key == sf::Keyboard::Down)
        m_IsMovingDown = isPressed;
    else if (key == sf::Keyboard::Left)
        m_IsMovingLeft = isPressed;
    else if (key == sf::Keyboard::Right)
        m_IsMovingRight = isPressed;

// Creation of a weapon on SPACE
    if (key == sf::Keyboard::Space)
    {
        if (isPressed == false)
        {
            return;
        }

        std::shared_ptr<Entity> sw = std::make_shared<Entity>();
        sf::Texture texture;
        texture.loadFromFile("Media/Textures/SI_Weapon.png");
        sw->m_sprite.setTexture(texture);
        sw->m_sprite.setPosition(
            EntityManager::GetPlayer()->m_sprite.getPosition().x + EntityManager::
GetPlayer()->m_sprite.getTexture()->getSize().x / 2,
            EntityManager::GetPlayer()->m_sprite.getPosition().y - 10);
        sw->m_type = EntityType::weapon;
        sw->m_size = sw->m_sprite.getTexture()->getSize();
        EntityManager::m_Entities.push_back(sw);
    }
}
```

Si on appuie sur Espace, on crée un objet weapon et on l'ajoute au gestionnaire d'objets EntityManager. EntityManager contient la liste des objets Entity. La méthode Update() gère la gestion des coordonnées et les propriétés des éléments par rapport à l'horloge.

```
void Game::Update(sf::Time elapsedTime)
{
    sf::Vector2f movement(0.f, 0.f);
    if (m_IsMovingUp)
        movement.y -= PlayerSpeed;
    if (m_IsMovingDown)
        movement.y += PlayerSpeed;
    if (m_IsMovingLeft)
        movement.x -= PlayerSpeed;
    if (m_IsMovingRight)
        movement.x += PlayerSpeed;

// On each time event, move the PLAYER

    for (std::shared_ptr<Entity> entity : EntityManager::m_Entities)
```

```
    {
        if (entity->m_enabled == false)
        {
            continue;
        }

        if (entity->m_type != EntityType::player)
        {
            continue;
        }

        entity->m_sprite.move(movement * elapsedTime.asSeconds());
    }
}
```

La méthode Render() se charge d'afficher les éléments du jeu : Weapon, Player, Block, Enemy. Ils sont contenus dans EntityManager.

```
void Game::Render()
{
    m_Window.clear();

    for (std::shared_ptr<Entity> entity : EntityManager::m_Entities)
    {
        if (entity->m_enabled == false)
        {
            continue;
        }

        m_Window.draw(entity->m_sprite);
    }

    m_Window.draw(m_StatisticsText);
    m_Window.display();
}
```

Le cœur du jeu

La méthode UpdateStatistics() contient le moteur du jeu. Regardons le code et je vous explique :

```
void Game::UpdateStatistics(sf::Time elapsedTime)
{
    m_StatisticsUpdateTime += elapsedTime;
    m_StatisticsNumFrames += 1;

    if (m_StatisticsUpdateTime >= sf::seconds(1.0f))
    {
        m_StatisticsText.setString(
            "Frames / Second = " + toString(m_StatisticsNumFrames) + "\n" +
            "Time / Update = " + toString(m_StatisticsUpdateTime.asMicroseconds
() / m_StatisticsNumFrames) + "us");

        m_StatisticsUpdateTime -= sf::seconds(1.0f);
        m_StatisticsNumFrames = 0;
    }

    //
    // Handle collision
    //

    if (m_StatisticsUpdateTime >= sf::seconds(0.050f))
    {
        for (std::shared_ptr<Entity> weapon : EntityManager::m_Entities)
        {
            if (weapon->m_enabled == false)
            {
                continue;
            }

            if (weapon->m_type != EntityType::weapon)
            {
                continue;
            }

            for (std::shared_ptr<Entity> enemy : EntityManager::m_Entities)
            {
                if (enemy->m_type != EntityType::enemy)
                {
                    continue;
                }

                if (enemy->m_enabled == false)
                {
                    continue;
                }


                // Check for collision between weapon and enemies

                sf::FloatRect boundWeapon;
                boundWeapon = weapon->m_sprite.getGlobalBounds();

                sf::FloatRect boundEnemy;
                boundEnemy = enemy->m_sprite.getGlobalBounds();

                if (boundWeapon.intersects(boundEnemy) == true)
                {
                    enemy->m_enabled = false;
                    weapon->m_enabled = false;
                    //break;
                    goto end;
                }
            }
        }

        end:

        //
        // Handle Weapon moves
        //
```

```
for (std::shared_ptr<Entity> entity : EntityManager::m_Entities)
{
    if (entity->m_enabled == false)
    {
        continue;
    }

    if (entity->m_type != EntityType::weapon)
    {
        continue;
    }

    float x, y;
    x = entity->m_sprite.getPosition().x;
    y = entity->m_sprite.getPosition().y;
    y--;

    if (y <= 0)
    {
        entity->m_enabled = false;
    }
    else
    {
        entity->m_sprite.setPosition(x, y);
    }
}

//
// Handle Enemy moves
//

for (std::shared_ptr<Entity> entity : EntityManager::m_Entities)
{
    if (entity->m_enabled == false)
    {
        continue;
    }

    if (entity->m_type != EntityType::enemy)
    {
        continue;
    }

    float x, y;
    x = entity->m_sprite.getPosition().x;
    y = entity->m_sprite.getPosition().y;

    if (entity->m_bLeftToRight == true)
        x++;
    else
        x--;
    entity->m_times++;

    if (entity->m_times>=50)
    {
            if (entity->m_bLeftToRight == true)
            {
                entity->m_bLeftToRight = false;
                entity->m_times = 0;
            }
            else
            {
                entity->m_bLeftToRight = true;
                entity->m_times = 0;
            }
        }

        entity->m_sprite.setPosition(x, y);
    }

  }
}
```

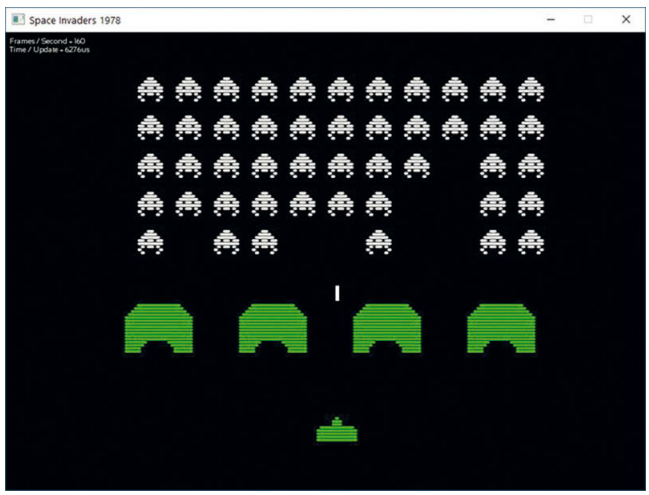
Cette méthode contient 3 parties :

- La gestion des collisions ;
- La gestion des tirs de missiles (weapons) ;
- La gestion du mouvement des ennemis.

Pour gérer la collision entre un weapon et les ennemis, on prend les rectangles de positions disponibles via Sprite::getGloablBounds sur weapon et ennemy et on utilise la méthode FloatRect::intersects. Si elle retourne true, il y a collision et dans notre cas, on positionne le flag m_ennemy des deux éléments, ce qui fait qu'ils ne seront plus affichés. Ce 3 parties sont des routines qui itèrent sur la collection des objets (EntityManager), et, suivant le type des objets, performent des opérations ou modifient des états comme Enabled pour désactiver l'entity.

Le résultat

Voici à quoi ça ressemble :



Conclusion

Le code du jeu est disponible sur GitHub dans https://github.com/ChristophePichaud/SpaceInvaders. Le code n'est pas très compliqué une fois que la boucle principale est comprise. J'espère que cet article vous donne envie de tester SFML en C++ ou en C#.

Python : langage superstar

Incontestablement, Python est LE langage à suivre. Ce n'est pas une réelle nouveauté. Mais pour celles et ceux qui en doutaient encore, il n'y a plus de doute à avoir. Python progresse dans tous les indices de popularités et est régulièrement dans les tops 5 des langages à suivre.

En début 2018, sur GitHub, Python était 2e avec presque 15 % des utilisateurs mensuels, derrière JavaScript (22 %) et devant Java (14 %). Même si la concurrence est très vive, en 2018, les langages comme Go, TypeScript, Kotlin, Rust explosent mais Python reste dans les premiers.

Une des causes de la popularité du langage vient du monde l'éducation. Une partie des ateliers et des professeurs l'utilisent. Les calculatrices des collégiens et lycéens sont compatibles Python ou le seront courant 2019. On code facilement avec le serpent sur Raspberry Pi, les ESP, etc. Dans les universités, cela fait bien longtemps qu'il est présent ainsi que dans les télécoms et les infrastructures IT.

Nous parlons de Python depuis de longues années. Une bonne occasion de lui consacrer un dossier.

Vous allez adorer parler le ***fourchelangue****.*

François Tonic

Développeur Python : un bon profil !

Comme vous vous en doutez, la compétence Python est recherchée et la demande devrait être soutenue dans les prochains mois avec l'accroissement du langage sur le marché. Souvent, le développeur Python connaît d'autres langages ou des frameworks / librairies Python dédiés, qui sont de plus en plus nombreux.

Sans surprise, Python 3 est la référence sur le marché. Et le développeur Python se doit d'avoir des affinités avec les méthodes agiles, ce qui n'est pas une surprise. Malheureusement, de nombreuses annonces demandent des profils bac+5. Malgré tout, un profil expert Python, pas si fréquent que cela sur le marché, trouvera des postes intéressants et motivants. Les bac+2 / +3 sont aussi recherchés.

Côté salaire, nous constatons des fourchettes de rémunérations plus élevées qu'un profil « standard » de développeur. On parle régulièrement d'un salaire de 50 à 60 000 € bruts. Comme toujours, il s'agit de moyenne et cela varie énormément. Pour un profil débutant, la fourchette change peu de la moyenne générale : 2000-2500 € bruts, voire, 3000.

Selon Indeed.fr (début novembre), le salaire moyen en île de France était de 45 500 € bruts. Quand on regarde les salaires annoncés, les disparités sont énormes : 34 000 à 55 000 ! Les chiffres donnés par Urbanlinker ne sont pas très différents : de 34 à +60 000 €. Les niveaux les plus levés sont pour les lead developers et les architectes.

Mais globalement, un développeur Python se « vend » mieux que d'autres compétences plus faciles à trouver. Un expert Python pourra prétendre à un salaire plus élevé que la moyenne. A charge, de se former aux dernières évolutions, mais est-ce encore utile de le dire ? Le niveau de salaire va aussi dépendre du secteur d'activité. Généralement, en big data, analytiques, machine / deep learning.

BAROMÈTRE HIRED RECHERCHE D'EMPLOI

Python s'affirme comme une valeur sûre ; le Ruby revient en grâce au côté de React et Vue.

Alors que Programmez! consacre le numéro de ce mois-ci au Python, il est intéressant de se pencher sur sa place au sein du baromètre mensuel Hired de la recherche d'emploi des développeurs. Si cette technologie a rarement brigué les places d'honneur, force est de constater que la demande pour les candidats qui la maîtrisent est toujours soutenue. En octobre, le Python, qui est mis en avant par 13,22 % des candidats, a ainsi suscité 5,6 demandes d'entretien en moyenne. « Il est intéressant de noter qu'aujourd'hui, le Python est utilisé dans de très nombreux domaines allant des réseaux au web en passant par l'analytique, la data science et la sécurité. La demande est donc toujours très soutenue et surtout constante alors que l'offre est également bien pourvue », déclare Antoine Garnier-Castellane, directeur du bureau France de Hired. Les chiffres des six derniers mois confirment cette impression. Sur cette période, le Python a suscité 5,6 demandes d'entretien en moyenne pour les candidats qui la mettaient en avant. Ces derniers représentent au cours des 6 derniers mois 12,23 % des chercheurs d'emploi inscrits sur Hired ce qui en fait la quatrième technologie la plus mise en avant sur la plateforme.

Parmi les autres technologies ayant marqué le mois d'octobre, React, Ruby et Vue ont suscité respectivement 7,4, 7,2 et 6,4 demandes d'entretiens, s'adjugeant ainsi les trois premières places du baromètre. Si pour le premier et le troisième la demande est restée stable par rapport à septembre, pour le deuxième, elle a en revanche décollé de 2 points. À noter que le nombre de candidats mettant en avant le Ruby sur leur CV a baissé de plus d'un point de septembre à octobre. Quant au Go, qui dominait le classement en septembre, il est relégué à la cinquième place avec 6,2 demandes d'entretiens suscitées en moyenne au mois d'octobre contre 8,6 le mois d'avant. Pourtant, de l'autre côté du classement, .NET, Android et iOS figurent toujours parmi les technologies les moins demandées par les recruteurs en octobre.

Octobre 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	20.25%	React	7.4
Node	14.88%	Ruby	7.2
PHP	10.95%	Vue	6.4
Python	13.22%	Node	6.2
React	10.33%	Go	6.2
Angular	9.09%	Python	5.6
Android	5.37%	DevOps	5.6
.NET	5.17%	Angular	5.0
Ruby	2.89%	PHP	4.6
Go	4.13%	Java	4.3
Vue	2.07%	.NET	3.6
iOS	1.65%	Android	3.1
DevOps	1.03%	iOS	3.1

De Juin à Octobre 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	19.56%	React	7.1
Node	15.20%	DevOps	6.9
PHP	12.76%	Go	6.6
Python	12.23%	Ruby	6.4
React	10.11%	Vue	6.2
Angular	11.04%	Node	6.1
Android	4.82%	Python	5.9
.NET	4.36%	Angular	5.7
Ruby	2.97%	PHP	5.2
Go	2.97%	Java	5.0
Vue	2.18%	.NET	4.3
iOS	1.78%	iOS	3.8
DevOps	1.19%	Android	3.8

HIRED.com est une plateforme technologique de recrutement qui attire et sélectionne les meilleurs profils techniques du marché afin de permettre aux entreprises de recruter des candidats qualifiés rapidement et efficacement. Chaque semaine, entre 80 et 100 nouveaux candidats de la communauté française HIRED en recherche active (développeurs, data scientists, designers, etc.) sont triés sur le volet grâce à des algorithmes et prêts à être contactés…

François Tonic

Un langage qui évolue constamment

L'une des forces du langage est le dynamisme des communautés et de sa capacité à évoluer régulièrement. La librairie standard constitue un socle extrêmement puissant qui possède des centaines de modules. Cette librairie est la base de la programmation Python.

La librairie standard est le cœur du langage, disponible par défaut. Elle contient un très grand nombre de modules accessibles aux développeurs. Cette librairie est par nature extensible. Notons que les modules sont écrits en C. Quand vous installez Python, la librairie standard est installée et vous avez la possibilité de compléter par une collection de modules.

Quand on parcourt la documentation de référence, on comprend mieux toute la diversité de la librairie standard : fonctions, constantes, gestion des textes, données, modules mathématiques, I/O, gestion des fichiers, cryptographie, exécution asynchrone et concurrent, réseau, XML, interface graphique (via Tk), les fonctions debug, le package applicatif, etc.

Il n'est pas toujours facile de comparer les langages entre eux. Côté back-end, on peut l'opposer à PHP, voire, à JavaScript.

Les arguments récurrents en faveur du serpent sont :

- apprentissage facilité par une syntaxe claire ;
- une communauté particulièrement active ;
- une documentation officielle de qualité ;
- une libraire standard étendue et de multiples librairies tierces ;
- un langage mature ;
- un langage qui va aussi pour le web, les apps standards et l'IoT ;
- une version dédiée à l'embarqué : MicroPython.

Une des difficultés - ou est-ce un avantage ? - c'est la possibilité d'utiliser plusieurs solutions pour faire la même chose. Par exemple, pour gérer l'asynchronisme et donc assurer la disponibilité de son application sans qu'une fonction la bloque, vous pourrez utiliser AsyncIO, appartenant à la librairie standard ou un module tiers tel Celery, populaire auprès des développeurs. La gestion des threads est parfois critiquée, notamment à cause du GIL (Global Interpreter Lock). Mais là encore, vous aurez les partisans et les opposants à GIL. Il est possible de contourner le problème GIL avec d'autres solutions telles CPython.

Plusieurs Python

Au-delà de l'opposition, close, entre les versions 2 et 3, Python est disponible sur les cartes IoT et de prototypages de type ESP, Micro:bit, et bien d'autres. Mais il s'agit dans ce cas d'une version dédiée et adaptée aux ressources limitées : la plus connue est MicroPython.

D'emblée, MicroPython (MP) ne supporte pas la librairie standard dans son entier. Il faut minimiser la prise mémoire. Autre exemple, la partie données n'est pas complète : l'objet modèle de données n'est pas présent mais un sous-ensemble allégé, pareillement pour l'introspection et la réflexion, I/O stream incomplet, absence d'Unicode. En principe, la même version de MP est disponible sur l'ensemble des cartes supportées. Cependant, certaines librairies peuvent s'avérer spécifiques à une carte. Il est parfaitement possible d'étendre et d'optimiser MP avec des fonctions C et C++.

Une autre version embarquée est disponible chez AdaFruit, un des fabricants les plus connus du monde Maker et DIY : CircuitPython. CircuitPython est dédié aux microcontrôleurs SAMD21 et ESP8266 et dérive de MicroPython. Il fut créé pour l'éducation en priorité.

Du serpent chez Texas Instruments

TI se prépare à injecter du Python dans la TI-83 Premium CE. Le langage est le langage de référence à partir de la seconde et dans les écoles prépas. Casio a déjà intégré le langage. Côté TI, le support sera effectif au 1er semestre 2019. L'objectif était de garder le matériel actuel et de faire uniquement une mise à jour logicielle. Mais pour pouvoir fonctionner, il est nécessaire de posséder l'adapteur TI-Python qui exécute le code Python. Plus de 80 000 modules ont été distribués. Il sera possible de l'acquérir pour -10 € (tarif estimé). Notons que côté Casio, ce module n'est pas nécessaire. On disposera d'un éditeur de code, d'un

PYTHON 2 OU PYTHON 3 : Y'A PLUS DE DÉBAT

Durant plusieurs années, le débat faisait rage entre les partisans de Python 2 et de Python 3. La v3 changeait de nombreux éléments de langage et cassait une partie de la compatibilité. Objectif : clarifier le code. Et depuis 10 ans, Python 2 disparaît peu à peu et il n'y a plus de débat à avoir. De nombreuses librairies ne supportent plus les v2 et les nouvelles sont le plus souvent uniquement en v3. D'autre part, le support de la 2.7 prendra fin officiellement le 1er janvier 2020.

Python 3 a changé, comme par exemple :

- Print devient une fonction ;
- gestion du texte améliorée : merci Unicode ;
- nettoyage dans les opérations de comparaisons ;
- réorganisation d'une partie des bibliothèques de base, nommage plus cohérent.

PYPARIS : UNE CONFÉRENCE DYNAMIQUE

C'est la grande conférence française sur le langage. L'édition 2018 s'est déroulée les 14 et 15 novembre dernier à l'école Epita près de Paris. Quatre grands thèmes découpaient l'agenda : les données, le langage et les outils, le web / cloud / DevOps et enfin l'éducation. L'agenda était particulièrement dense avec des sessions particulièrement intéressantes : AsyncIO, les annotations, Pyrser & les DSL, serverless en Python, Machine Learning avec ONNX ou Scikit, Django…

gestionnaire de fichiers et d'un shell pour exécuter les codes. La colorisation sera disponible. La librairie Random sera disponible par défaut. Le constructeur utilise Circuit Python d'Adafruit et non directement Micro Python. Cette première version ne permettra pas de programmer le HUB ou le Rover mais on peut espérer cette possibilité fin 2019 ou courant 2020…

Un agenda chargé

La prochaine version majeure du langage est attendue en octobre 2019 : la 3.8. Concernant la branche 3.7.x, la 3.7.1 est la version en cours. La 3.7.2 devrait arriver mi-décembre. Les améliorations et nouveautés sont proposées dans les PEP (Python Enhancement Proposals).

La 3.8 devrait proposer plusieurs nouveautés intéressantes dont :

- Retrait ou dépréciation : Pyvenv, doctype() dans XMLParser, module Gettext, Ast.
- Des améliorations : Gzip, AsyncIO (Windows), nouvelle option dans json.tool, nouvelles méthodes dans Tkinter, etc. •

INTERVIEW

Questions – réponses avec Jean-Bernard Boichat

Ancien développeur, il travaille aujourd'hui beaucoup avec Python et notamment avec PyDev. Il termine un livre sur le Raspberry Pi 3, Python et Java.

Comment êtes-vous arrivé à développer en Python ?

Je ne suis pas allé à Python, ***c'est Python qui est venu à moi.*** Je m'explique. Quand j'ai débuté avec Java dans les années 1990, j'ai choisi Netbeans. Lorsque j'ai développé des applications Java dans ma nouvelle entreprise, je suis venu avec Java et Netbeans. Mes collègues programmeurs Microsoft Basic, C et C++ et Oracle qui sont passés à Java (servlets et autres) m'ont naturellement suivi. Lorsque j'ai changé d'entreprise, j'ai été le premier programmeur Java, ils travaillaient tous sous Linux, C++ et Eclipse. Je suis donc passé en quelques jours à Eclipse, c'est un choix naturel ... et je n'utilise plus Netbeans. Lorsqu'un de mes collègues à cherché une solution scripting sous Linux, il a choisi PHP. Je n'étais pas d'accord, car PHP, c'est plus le Web.

Ensuite une quinzaine de programmeurs C++ sous Eclipse ont dû utiliser PHP. Pas le choix. Quand j'ai dû chercher des scripts pour mettre en place une base de données monstre avec mySQL pour des systèmes embarqués (train et tram et définitions des layouts des afficheurs), j'ai choisi Ruby. La raison était simple : un langage facile et flexible pour ma collègue qui ne connaissait, à peine, que le Visual Basic. Lorsqu'elle nous a quitté, elle m'a remercié : j'ai appris Ruby avec toi, quel plaisir. Donc ce n'est souvent pas un choix, c'est l'environnement ou le système qui le définit. C'est le cas du Raspberry Pi ... et des centaines d'exemples sur le Web en Python. Le Raspberry Pi ... c'est Python. J'ai 5 petits enfants et l'achat d'un robot Lego EV3 était naturel. J'essaie toujours de trouver un environnement pour Python ... mais que de frustration avec l'EV3. Plus de 6000 lectures et des emails de remerciements pour mon article. Mais ce n'est pas stable du tout. Je vais sans doute réessayer. Le logicel LEGO MINDSTORMS Education EV3 n'est pas une direction que je choisirais pour apprendre un langage ! Python est devenu un des langages les plus utilisés y compris dans les écoles. Donc Python est tellement simple pour ce travail dans mon bouquin sur le Raspberry Pi 3 pour tester mes circuits. Même pas besoin de vraiment le connaître, ce Python. Et pas besoin d'un environnement de développement dans ce cas, évidemment ! Notepad++ suffit :-D:-P:-P

Qu'est-ce qui vous plaît le plus dans le langage ?

Sa simplicité pour des choses simples.

Qu'est-ce que vous aimez le moins ?

Sans aucun doute, quand cela se complique. Donc je passe à Java. J'ai des doutes sur comment structurer ces librairies ou encore de travailler en team (C++ et Java n'ont pas ce problème).

On critique souvent le manque d'outils de qualité pour faire du debug ou parfois les difficultés à optimiser un code, avez-vous rencontré ces problèmes ?

PyDev pourrait aider. Python reste un interpréteur donc lent (avec les processeurs d'aujourd'hui, Java suffit). J'ai fait du C++ pendant presque 10 ans.

INTERVIEW

Questions – réponses à Casio France sur le support Python

Dans la Graph 90+E vous supportez micropython. Pourquoi proposer ce langage ?

Nous avons décidé de proposer un menu de programmation en langage Python afin de répondre aux évolutions des programmes scolaires et des besoins des professeurs de mathématiques. Python étant identifié comme langage de référence par l'Éducation nationale.

Vous intégrez micropython et non-python proprement dit : est-il mieux adapté aux contraintes d'une calculatrice ?

Micropython est plus adapté à l'environnement calculatrice. Comme il s'agit d'une version allégée de Python, celle-ci nous apparaît suffisamment complète pour une première version.

Aujourd'hui quelles sont les limites du langage (librairies par exemple) ?

Pour cette 1re version, nous proposons les librairies Math et Random qui nous ont paru indispensables et prioritaires. La calculatrice étant amenée à évoluer, de nouvelles librairies viendront enrichir ce nouveau menu.

D'autres calculatrices seront-elles concernées ?

Seule la calculatrice Graph 90+E est équipée de ce menu. Il s'agit de notre calculatrice -écran couleur pour le lycée - une version avancée de notre modèle phare la Graph 35+E. Il a donc été naturel de faire évoluer ce modèle, car la couleur permet de simplifier la rédaction et la lecture des programmes (coloration syntaxique).

Des nouveautés langages sont-elles attendues comme le support de nouvelles versions ?

La Graph 90+E pouvant être mise à jour gratuitement, l'idée est de l'enrichir dans le temps et de suivre l'évolution des programmes scolaires pour coller au plus juste aux besoins des professeurs et étudiants.

Youssef Bennani
Senior Data Scientist
MARGO
www.margo-group.com

Quelques bases en Python pour la prédiction de séries temporelles

Dans ce tutoriel, nous introduirons quelques concepts élémentaires en séries temporelles afin de pouvoir effectuer "rapidement" des prédictions de valeurs futures sur des données temporelles. Loin d'être exhaustif, ce premier tutoriel présente quelques outils de base en Python permettant d'effectuer de premiers traitements. Le code permettant de retrouver ces résultats est ici : https://gitlab.com/margo-group/public/SeriesTemporelles.

niveau 200

Une série temporelle, ou série chronologique, est une séquence de valeurs numériques indexées dans le temps, souvent avec un même pas de temps séparant deux observations successives.

Le choix du langage Python pour ce tutoriel est plutôt évident. Bien qu'il existe un certain nombre de langages qui offrent les outils nécessaires pour un Data Scientist, le combat en termes de popularité est clairement gagné par Python. (lien : https://www.kdnuggets.com/2017/05/poll-analytics-data-science-machine-learning-software-leaders.html).

En plus du fait que Python a été adopté par les géants de la technologie tels que Google, Facebook, NASA (pour ne citer que ces trois), Python présente d'autres avantages qui ont accéléré son ascension. En effet, sa compatibilité avec les fournisseurs des services du cloud et sa prise en charge du multitraitement pour le calcul parallèle offrent un net avantage par rapport à la concurrence (R ou Java par exemple) en matière de performances à grande échelle.

La figure 1 montre une série temporelle correspondant au chiffre d'affaires (CA) journalier réalisé par un site de vente en ligne durant la première année de son lancement. Cette série nous accompagnera tout au long de ce tutoriel afin d'illustrer les modèles de prédiction que nous présenterons.

L'un des concepts essentiels dans l'étude des séries temporelles est la "stationnarité". Il s'agit de savoir si les observations d'une série temporelle sont générées par une structure* qui change, ou non, avec le temps.

Une façon pratique de vérifier si une série temporelle est stationnaire ou pas est d'étudier l'évolution dans le temps de la structure de sa moyenne, de sa variance et de ses corrélations croisées avec elle-même.

Plus explicitement, si on note $\{st\}_{Tt=1}$ une série temporelle, elle est dite stationnaire** si :

$$E[st]=m<\infty,\forall t,$$
$$V[st]=\sigma 2<\infty,\forall t,$$
$$\gamma(t1,t2)=\gamma(t2-t1),$$

avec :

$$Cov(st1,st2)=E[(st1-m)(st2-m)].$$

Ces trois assertions indiquent que la moyenne, la variance et la dépendance entre les valeurs de la série ont une structure finie et indépendante du temps.

La figure 1 montre que cette série n'est pas stationnaire ; nous remarquons que les niveaux du CA et de sa variabilité évoluent dans le temps. Il existe de nombreux tests statistiques*** pour déterminer si une série est stationnaire ou pas. Il se peut que la moyenne, la variance et les autocorrélations de la série originale ne soient pas constantes dans le temps mais que les statistiques relatives aux changements de la série entre d périodes soient constantes. Et il est donc possible de rendre stationnaire une série non-stationnaire en la différenciant d fois, c'est à dire de considérer la série $\{st-st-d\}$. On parle d'opérateur retard, noté L, appliqué d fois.

Modèle ARIMA

Le modèle ARIMA, appelé aussi modèle de Box-Jenkis, est la combinaison de trois termes : le terme autorégressif (AR), le terme de différenciation (I) et le terme de moyennes mobiles (MA).

Le terme autorégressif suppose que la valeur à l'instant t est une combinaison linéaire des valeurs des instants précédents plus un terme d'erreur.

$$AR(p):st=m+\sum_{pi=1}\phi ist-i+\in t$$

La partie moyennes mobiles suppose que la série est une combinaison linéaire de bruits blancs**** et s'écrit comme suit :

$$MA(q):st=\in t+\sum_{qi=1}\theta i\in t-i$$

Enfin, le terme de différenciation est une application de l'opérateur retard L une ou plusieurs fois.

Ainsi le modèle ARIMA s'écrit :

$$ARIMA(p,d,q):(1-\sum_{pi=1}\phi iLi)(1-L)dst=m+(1+\sum_{qi=1}\theta iLi)\in t)$$

Une version plus détaillée du modèle ARIMA (appelé SARIMA) propose de modéliser la saisonnalité de la série temporelle en ajoutant le paramètre S qui est la période et les paramètres P, D et Q qui correspondent aux facteurs p, d et q de la série différenciée S fois.

Les paramètres p, d, q, S, P, D et Q sont déterminés par optimisation du critère d'information d'Akaike (AIC). Ce critère mesure le compromis entre la complexité d'un modèle et sa qualité d'ajustement.

La figure 2 montre le résultat de la prédiction de la série par un modèle ARIMA sur les derniers 165 jours de l'année (zone rose). On peut remarquer que la prédiction obtenue a un niveau de variance proche de la série d'origine. Cependant cette prédiction rate souvent le sens de l'évolution de la série.

Le 277ème point de la série d'origine est nul. Cela pourrait correspondre à une panne rencontrée sur le site ce jour-là, ou tout simplement à une donnée manquante. A cause de la périodicité calibrée par le modèle ARIMA on remarque que le modèle prédit un pic de baisse de CA pendant les quatre semaines qui suivent le

*On parle de processus stochastique générant les données.

**Plus exactement, la définition donnée ici est celle de la stationnarité faible ou de second ordre.

***On distingue deux types de tests de stationnarité : les tests de racine unitaire comme le test de Dicky-Fuller augmenté et le test de Phillips-Perron où l'hypothèse nulle est que la série n'est pas stationnaire et les tests comme le KPSS où l'hypothèse nulle est que la série est stationnaire.

****Un bruit blanc est la réalisation d'un processus aléatoire dont la variance est constante et la moyenne et l'autocovariance sont nulles.

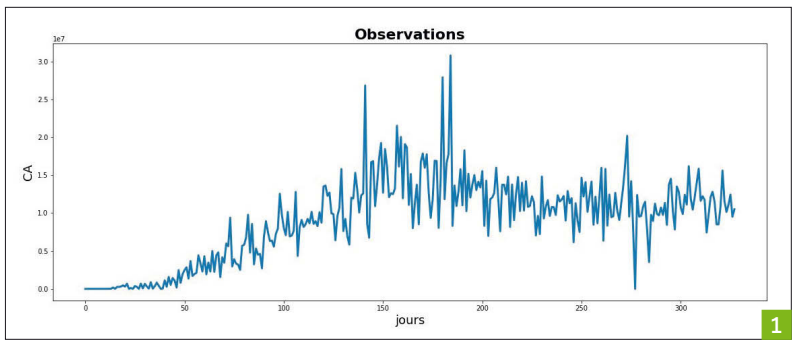
277ème jour de la série. C'est une illustration à la fois du pouvoir de ce modèle et de ses limites.
Dans ce qui suit, nous mettrons l'accent sur un autre modèle de prédiction basé sur les réseaux de neurones.

Réseau de neurones récurrents (RNN)

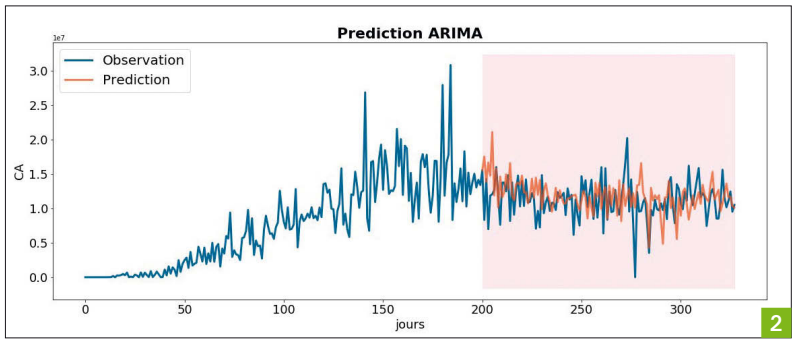
Les réseaux de neurones artificiels sont des systèmes informatiques inspirés des réseaux de neurones biologiques qui constituent les cerveaux biologiques. Ces systèmes apprennent un phénomène en considérant beaucoup d'exemples. Un réseau est basé sur une collection d'unités connectées appelées neurones. Chaque connexion entre deux neurones signifie la possibilité de circulation de signal entre les deux neurones.
Un réseau de neurones récurrents (RNN) est un réseau de neurones artificiels avec des connexions récurrentes, c'est à dire des neurones interconnectés pouvant présenter des cycles. Ces réseaux ont prouvé leur efficacité dans la modélisation de problématiques de reconnaissance automatique de la parole, de l'écriture manuscrite, des formes et dans la prédiction des séries temporelles.
Afin d'évaluer la performance de la prévision par RNN, les données de la série sont séparées en deux sous-ensembles : ensemble de training et ensemble de test. Comme fait précédemment, nous séparons les données des premiers 200 jours du reste de la série.
Une idée simple pour traiter des données temporelles par des RNN est de considérer la question de prédiction comme un problème de régression. En effet, nous réorganisons les données de façons à expliquer une valeur courante de la série par un nombre choisi des valeurs qui lui précèdent. Ce nombre est la taille de la fenêtre par laquelle on regarde la série et qui glisse d'un cran tous les jours.
Le réseau mis en place a pour valeur d'entrée le nombre de retards considérés, une couche cachée avec 3 neurones et une couche de sortie.
Pour évaluer la performance du modèle, on utilise le critère RMSE (root-mean-square error), c'est à dire la racine carrée de la moyenne des carrés des erreurs entre valeurs prédites et valeurs observées. On ajuste le réseau en spécifiant une fonction d'activation et une méthode d'optimisation grâce à la librairie keras. Enfin, le modèle est adapté à l'ensemble de données d'apprentissage. On obtient les prédictions dans la figure 3.
On obtient un RMSE égal à 317722 alors que pour le modèle ARIMA on obtient un RMSE égal à 344723. On remarque donc que la prévision par RNN est plus pertinente que celle par ARIMA.

Réseau de neurones RNN-LTSM

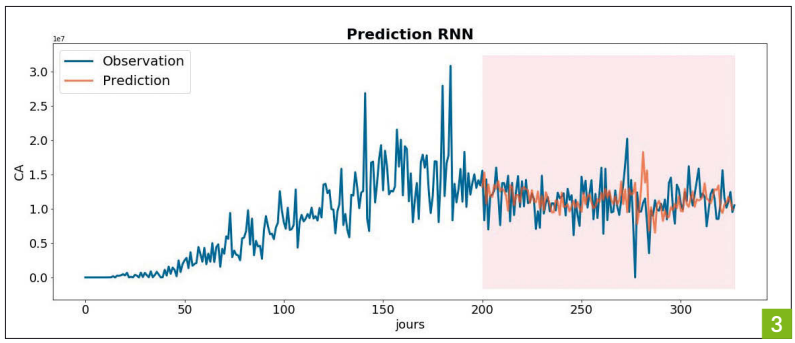
RNN-LSTM (LSTM pour long short term memory) est une architecture de réseau de neurones récurrent qui se souvient des valeurs sur des intervalles arbitraires pendant l'apprentissage. Cette architecture est bien adaptée pour classer, traiter et prédire des séries temporelles avec des décalages temporels de taille et de durée inconnues entre des événements importants.
Le résultat de la prédiction par RNN-LTSM est donné dans la figure 4. Le RMSE correspondant est égal à 294995. Cela indique que le modèle RNN-LTSM dépasse ARIMA et RNN en ce qui concerne la précision de prédiction sur l'ensemble des données de test. Autre fait que nous soulignons ici est que le modèle ne prévoit pas la chute du CA du 277ème jour (ce qui est normal). Par contre il arrive bien à prédire le pic qui survient une semaine après sans que cela n'affecte les prédictions des semaines qui suivent, comme c'était le cas avec le modèle ARIMA.



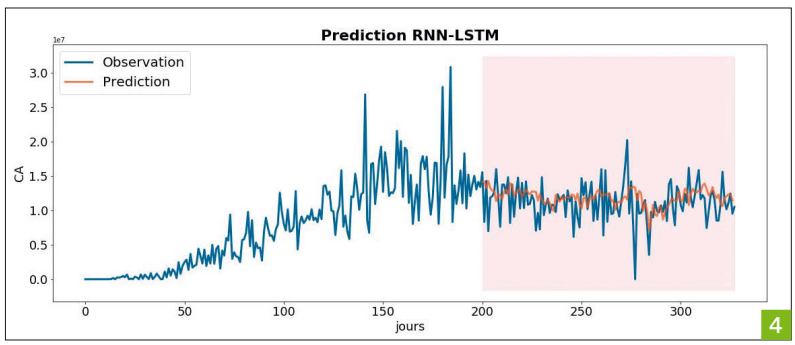
1



2



3



4

Conclusion

Dans cet article (et le code qui l'accompagne disponible ici : https://gitlab.com/margo-group/public/SeriesTemporelles), nous avons montré comment mettre en place les modèle ARIMA, RNN et RNN-LSTM pour prédire les données de séries temporelles en utilisant des librairies Python. Nous constatons que RNN-LSTM offre une meilleure précision sur les données utilisées dans le cadre de ce test. •

Denis Duplan
Sociologue et développeur à ses heures.
http://www.stashofcode.fr

La souplesse des fonctions en Python

Python offre une grande souplesse pour définir et utiliser des fonctions. Concernant la définition, il est possible de recourir à différents types de paramètres. Concernant l'utilisation, il est possible de s'appuyer sur le déballage (unpacking) des paramètres. Petit tour de ce potentiel.

niveau 200

Commençons par nous caler sur le vocabulaire. Selon la terminologie de la documentation Python :

- un paramètre est ce qui figure dans la définition d'une fonction ;
- un argument est une valeur qui figure dans l'appel à cette fonction.

La distinction est importante, car plusieurs arguments peuvent alimenter un même paramètre, comme ce sera expliqué ici.

Les catégories de paramètres d'une fonction

Python permet d'écrire des fonctions comportant de nombreux paramètres qui peuvent avoir des mines des plus patibulaires. Par exemple :

```
def f (subject, *args, **kwargs):
    print (subject, ' '.join (args), kwargs['complement'])
f ('I', 'love', 'the', complement='Amiga')
I love the Amiga
```

La documentation Python contient un passage intéressant sur les fonctions, mais qui peut sembler un peu court au débutant. En particulier, ce passage essentiel peut être lu trop rapidement, étant assez laconique.

*Quand un dernier paramètre formel est présent sous la forme **name, il reçoit un dictionnaire contenant tous les paramètres nommés à l'exception de ceux correspondant à un paramètre formel. Ceci peut être combiné à un paramètre formel sous la forme *name qui lui reçoit un tuple contenant les paramètres positionnés au-delà de la liste des paramètres formels (*name doit être présent avant **name).*

De ce fait, il arrive qu'il y règne une certaine confusion dans les esprits entre ce qui est possible pour définir une fonction et ce qui est possible pour l'appeler.

Par ailleurs, la notion de paramètre formel peut être confondue avec celle de paramètre informel, pour deux raisons :

- Python offre une certaine souplesse dans l'ordre dans lequel les arguments peuvent être mentionnés dans un appel ;
- un paramètre informel peut être nommé, à la manière d'un paramètre formel prenant un argument par défaut.

Pour bien comprendre comment écrire des fonctions en Python, mieux vaut établir une distinction entre paramètre formel et informel, comme le sous-entend le passage de la documentation cité à l'instant.

Les paramètres formels

Une fonction de base ne comprend que des paramètres formels, c'est-à-dire des paramètres identifiés par des noms qui vont permettre de manipuler les valeurs transmises (les arguments) via des variables dans le corps de la fonction :

```
def f (subject, verb, complement):
    return (subject + ' ' + verb + ' ' + complement)
print (f ('I', 'love', 'the Amiga'))
I love the Amiga
```

Un argument peut être une variable ou une constante. Dans le premier cas, la variable est transmise « par valeur » quand elle est de type immuable (la variable n'est pas modifiée au retour de la fonction), et « par adresse » dans le cas contraire (la variable est modifiée au retour de la fonction) :

```
def f (s, a):
    s += ', world!'          # N'affecte pas la chaîne du programme principal, car une chaîne
 est immuable
    a.append (', world!')    # Affecte le tableau du programme principal, car un tableau
n'est pas immuable
s = 'Hello'
a = ['Hello']
f (', world!', a)
print (s)
print (' '.join (a))
Hello
Hello, world!
```

L'argument d'un paramètre formel parvient à la fonction à la position que ce paramètre occupe dans la définition de la fonction. On parle de paramètre formel positionnel.

Tous les paramètres formels ne sont pas positionnels. En effet, un paramètre formel peut prendre un argument par défaut, ce qui permet de mentionner ou non cet argument dans l'appel. Or dans l'appel, l'argument peut être mentionné :

- sans nommer son paramètre, mais il doit alors être mentionné à la position que le paramètre occupe dans la définition, comme l'argument de n'importe quel paramètre formel positionnel ;
- en nommant son paramètre, ce qui permet de mentionner l'argument à n'importe quelle position pour autant que ce soit après les arguments des paramètres formels positionnels.

```
def f (subject='I', verb='have', complement='a computer'):
    print (subject, verb, complement)
f ()
f ('We', complement ='a supercomputer')
f (verb='bought', subject='We')
f (complement ='a megacomputer', 'They')
I have a computer
We have a supercomputer
```

```
We bought a computer
SyntaxError: positional paramètre follows keyword paramètre
```

L'argument par défaut d'un paramètre formel peut être dynamique. Toutefois, cet argument ne sera évalué qu'une fois, au moment où la définition de la fonction sera rencontrée :

```
i = 10
def f (v = i):
    print (v)
f ()
i = 100
f ()
10
10
```

Cette limitation peut être contournée selon le type de l'argument. En effet, l'argument peut être un objet qui évolue :

```
def f (i, v = []):
    v.append (i)       # Rajouter à v la valeur i à chaque appel
    print (v)
f (10)
f (100)
[10]
[10, 100]
```

Aparté sur les opérateurs * et **

Avant d'aborder les paramètres informels, il faut faire un aparté sur les opérateurs * et ** . En Python, * permet d'effectuer une multiplication sur des opérandes numériques, ou à répéter un opérande quelconque, tandis que l'opérateur ** permet d'effectuer une élévation à une puissance quelconque d'un opérande numérique :

```
print (2*3)
print (2*'Hello, world!')
print (2**3)
6
Hello, world!Hello, world!
8
```

Ce qui est moins connu, car elle est assez mal documentée, c'est la possibilité d'utiliser ces opérateurs pour du déballage (*unpacking*) d'itérables divers en vue de constituer un autre itérable.
La documentation relative à cette fonctionnalité est enterrée dans diverses sections relatives aux affichages (*displays*) et aux appels de fonctions (*calls*). Le second point nous intéresse plus que le premier, mais il faut comprendre le premier pour comprendre le second, d'où cet aparté.
Le choix du terme « affichage » (*display*) n'est certainement pas des plus heureux pour désigner un mécanisme qui permet de faciliter la construction d'un itérable. En effet, l'opérateur * permet de déballer un itérable (chaîne, tuple, liste, set, dictionnaire et tout objet itérable) pour en créer un autre. Par exemple, pour donner à voir les différents types d'itérables :

```
a = 'AB'
b = ('C', 'D')
c = ['E', 'F']
d = {'G', 'H'}
e = {0:'I', 1:'J'}
x = *a, *b, *c, *d, *e
print (x)
x = (*a, *b, *c, *d, *e)
print (x)
x = [*a, *b, *c, *d, *e]
print (x)
x = {*a, *b, *c, *d, *e}
print (x)
('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 0, 1)
('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 0, 1)
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 0, 1]
{0, 'G', 1, 'H', 'F', 'A', 'D', 'B', 'C', 'E'}
```

Pour rappel, un objet itérable est un objet dont la classe comporte des méthodes __iter__ () et __next__ . Plutôt que de s'ennuyer à définir ces méthodes, on sait qu'il est plus facile de s'appuyer sur un générateur. Un tel objet peut donc être utilisé comme les précédents pour construire un autre itérable :

```
def f ():
    values = ['A', 'B']
    for i in range (len (values)):
        yield values [i]
a = [*f (), 'C']
print (a)
['A', 'B', 'C']
```

Comme il est possible de le noter dans l'exemple donné plus tôt, le déballage d'un dictionnaire retourne les clés, pas les valeurs. De plus, l'opérateur * ne permet pas de créer un dictionnaire, mais seulement un tuple, une liste ou un set. L'opérateur ** permet de déballer un dictionnaire pour en créer un autre :

```
a = {0:'A', 1:'B'}
b = {'c':'C', 'd':'D'}
x = {**a, **b}
print (x)
{0: 'A', 1: 'B', 'c': 'C', 'd': 'D'}
```

Un itérable peut être déballé dans un itérable qui existe déjà :

```
a= ('B', 'C')
b = ('A', *a, 'D')
print (b)
('A', 'B', 'C', 'D')
```

Il faut bien noter que le déballage d'un itérable via * et ** ne vise qu'à créer un autre itérable. Pour déballer un itérable dans des variables, il faut utiliser la notation classique :

```
a = 'AB'
b, c = *a  # Il fallait écrire : b, c = a
SyntaxError: can't use starred expression here
```

Toutefois, il faut mentionner une syntaxe intermédiaire qui permet de procéder au déballage d'un itérable pour créer un itérable contenant tous les éléments qui n'ont pas pu être déballés dans des variables :

```
a = ('A', 'B', 'C', 'D')
```

```
left, *middle, right = a
print (left)
print (middle)
print (right)
A
['B', 'C']
D
```

Les paramètres informels

En plus de paramètres formels (dont il est donc possible d'omettre l'argument dans l'appel, s'ils prennent un argument par défaut), la définition d'une fonction peut mentionner des paramètres informels, c'est-à-dire des paramètres dont la liste dépend des arguments fournis lors de l'appel.
Dans la définition de la fonction, l'existence éventuelle de tels paramètres informels est mentionnée à l'aide d'un seul paramètre précédé de `*`, qui figure en dernier. Cela signifie que dans l'appel, les arguments qui doivent être réunis dans le tuple doivent être fournis en dernier :

```
def f (subject, verb, *args):
    print (subject, verb, ' '.join (args))
f ('I', 'love', 'the', 'Amiga')
I love the Amiga
```

Attention à ne pas confondre l'usage de `*` dans la définition d'une fonction pour mentionner la présence de paramètres informels, d'une part, et dans l'appel à cette dernière pour déballer des arguments destinés à alimenter des paramètres formels et/ou informels, d'autre part. Il est parfaitement possible d'utiliser l'une et l'autre de ces fonctionnalités simultanément :

```
def f (subject, verb, *args):
    print (subject, verb, ' '.join (args))
words = 'I', 'love', 'the', 'Amiga'        # 'the' et 'Amiga' seront informels
f (*words)
I love the Amiga
```

La règle concernant la position dernière des paramètres informels dans la définition de la fonction souffre une exception. En effet, comme déjà expliqué, il est possible de mentionner la présence de paramètres formels nommés. Or ces paramètres doivent toujours être mentionnés après les paramètres informels dans la définition :

```
def f (subject, *args, verb='love'):
    print (subject, verb, ' '.join (args))
f ('I', 'the', 'Amiga', verb='love')
I love the Amiga
```

Noter que cela n'offre aucune souplesse dans l'appel, car le positionnement des arguments doit respecter celui des paramètres. Autrement dit, les arguments des paramètres formels nommés doivent être fournis en dernier :

```
f ('I', verb='love', 'the', 'Amiga')
SyntaxError: positional paramètre follows keyword paramètre
```

Toutefois, comme un paramètre formel, un paramètre informel peut être nommé - s'agissant d'un paramètre informel, ce sera évidemment lors de l'appel en associant l'argument à un nom. Dans ce cas, une fonction accède aux arguments transmis non via un tuple, mais via un dictionnaire. Dans la définition de la fonction, ce dictionnaire, c'est-à-dire le paramètre qui regroupe les paramètres informels nommés, doit toujours être mentionné en dernier :

```
def (subject, **kwargs):
    print (subject, kwargs['verb'], kwargs['completement']);
f ('I', verb='love', complement='the Amiga')
I love the Amiga
```

Noter qu'il n'y a aucune confusion possible entre les arguments des paramètres formels nommés et ceux des paramètres informels nommés, car la priorité est donnée aux premiers lors de l'appel :

```
def f (subject, verb='love', **kwargs):
    print (subject, verb, kwargs['complement'])
f ('I', verb='love', complement='the Amiga')
I love the Amiga
```

D'ailleurs, cela permet de fournir les arguments des paramètres formels nommés après ceux des paramètres informels nommés :

```
f ('I', complement='the Amiga', verb='love')
I love the Amiga
```

La définition d'une fonction peut mentionner des paramètres informels nommés, et d'autres qui ne le sont pas. Lors d'un appel, les arguments des premiers parviendront à la fonction via un dictionnaire, et les seconds via un tuple :

```
def f (subject, *args, **kwargs):
    print (subject, ' '.join (args), kwargs['complement'])
f ('I', 'love', 'the', complement='Amiga')
I love the Amiga
```

Noter qu'il est impossible de mentionner alors aussi la présence de paramètres formels nommés dans la définition de la fonction. Aucun moyen de faire marcher ça :

```
def f (subject, verb='love', *args, **kwargs):
    print (subject, ' '.join (args), kwargs['complement'])
f ('I', verb='love', 'the', 'famous', complement='Amiga')
```

Faut pas pousser...

Déballage et appel

Pour terminer ce passage en revue des modalités de définition et d'appel à une fonction au regard des différentes catégories de paramètres, signalons qu'il est possible d'appeler une fonction en déballant un itérable pour fournir les arguments. C'est une utilisation singulière des opérateurs `*` et `**`, car nous avons vu qu'ils ne permettent pas de déballer un itérable dans des variables au sens du déballage classique.
Par exemple, pour déballer des arguments alimentant des paramètres formels à partir d'un tuple ou d'un dictionnaire :

```
def f (subject, verb):
    print (subject, verb)
x = 'Amiga', 'rulez!' # Rappel : notation simplifiée pour créer un tuple
f (*x)
x = {'verb':'rulez!', 'subject':'Amiga'}
f (**x)
Amiga rulez!
Amiga rulez!
```

Dans le cas d'un dictionnaire, il convient de noter que les clés doivent reprendre les noms des paramètres formels spécifiés dans la définition de la fonction. De ce fait, il est impossible de transmettre des arguments de paramètres informels anonymes (ceux qui seront regroupés dans le tuple `*args`) à partir d'un dictionnaire, alors que cela est possible à partir d'autres itérables, comme par exemple un tuple :

```
def f (subject, verb, *complements):
    print (subject, verb, ' '.join (complements))
x = 'I', 'love', 'the', 'Amiga'
f (*x)
x = {'verb':'love', 'subject':'I', 'complements':('the', 'Amiga')}
f (**x)
I love the Amiga
NameError: name 'complements' is not defined
```

A l'inverse, il est impossible de transmettre un dictionnaire à partir d'un itérable qui n'est pas un dictionnaire.
La forme la plus complexe d'appel s'appuyant sur le déballage implique donc deux itérables, un premier qui n'est pas un dictionnaire pour fournir les arguments des paramètres formels (nécessairement anonymes si des paramètres informels nommés sont présents, comme nous l'avons vu) et informels anonymes, et un dictionnaire pour fournir les arguments des paramètres informels nommés (au passage, noter l'usage de `*` et `**` sur des littéraux) :

```
def f (subject, *args, **kwargs):
    print (subject, ' '.join (args), kwargs['computer'], kwargs['model'])
f (*('I', 'love', 'the'), **{'computer':'Amiga', 'model':'500'})
I love the Amiga 500
```

Dans ces conditions, on comprend que s'il faut relayer le tuple `*args` et/ou le dictionnaire `**kwargs` d'une fonction à une autre, cela devra se faire en les déballant au passage :

```
def f (*args, **kwargs):
    print (' '.join(args), kwargs['computer'], kwargs['model'])
def g (*args, **kwargs):
    f (*args, **kwargs)      # Déballage pour l'appel
g ('I', 'love', 'the', computer='Amiga', model='500')
I love the Amiga 500
```

Un peu de décoration

En plus de la souplesse offerte pour définir une fonction, Python permet recourir à des décorateurs pour modifier le comportement d'une fonction après qu'elle a été définie.
Un décorateur est une fonction `g()` qui retourne une fonction `h()`, dite wrapper, à la place d'une fonction `f()` lorsque cette dernière est appelée. En ce sens, c'est du sucre syntaxique.
Il est possible de décorer `f()` avec `g()` en précédant la définition de `f()` par `@g` . Le décorateur est appelé lorsque la définition de la fonction qu'il décore est rencontrée. La fonction `h()` retournée par `g()` accède aux paramètres transmis à `f()` en adoptant le même prototype :

```
def g (someFunction):
    def h (computer):
        someFunction ('Amiga' if computer == 'Atari' else computer)
    return h
@g
def f (computer):
    print ('I love the', computer)
f ('Atari')
I love the Amiga
```

Dans l'exemple précédent, le wrapper utilise la fonction décorée, mais il pourrait tout aussi bien ne pas le faire. Tout dépend de la manière dont on veut altérer l'appel à la fonction décorée.
Le décorateur peut accepter des paramètres. Il doit alors retourner une fonction qui fait le travail à sa place. Autrement dit, tout se passe comme si `g()` prenant des paramètres, c'est donc qu'il ne pourrait pas prendre `f()` en paramètre. Partant, `g()` devrait retourner `h()` qui peut prendre `f()` en paramètre, et qui retournerait un wrapper `k()`. Ce wrapper peut faire référence aux paramètres transmis à `g()` comme si c'était des variables locales, car il a accès à l'espace de noms de `g()` :

```
def g (model):
    def h (someFunction):
        def k (computer):
            s = 'Amiga' if computer == 'Atari' else computer
            someFunction (s + ' ' + model)
        return k
    return h
@g ('500')
def f (computer):
    print ('I love the', computer)
I love the Amiga 500
```

Mais si un décorateur est utilisé pour substituer `h()` à `f()` , ce n'est donc plus la même fonction qu'il faut s'attendre à utiliser. En particulier, des métadonnées sur `f()` seront perdues, donc son nom (`__name__`) et sa documentation (`__doc__`) :

```
def g (someFunction):
    def h ():
        """This is function h () docstring"""
        pass
    return h
@g
def f ():
    """This is function f () docstring"""
    pass
print (f.__name__, ":", f.__doc__)
h : This is function h () docstring
```

Le décorateur `wraps()` du package `functools` permet de l'éviter en reprenant dans `h()` ces métadonnées :

```
from functools import wraps
def g (someFunction):
    @wraps (someFunction)
    def h ():
        pass
    return h
@g
def f ():
    pass
```

```
print (f.__name__)
f : This is function f () docstring
```

Pourquoi utiliser `wraps ()` ? Visiblement pour rien d'autre que pour la documentation…
Plusieurs décorateurs peuvent être chaînés :

```
def trashAtari (someFunction):
    def g (computer):
        someFunction ('Amiga' if computer == 'Atari' else computer)
    return g
def capitalizeComputer (someFunction):
    def h (computer):
        someFunction (computer.upper ())
    return h
@trashAtari
@capitalizeComputer
def f (computer):
    print ('I love the', computer)
f ('Atari')
I love the AMIGA
```

Tout comme il est possible de décorer une fonction, il est possible de décorer une méthode d'une classe.
Enfin, des décorateurs sont prédéfinis :

```
@staticmethod
@classmethod
@property
@showdoc
@announce
```

Par exemple, `@classmethod` permet de modifier une méthode de sorte qu'elle reçoive une référence sur la classe et non sur l'instance, et `@staticmethod` permet de modifier une méthode de sorte qu'elle ne reçoive pas de référence sur l'instance :

```
class Thing:
    def f (*args):
        print (args)
    @classmethod
    def g (*args):
        print (args)
    @staticmethod
    def h (*args):
        print (args)
Thing.f ()          # f () ne reçoit rien
Thing.g ()          # g () reçoit une référence sur la classe
Thing.h ()          # h () ne reçoit rien
t = Thing ()
t.f ()              # f () reçoit une référence sur l'instance
t.g ()              # g () reçoit une référence sur la classe
t.h ()              # h () ne reçoit rien
()
(<class '__main__.Thing'>,)
()
(<__main__.Thing object at 0x000000000269CFD0>,)
(<class '__main__.Thing'>,)
()
```

`@classmethod` est utilisé pour créer des constructeurs alternatifs :

```
class Thing:
    def __init__ (self, name):
        self.name = name

    @classmethod
    def fromAnotherThing (cls, t):
        t = cls (t.name)
        return t

t0 = Thing ('The thing')
t1 = Thing.fromAnotherThing (t0)
print (t1.name)
The thing
```

Autre exemple, @staticmethod `@staticmethod` permet de regrouper des méthodes dans une classe tout en indiquant qu'elle n'est donc pas dépendante de la classe : une forme de packaging.
Les occasions d'utiliser @classmethod `@classmethod` et @staticmethod `@staticmethod` sont certainement des plus rares.

Pour faire bref : les lambdas

Il reste à mentionner les lambdas. On parle ici « sugar », c'est-à-dire d'une facilité offerte pour écrire du code. En effet, c'est un moyen de déclarer une fonction sous la forme d'une expression tenant sur une ligne :

```
def f (x):
    return lambda a, b : x + a - b
g = f (1)
f (10, 100)
-89
```

La caractéristique de la fonction ainsi créée est donc qu'elle est anonyme. Comme c'est le cas pour toute fonction, une lambda peut être utilisée comme argument :

```
def f (myLambda, x, y):
    print (myLambda (x, y))
g = lambda x, y : x + y
f (g, 10, 100)
110
```

Conclusion

Python offre une très grande souplesse pour écrire des fonctions. La contrepartie est que la syntaxe peut ne pas toujours sembler des plus claires au profane qui découvre le langage, mais il ne faut surtout pas rester sur cette impression : en s'exerçant un peu, on trouve vite ses repères.
Il n'en reste pas moins que certaines des fonctionnalités qui ont été présentées ne trouvent sans doute leur usage qu'exceptionnellement, ayant été rajoutées au fur et à mesure pour répondre à des besoins nécessairement toujours plus spécifiques.
C'est ce qui fait le charme de Python diront certains, qui apprécient d'avoir le sentiment de ne jamais en avoir épuisé la richesse en découvrant chaque jour de nouvelles évolutions ; c'est le signe d'une incapacité à gérer la complexification du langage, jusqu'à en faire un vrai bazar (*clutter*), diront d'autres qui courent après les évolutions pour en entretenir une connaissance exhaustive… •

Damien Nicolet
Développeur java dans une PME le jour, et « bidouilleur » le reste du temps.

MicroPython : l'autre Python

niveau 100

Python est un langage de programmation interprété, interactif, orienté objet. Il incorpore des modules, des exceptions, et un typage dynamique et des classes.

La PyBoard est une carte de développement électronique compacte qui exécute MicroPython. Elle se connecte à un ordinateur avec un câble micro USB, et donne accès à un lecteur USB et un terminal série Python (un REPL). Elle fonctionne avec Windows, macOS et Linux.

MicroPython est un interpréteur Python 3 destiné à s'exécuter sur des systèmes présentant des ressources limitées, typiquement des microcontrôleurs.

MicroPython ne supporte pas la totalité de la librairie standard Python. Certains modules manquent, car ils ne sont pas applicables aux microcontrôleurs, en raison de consommation des ressources, ou de l'absence de certaines fonctionnalités matérielles.

Il y a plusieurs différences entre CPython 3 (considéré comme l'implémentation de référence du langage Python 3) et MicroPython. Ces différences sont classées en deux catégories.

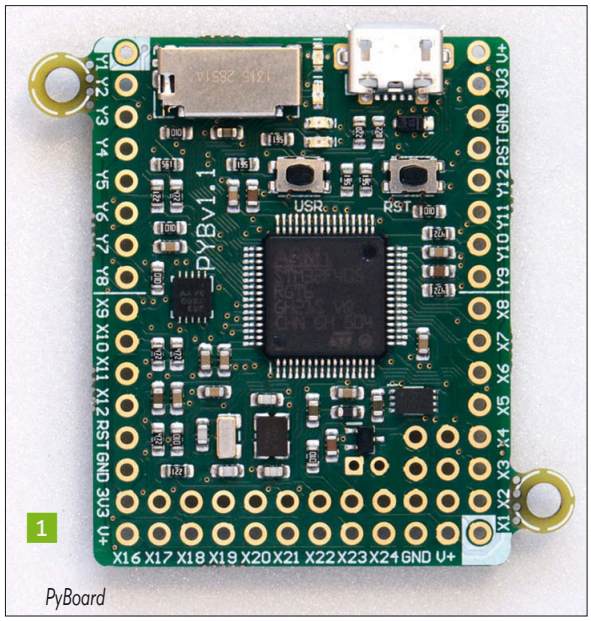
Les différence "par design" : - la librairie standard est volontairement limitée, le projet micropython-lib fournit une bibliothèque standard non monolithique pour MicroPython - MicroPython utilise le garbage collection comme principal moyen de gestion de la mémoire. - MicroPython n'implémente pas le modèle de données d'objet CPython complet, mais seulement un sous-ensemble de celui-ci (les utilisations avancées de l'héritage multiple, la méthode __new__ peuvent ne pas fonctionner. L'ordre de résolution de la méthode est différent. Les métaclasses ne sont pas prises en charge) - En tant que "micro", MicroPython ne prend en charge qu'un sous-ensemble minimal de fonctionnalités d'introspection et de réflexion - MicroPython optimise la gestion des variables locales et n'enregistre ni ne fournit d'informations sur leur introspection.

Les différences d'implémentation : - Le support Unicode est en cours - Les flux d'entrées / sorties mis en mémoire tampon ne sont pas pris en charge. - Le mot clé async def est implémenté avec les simplifications suivantes : il n'y a pas de type d'objet "coroutine" distinct, async def définit simplement une fonction génératrice sans qu'il soit nécessaire que "yield" ou "yield from" apparaisse dans le corps de la fonction.

La PyBoard 1

MicroPython s'exécute nativement sur la PyBoard. Il est possible d'accéder à travers lui à un certain nombre de périphériques disponibles sur la carte (UART, I2C, SPI, ADC and DAC, ...).

La carte est constituée principalement de : microcontrôleur STM32F405RG (168 MHz Cortex M4 CPU, 1024Ko flash ROM et 192Ko RAM) - connecteur Micro USB pour l'alimentation et la communication avec l'ordinateur - slot Micro S - accéléromètre 3 axes (MMA7660) - 24 GPIO sur les côtés, plus 5 GPIO autres sur la ligne du bas - 3x ADC 12-bit - 2x DAC 12-bit - 4 LEDs (rouge, verte, jaune et bleue) - 1 bouton reset et 1 bouton utilisateur - régulateur de tension 3.3V, jusqu'a 250mA - bootloader DFU en ROM pour la mise à jour.



1 *PyBoard*

Comment utiliser la PyBoard

Branchement et mise sous tension

Le plus simple pour commencer est de brancher la PyBoard directement sur un ordinateur avec un câble micro USB. Une fois connectée, la LED verte doit clignoter rapidement.

Il est aussi possible de l'alimenter depuis une source externe en connectant les broches GND et VIN à une source de tension comprise entre 3,6 V et 10 V.

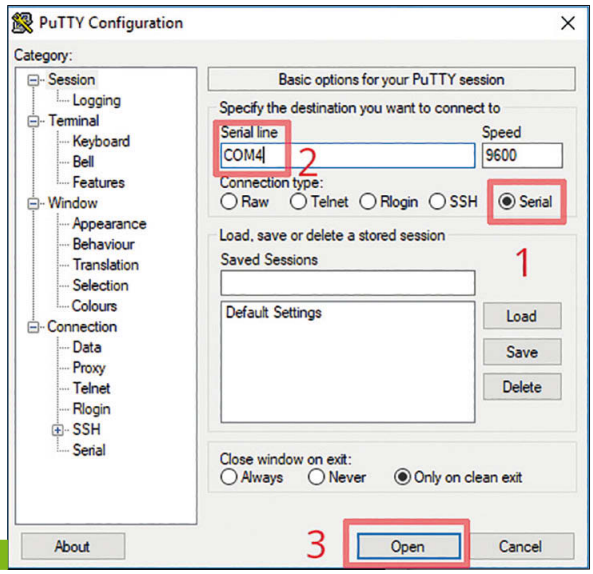
Premier script

Lorsque la PyBoard est connectée, la LED verte clignote le temps du démarrage de la carte (moins d'une seconde). Lorsqu'elle s'éteint, cela indique que la carte est prête.

Un nouveau périphérique de stockage USB a dû être détecté par l'ordinateur. Sur celui-ci, on peut trouver 4 fichiers :

- boot.py : ce script est exécuté lors du démarrage de la PyBoard. Il permet de configurer certaines options de la carte ;
- main.py : contient le code principal de l'application. Il est exécuté automatiquement après boot.py ;
- README.txt : un guide de démarrage très succinct ;
- pybcdc.inf : un pilote Windows pour le pour USB de la carte.

Pour écrire notre premier script en Python, il suffit d'éditer le fichier



putty 2

main.py dans un éditeur de texte (Notepad++ sous Windows par exemple : https://notepad-plus-plus.org). Le fichier contient la ligne suivante :

```
# main.py -- put your code here!
```

Pour faire s'allumer une des LED de la carte, par exemple, rajouter deux lignes au fichier pour qu'il contienne :

```
# main.py -- put your code here!
import pyb
pyb.LED(4).on()
```

Pour exécuter ce script, il faut sauvegarder le fichier et éjecter le disque USB de la PyBoard, comme avec une clé USB classique.
Une fois le disque éjecté, il faut appuyer sur le bouton RST (le bouton en dessous du port USB) pour réinitialiser la carte.
La LED verte doit clignoter rapidement pour indiquer l'initialisation de la carte, puis la LED bleue doit rester allumée.
Félicitations ! Votre premier script en MicroPython fonctionne !

Le REPL

Éditer le ficher main.py permet de programmer la carte, mais cela reste assez lourd. Pour faciliter les choses, MycroPython propose un REPL. REPL est un acronyme signifiant "Read Evaluate Print Loop". Il correspond à l'interface interactive de MicroPython. C'est la façon la plus facile de mettre au point des scripts sur la PyBoard. Avec le REPL, il est possible d'entrer directement des commandes Python, et d'avoir le résultat immédiatement. Pour avoir accès au REPL, en fonction de votre système d'exploitation :

Windows

Pour des versions inférieures à 10, il faut installer le pilote présent sur le disque flash de la carte (pybcdc.inf). Avec Windows 10, la carte est détectée automatiquement.
Ensuite il faut utiliser un terminal série, comme par exemple PuTTY. Avec celui-ci, il faut ouvrir le port série COM4, en général, sinon en essayer d'autres (COM1, COM2, ...). 2

macOS

Ouvrir un terminal, et lancer la commande :

```
screen /dev/tty.usbmodem*
```

Pour quitter, utiliser la combinaison de touches CTRL-A puis CTRL-.

Linux

Ouvrir un terminal, et lancer la commande :

```
screen /dev/ttyACM0
```

Il est peut-être nécessaire d'essayer /dev/ttyACM1 ou plus en fonction de votre ordinateur. Il faut aussi pouvoir accéder aux ports série, en règle générale cela se traduit par être membre du groupe 'uucp' ou 'dialout', en fonction de votre distribution.

Utiliser le REPL

Maintenant, nous pouvons essayer d'exécuter des scripts directement sur la PyBoard. Dans le terminal ouvert précédemment, appuyer sur "Entrée". L'invite MicroPython ">>>" doit alors s'afficher.

```
>>> print("hello pyboard!")
hello pyboard!
```

D'autres commandes à essayer, par exemple :

```
>>> pyb.LED(1).on()
>>> pyb.LED(2).on()
>>> 1 + 2
3
>>> 1 / 2
0.5
>>> 20 * 'py'
'pypypypypypypypypypypypypypypypypypypypy'
```

Reset de la carte

Si quelque chose ne va pas, il est possible de réinitialiser la carte de deux façons : en utilisant CTRL-D pour faire un "soft reset", le terminal doit alors afficher :

```
>>>
PYB: sync filesystems
PYB: soft reboot
Micro Python v1.0 on 2014-05-03; PYBv1.0 with STM32F405RG
Type "help()" for more information.
>>>
```

Si cela ne fonctionne pas, il faut appuyer sur le bouton RST de la carte. Cela va déconnecter la session, et il faudra relancer la commande qui a permis d'accéder au terminal.

Interfaçage de la carte avec le monde extérieur 3

La carte est divisée en deux "skin" quasiment symétriques le X et le Y. Un certain nombre de skins sont disponibles sur la boutique officielle : https://store.micropython.org/category/skins
Les GPIOs sont nommées Xn et Yn en fonction du "skin" auxquelles elles appartiennent. Ce schéma contient l'essentiel des fonctions de

la carte, et permet de s'y retrouver notamment dans l'association entre les GPIOs et les timers associés pour la fonction PWM.

Programmation

Délais et gestion du temps

```
import time

time.sleep(1)        # "dormir" 1 seconde
time.sleep_ms(500)   # "dormir" 500 millisecondes
time.sleep_us(10)    # "dormir" 10 microsecondes
start = time.ticks_ms() # récupérer la valeur du compteur de millisecondes
delta = time.ticks_diff(time.ticks_ms(), start) # calculer une différence de temps
```

LEDs

```
from pyb import LED

led = LED(1) # 1=rouge, 2=verte, 3=jaune, 4=bleue
led.toggle()
led.on()
led.off()

# Les LEDs 3 et 4 peuvent varier d'intensité
LED(4).intensity()   # récupérer l'intensité
LED(4).intensity(128) # assigner l'intensité
```

Bouton interne

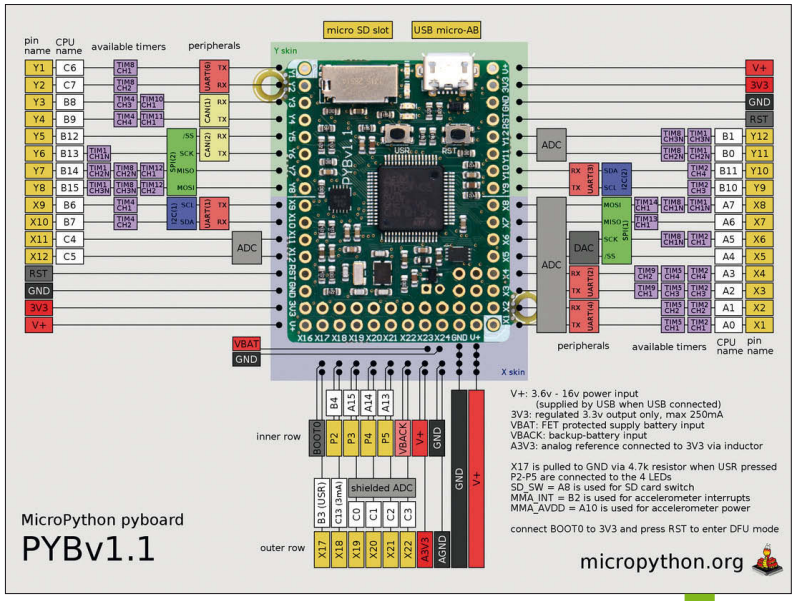
```
from pyb import Switch

sw = Switch()
sw.value() # True ou False
sw.callback(lambda: pyb.LED(1).toggle())
```

GPIO

```
from pyb import Pin

p_out = Pin('X1', Pin.OUT_PP)
p_out.high()
p_out.low()

p_in = Pin('X2', Pin.IN, Pin.PULL_UP)
p_in.value() # 0 ou 1
```

Servo moteur

```
from pyb import Servo

s1 = Servo(1) # servo à la position 1 (X1, VIN, GND)
s1.angle(45) # servo à 45 degrés
s1.angle(-60, 1500) # se déplacer de 60 degrés en  1500ms
```



3 *pyboard*

Interruptions

```
from pyb import Pin, ExtInt

callback = lambda e: print("intr")
ext = ExtInt(Pin('Y1'), ExtInt.IRQ_RISING, Pin.PULL_NONE, callback)
```

Timers

```
from pyb import Timer

tim = Timer(1, freq=1000)
tim.counter() # récupérer la valeur du compteur
tim.freq(0.5) # 0.5 Hz
tim.callback(lambda t: pyb.LED(1).toggle())
```

RTC (real time clock)

```
from pyb import RTC

rtc = RTC()
rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # assigner la date et l'heure
rtc.datetime() # récupérer la date et l'heure
```

PWM (pulse width modulation)

```
from pyb import Pin, Timer

p = Pin('X1') # X1 = TIM2, CH1
tim = Timer(2, freq=1000)
ch = tim.channel(1, Timer.PWM, pin=p)
ch.pulse_width_percent(50)
```

ADC (analog to digital conversion)

```
from pyb import Pin, ADC

adc = ADC(Pin('X19'))
adc.read() # 0-4095
```

DAC (digital to analog conversion)

```
from pyb import Pin, DAC

dac = DAC(Pin('X5'))
dac.write(120)
```

UART (serial bus)

```
from pyb import UART

uart = UART(1, 9600)
uart.write('hello')
uart.read(5) # lire 5 octets
```

SPI bus

```
from pyb import SPI

spi = SPI(1, SPI.MASTER, baudrate=200000, polarity=1, phase=0)
spi.send('hello')
spi.recv(5) # réception de 5 octets du bus
spi.send_recv('hello') # émission et réception de 5 octets
```

I2C bus

```
from pyb import I2C

i2c = I2C(1, I2C.MASTER, baudrate=100000)
i2c.scan() # liste des esclaves
i2c.send('hello', 0x42) # envoie  octets à l'esclave
i2c.recv(5, 0x42) # reçoit 5 octets de l'esclave
i2c.mem_read(2, 0x42, 0x10) # lecture de  2 octets depuis l'esclave 0x42, adresse mémoire 0x10
i2c.mem_write('xy', 0x42, 0x10) # écrire deux bytes dans la mémoire de l'esclave
```

CAN bus (controller area network)

```
from pyb import CAN

can = CAN(1, CAN.LOOPBACK)
can.setfilter(0, CAN.LIST16, 0, (123, 124, 125, 126))
can.send('message!', 123)
can.recv(0)
```

Internal accelerometer

```
from pyb import Accel

accel = Accel()
print(accel.x(), accel.y(), accel.z(), accel.tilt())
```

Safe mode & reset usine

Si jamais quelque chose ne va pas dans le code, il est toujours possible de reprendre la main sur la PyBoard.

La première chose à essayer est de démarrer la carte en "safe mode" : cela inhibe temporairement l'exécution des fichiers boot.py et main.py, et restaure le fonctionnement USB par défaut.

Si le système de fichiers du disque USB est corrompu, il est possible de faire un reset d'usine. Cela restaurera l'état initial de la carte.

Safe mode

Pour entrer en safe mode : 1. Connecter la PyBoard au port USB pour l'alimenter. 2. Appuyer sur le bouton USR. 3. Tout en conservant le bouton USR appuyé, appuyer et relâcher le bouton RST. 4. Les LED vont cycler sur vert, puis orange puis vert et orange. 5. Garder le bouton USR appuyé jusqu'à ce que *seulement la LED orange soit allumée*, et relâcher le bouton USR à ce moment. 6. La LED orange doit clignoter 4 fois, puis s'éteindre. 7. La carte est en "safe mode"

Dans ce mode, boot.py et main.py ne sont pas exécutés, et la carte démarre donc avec les réglages par défaut. Le disque USB doit apparaître, et il est donc alors possible de corriger les erreurs dans les fichiers boot.py et main.py.

Le "safe mode" est temporaire, et ne modifie rien sur la PyBoard. Pour redémarrer normalement la carte, il suffit d'appuyer sur le bouton RST.

Réinitialisation usine

Si le système de fichiers est corrompu (à cause d'un problème d'éjection du disque USB par exemple), il est possible de réinitialiser le système de fichiers.

La réinitialisation du système de fichiers supprime tous les fichiers sur le stockage interne (pas sur la carte SD), et restaure les fichiers boot.py, main.py, README.txt et pybcdc.inf dans leurs états d'origine.

Pour faire la réinitialisation d'usine : 1. Connecter la PyBoard au port USB pour l'alimenter. 2. Appuyer sur le bouton USR. 3. Tout en conservant le bouton USR appuyé, appuyer et relâcher le bouton RST. 4. Les LED vont cycler sur vert, puis orange puis vert et orange. 5. Garder le bouton USR appuyé jusqu'à ce que *les LEDs verte et orange soient allumées*, et relâcher le bouton USR à ce moment. 6. Les LEDs verte et orange doivent clignoter 4 fois, puis s'éteindre. 7. La LED rouge va s'allumer. 8. La PyBoard est en train de réinitialiser le système de fichiers. 9. Toute les LEDs s'éteignent. 10. Le système de fichiers est réinitialisé. 11. Appuyer sur RST pour redémarrer normalement la carte.

Pour aller plus loin

Un certain nombre de modules supplémentaires : https://github.com/micropython/micropython/tree/master/drivers

Autres plateformes disponibles : ESP8266, WiPy (ESP32)...

Intégré à d'autres types de matériels : calculatrices NumWorks et Casio.

CONCLUSION

Pour : - Très facile à mettre en œuvre, la mise au point via le REPL permet un retour immédiat, bien plus rapide que le cycle modification, compilation, upload d'Arduino par exemple.

Contre : - MicroPython est moins populaire, donc moins de librairies disponibles. - Efforts un peu dispersés par le fork d'Adafruit, CircuitPython. •

Philippe BOULANGER
Manager des expertises C/C++ et Python
www.invivoo.com

La migration de Python 2.X à Python 3.X

Depuis 2008, deux versions de Python coexistaient avec, pour chacune d'entre elles, son lot de défenseurs… Guido Van Rossum avait souhaité, avec la version 3.X, corriger certaines syntaxes qui limitaient l'évolution du langage. Python 3 a été conçu comme un vrai langage fonctionnel. Malheureusement, pendant un temps, la version Python 2 étant plus performante, cela a ralenti l'adhésion de la communauté à cette nouvelle mouture. Aujourd'hui, si on regarde les différents benchmarks, la version 3.X est globalement plus performante que la version 2.X. et la communauté a basculé petit à petit mais point assez vite. Aussi Guido a sifflé la fin de la récréation ! Le support de Python 2 prendra fin le 1er janvier 2020… Deux choix s'offrent aux entreprises et particuliers qui sont encore sous Python 2.X : une migration forcée ou rester avec une version obsolète qui n'aura plus de mises à jour, plus aucune nouvelle fonctionnalité et des coûts d'exploitation qui augmenteront au fil des années.

niveau 200

En février 1991, la première version publique de Python (0.9) est mise à disposition de la communauté. En 2000, la première version Python 2.0 est disponible.
Puis en 2008, Python 3.0 et sa rupture de syntaxe est mise à la disposition du public en même temps qu'une nouvelle mise à jour de la branche 2.X (Python 2.6.1). Et depuis, les deux versions continuent de coexister.
Le changement de version majeure n'est pas très fréquent si on le compare à d'autres logiciels ou langages. La première raison est le manque de développeurs comme l'a si bien énoncé M. Victor Stinner lors de la dernière PyConFR qui s'est déroulée à Lille du 4 au 7 octobre : leur force de développement consiste en deux développeurs à temps plein pour valider et administrer les corrections apportées par la communauté… C'est bien peu au regard d'autres suites logicielles. Après dix années de coexistence, Guido Van Rossum a décidé d'arrêter le support de la branche 2. Mais des fournisseurs de librairies comme Django (l'un des plus populaires parmi les frameworks web) avaient déjà annoncé ne plus être compatible avec les versions 2.X à partir de Django 2.0 (avril 2017) suivi par d'autres logiciels très prisés par la communauté (comme Numpy).
Ce désengagement a incité de nombreuses sociétés à migrer leur code sans tarder. Par exemple, Instagram a annoncé en 2017 avoir migré la majorité de son code et Dropbox a annoncé en septembre 2018 avoir achevé sa migration commencée en 2015.
En octobre 2017, il restait de l'ordre de 25 % des codes écrits en Python 2 si on en croit les différentes statistiques. Il reste donc encore beaucoup de travail de migration.

> print

On a parlé de l'évolution de la situation entre Python 2 et Python 3 mais nous n'avons pas abordé quelles étaient les différences entre les deux versions.
La première de ces différences est liée à « print » :

- dans Python 2 c'est une commande

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 5
>>> print "value=", x
value= 5
>>>
```

- dans Python 3 c'est une fonction.

```
$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 5
>>> print( "value=", x )
value= 5
>>>
```

Les parenthèses deviennent obligatoires ce qui, pour certains, est une contrainte. Mais étant une fonction, on peut désormais l'utiliser dans les fonctions en le passant en paramètre. De plus, sa syntaxe étendue permet facilement de rediriger le flux vers un fichier plutôt que vers la console.

> La division sur les entiers

Ce deuxième changement est de ceux qui pourraient nous créer le plus de problèmes. Plutôt que de faire un long discours, un petit exemple va vous montrer le risque :

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print( 2/5 )
0
>>>

$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
```

```
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print( 2/5 )
0.4
>>>
```

En Python 2, la division de deux entiers nous retourne la division euclidienne de ces deux entiers. En Python 3, nous obtenons le résultat de la division flottante.

> Les chaînes de caractères

En Python 2, toutes les chaînes sont par défaut en ASCII. Pour avoir des chaînes unicodes, il fallait faire un appel explicite : u"ceci est une chaine unicode". En Python 3, toutes les chaînes sont unicodes.

> L'appel à raise

La syntaxe a un peu évolué :

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> raise SyntaxError, "une erreur"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: une erreur
>>>

$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> raise SyntaxError( "une erreur" )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
SyntaxError: une erreur
>>>
```

On se rapproche ainsi des syntaxes présentes dans des langages tels que C++ ou C#.

> Les générateurs

Dans un souci d'améliorer les performances ainsi que la consommation mémoire, une des grandes modifications apportées est le changement de certains types de retours. En Python 2, beaucoup de fonctions retournent des listes. En Python 3, ces fonctions retournent des itérateurs.

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print range(5)
[0, 1, 2, 3, 4]
>>> {"a":5, "b":3}.keys()
['a', 'b']
>>>

$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print( range(5) )
range(0, 5)
>>> {"a":5, "b":3}.keys()
dict_keys(['a', 'b'])
>>>
```

> LES IMPORTS

Dans une moindre mesure, des changements sur la façon dont les imports sont effectués ont été apportés en Python 3.

STRATEGIES DE TESTS

On sait lorsque l'on commence la migration. Mais comment sait-on que l'on a fini, qu'il n'y a plus aucun problème caché ? Pour répondre à cette question je vais laisser la parole à mon collègue **Christophe Godard** (Manager de l'expertise « Méthodologies & Pratiques Agiles » au sein d'Invivoo) :

La réussite d'un projet de migration technique (montée de version du langage utilisé comme décrit dans l'article ci-contre, changement de version d'un framework …) passe nécessairement par la mise en place de tests. Pour des raisons de coût et de délai pour les exécuter, il est évident qu'une majeure partie de ces tests doivent être automatisés, d'autant plus qu'ils devront certainement être joués à plusieurs reprises durant la vie du projet. Toutefois, ***quels tests faut-il automatiser et quels tests n'est-il pas préconisé d'exécuter de manière automatique ?*** *D'autre part, il existe une multitude de familles de tests :*

- *les tests unitaires qui ont pour portée une fonction ou un composant ;*
- *les tests d'intégration qui ont pour portée les interactions entre plusieurs composants ;*
- *les tests systèmes ;*
- *les tests fonctionnels ;*
- *les tests de performance ;*
- *les tests de sécurité ;*

Et n'oublions pas le taux de couverture du code. Ces questions sont adressées par un seul artefact : ***la stratégie de test*** *(un document d'assez haut niveau qui définit le cadre d'écriture et d'exécution des tests sur une ou plusieurs applications afin d'en assurer la qualité. Elle doit également définir le suivi et le pilotage des tests : reporting attendu, KPI…). Concrètement, une stratégie de test peut être un simple visuel montrant les activités de test sur une fonctionnalité tout au long de son cycle de réalisation. Elle peut aussi être un document plus complexe en fonction du contexte projet.*

Pour revenir à la question du choix des tests dans une stratégie de test, il y a un pattern à respecter : la pyramide des tests de Mike Cohn. Cette dernière consiste à préconiser d'avoir un grand nombre de tests unitaires (la base de la pyramide) car peu coûteux à développer et maintenir si l'application évolue, un peu moins de tests d'intégration et nettement moins de tests de bout en bout (le sommet de la pyramide) qui sont, eux, plus coûteux à développer et maintenir.

> METTRE EN PLACE LES TESTS

Il faut commencer par mettre en place des tests unitaires afin d'avoir des KPI fiables sur les briques élémentaires du code (fonctions, classes ou petits modules). Pensez à bien couvrir les fonctions utilisant des calculs sur les entiers (notamment la division).

Ensuite il faut mettre en place les tests fonctionnels en commençant par les fonctionnalités les plus importantes : celles qui sont les plus utilisées. En effet, il est logique de dépenser son temps sur les fonctionnalités les plus souvent utilisées…

Il est essentiel d'avoir un retour sur les temps d'exécutions afin d'avoir des éléments de comparaison qui vous permettent de détecter d'éventuelles contre-performances. Ceci étant, il faut que vous ayez des tests de charge afin de certifier que face à une augmentation anormale de la volumétrie il n'y ait pas de comportements bloquants ou dégradés. C'est un problème que j'ai croisé :

Après une migration d'un progiciel nous avons déployé celui-ci, en test, chez certains de nos clients. S'en sont ensuivis plusieurs mois de tests. Le logiciel a été déployé chez tous nos clients et notamment chez le client qui gérait la plus grosse volumétrie. Très rapidement ils nous ont remonté des erreurs dans la génération des rapports. Ils en créaient plus de 10 000. Après plusieurs semaines pour comprendre l'environnement de production du client et estimer sa volumétrie, nous avons pu reproduire le problème dans un environnement de développement. Malheureusement, il y avait un bug dans l'API fichier de Python qui ne les fermait pas correctement. Le bug était, certes, corrigé par un patch mais celui-ci n'avait pas été intégré dans notre logiciel.

Les tests de charge sont donc nécessaires. Ne faites pas l'impasse dessus pour des raisons de budget ou de manque de temps car perdre la confiance de vos clients vous coûtera bien plus cher !

> OUTILS

De nombreux outils vous seront nécessaires pour :

- gérer le workflow, exécuter les tâches comme Buildbot, Jenkins, etc. ;
- créer et exécuter les tests unitaires tels que Unittest ou Pytest ;
- créer et mettre en œuvre les tests fonctionnels ;
- automatiser les tests GUI (comme UFT, anciennement QTP) ;
- mesurer le taux de couverture des tests (le nombre de lignes de code exécutées par les tests).

Le choix des outils dépend avant tout de vos contextes applicatifs et/ou humains. Si votre projet n'est pas outillé ou mal outillé, faites vous aider par un spécialiste de l'intégration continue. Vous perdrez un peu de temps au début mais vous en gagnerez beaucoup lorsque vous passerez en vitesse de croisière.

PERIMETRE

Le périmètre est une notion importante. Il décrit l'ensemble des codes, des modules internes, des librairies externes et des outils que vous utiliser lorsque vous travailler avec votre logiciel. Mais celui-ci inclut aussi le hardware des ordinateurs, les systèmes d'exploitation que vous supportez, la version de la base de données que vous utilisez, etc. Pour reprendre un exemple vécu :

Lors de la migration du progiciel sur lequel je travaillais, nous avions pour cible Windows XP, Windows 7 ainsi que plusieurs versions de Windows Server (de 2002 à 2008 avec différents niveaux de service pack) et pour la base de données nous avions prévu Oracle 9 et Oracle 10. Nous en étions à la phase de tests chez nos clients lorsque nous avons découvert qu'ils avaient migrés. En effet, certains étaient passés à Windows Server 2012 et à Oracle 11.2G. Ceci entrainant des mises à jour de notre côté.

Malheureusement un changement en entraîne souvent un autre ! Et faire une mise à jour majeure d'un logiciel peut entraîner, chez un client, l'envie d'effectuer un changement d'architecture hardware pour décommissionner des serveurs en fin de vie ou des bases de données qui n'ont plus de support : une opportunité en déclenchant une autre.

> LE PERIMETRE « EXTERNE »

Par périmètre externe, je souhaite parler des éléments externes à votre projet sur lequel vous n'avez pas forcément le pouvoir de décision : le hardware, l'OS, la base de données ou certains outils externes (comme CrystalReport) qu'un client vous imposerait. Ne perdez pas de vue que le choix d'un environnement de développement fait aussi partie du périmètre externe. Du fait de sa nature, il ajoute des librairies à votre OS. Utiliser Visual Studio 2017 nécessitera de déployer chez le client un Runtime spécifique (Etes-vous sûr que celui-ci voudra déployer ce runtime ? Certains de vos choix techniques pourraient être invalidés par certaines politiques de sécurité ou d'harmonisation des technologies utilisées). Et par environnement de développement, il faut aussi inclure les outils d'intégration continue.

On a la fausse impression que lorsque l'on change de version, on reste compatible pourtant il y a un tas de petites modifications dont nous n'avons pas conscience qui auront un impact sur notre projet en bien ou en mal. Réussir une migration consiste à prévoir les risques et à les gérer.

Avant de démarrer la migration, il est bon de construire une carte de ce périmètre externe avec les versions supportées aujourd'hui et celles que l'on souhaite supporter après la migration. Faire cette carte peut nécessiter de discuter avec les architectes des clients afin de savoir quels choix ceux-ci ont faits et lesquels d'entre eux auront un impact sur votre migration.

> PERIMETRE « INTERNE »

Pour déterminer le périmètre interne il est intéressant de partir du code et des tests. Rechercher les 'imports' dans le code Python est un bon point de départ mais ce n'est pas forcément suffisant. Python étant un langage dynamique, on peut récupérer un script en base de données puis l'exécuter et ce script aura des dépendances vers d'autres modules.

En exécutant les tests, on va pouvoir déterminer nos dépendances grâce à une fonctionnalité ajoutée à Python 2.3 : modulefinder.ModuleFinder. Nous allons reprendre l'exemple utilisé dans la documentation officielle de Python. 1

La seule limite de cette stratégie sera votre taux de couverture de code. A partir de là vous allez pouvoir créer une carte de vos dépendances en termes de modules Python. Et si vous avez une bonne granularité de vos tests vous pourrez mettre en évidence les dépendances entre les modules. On peut aussi en déduire quels sont les modules utilisés par une fonctionnalité.

```
from modulefinder import ModuleFinder

finder = ModuleFinder()
finder.run_script('bacon.py')

print 'Loaded modules:'
for name, mod in finder.modules.iteritems():
    print '%s: ' % name,
    print ','.join(mod.globalnames.keys()[:3])

print '-'*50
print 'Modules not imported:'
print '\n'.join(finder.badmodules.iterkeys())
```

1

> COMPRENDRE ET ENRICHIR SA CARTE

La première étape consiste à analyser la carte des dépendances. Pour chaque module, il va falloir savoir si :

- il a été codé en interne ou non ?
  - en Python ?
  - dans un autre langage ?
- c'est un module standard ?
  - il a été déclaré obsolète et remplacé par une nouvelle API ?
  - il a des nouvelles fonctionnalités qui pourraient invalider une partie de notre code ?
  - il a des contraintes de compatibilité (OS, par exemple) ?
- c'est une librairie supportée en Python 3 ?
  - elle a changé de nom ?
  - ses API ont évolué invalidant de facto une partie de votre code ?
  - il a des contraintes liées à d'autres modules en terme de compatibilité ou à des éléments tels que l'OS?
  - il a un changement de licences qui nécessiterait de racheter des licences ou passer de nouveaux accords ?
- La librairie n'est pas portée (pendant longtemps cela a été le cas de wxPython par exemple) :
  - Avez-vous le code source ? Etes-vous prêt à maintenir et faire évoluer ce code ?
  - Avez-vous la possibilité de passer à une autre bibliothèque qui fournirait les mêmes services ?

Répondre à toutes ces questions vous permettra de dessiner le contour de votre cible technique et vous obligera à effectuer un travail de recherche pour trouver des équivalents.

> DECOUPER

La clé du succès : ***diviser pour régner*** !!! Vouloir adresser tous les problèmes en même temps en migrant tout en une passe est illusoire et risqué. En effet, il est préférable de se donner des jalons en commençant par migrer des briques indépendantes puis de migrer de nouvelles parties dépendantes des 1ères briques et ainsi de suite.

Il faudra donc : découper ces blocs que vous aller migrer, définir les dépendances entre les blocs qui vous permettraient, si vous avez les ressources disponibles, de les migrer séparément mais en même temps… La planification de votre migration (diagramme de GANTT) commence à prendre forme. Certains risques sont déterminés et des actions peuvent être entreprises pour les réduire.

METHODOLOGIE

Une fois les blocs choisis, il va falloir commencer à les migrer et, là aussi, il est nécessaire de découper la tâche en plusieurs étapes. Et, dans leur grande sagesse, les développeurs de Python ont mis en place des outils qui vont faciliter le travail !

> __future__ ?

Nous allons commencer par intégrer, dans le code Python 2 certaines des nouvelles fonctionnalités de Python 3 grâce aux packages __future__. Ces ajouts seront faits l'un après l'autre, et à chaque fois lancez tous les tests relatifs à la brique logicielle.

- from __future__ import division
  Vous importerez ainsi la nouvelle version de la division entière. Et ainsi vous pourrez corriger le code
- from __future__ import print_function
  'print' devient une fonction, rajouter les parenthèses
- from __future__ import absolute_import
  utilisation des mêmes règles de gestion des imports qu'en Python 3
- from __future__ import unicode_literals
  ajout des modifications liées à la gestion de l'unicode

> LE GRAND SAUT

Une fois que les modifications ont été apportées, on est prêt à faire le grand saut : grâce à 2to3.py nous allons pouvoir convertir nos fichiers sources pour être compatibles avec la syntaxe de Python 3. Mais tout d'abord, n'oubliez pas de migrer les fichiers de tests. Car si vous n'avez pas de tests unitaires qui tournent en Python 3 comment pourriez-vous valider le travail accompli ?

Les premiers tests sont susceptibles de remonter des erreurs de compilation :

- des modules non-trouvés : les fameux modules qui ont changé de noms ou qui ne sont pas portés ;
- des méthodes ou des objets inconnus : les fameux changements d'API ;
- des options invalides dans des appels de fonctions : des changements d'API.

Si vous avez correctement fait votre travail au cours de l'analyse du périmètre, vous connaissez déjà la liste des actions pour passer ce palier. Pour les modules non-portés, il y a un outil qui peut vous débloquer : 2to6. Il vous permettra d'importer en Python 3 des modules écrits pour Python 2. Mais cette solution doit rester temporaire pour vous permettre d'aller plus loin dans la migration.

Puis arrivent les premières exécutions avec leurs lots de problèmes. L'un des premiers problèmes que vous rencontrerez est lié au fait que les fonctions qui retournaient des listes retournent désormais des itérateurs : convertir l'itérateur en liste via la fonction '*list(it)*' est certes rapide mais pas forcément efficace dans votre contexte.

Et à ce niveau il n'y a pas de solution miracle : cela dépend uniquement de votre code et de ses dépendances. Il faut prendre les problèmes les uns après les autres, les analyser et leur trouver des réponses.

REFACTORING

Vous venez de finir la migration, votre code tourne sous Python 3. Cependant, pour parachever votre migration il va falloir remplacer certaines façons de faire (liées à Python 2) par celles qui sont

conseillées en Python 3. Vous tirerez ainsi parti des nouvelles fonctionnalités, des meilleures performances ainsi que d'un code plus compact. Vous trouverez ci-dessous une liste non-exhaustive du refactoring qui vous permettra de rendre votre code plus lisible ou plus performant. Là aussi, ne faites pas toutes les modifications en une fois. Faites-les, une par une et brique par brique (les mêmes que celles que vous aviez isolées grâce à votre carte de dépendances).

> listdir vs scandir

En Python 2, vous aviez deux manières de parcourir une liste de fichiers/répertoires via le module 'os' : os.walk et os.listdir.

```
from os     import listdir
from os.path import splitext, isdir
PATH = "C:\\Tools\\Anaconda3"

def nb_file_listdir( path, ext ):
  nb = 0
  for name in listdir( path ):
    fname = F"{path}\\{name}"
    if isdir( fname ):
      nb += nb_file_listdir( fname, ext )
    else:
      r, e = splitext( name )
      if e.lower() == ext:
        nb += 1
  return nb
print( "nb files=", nb_file_listdir( PATH, ".py" ) )
```

En Python 3, une nouvelle API est apparue : os.scandir. Celle-ci est bien plus performante que os.listdir.

```
from os     import scandir
from os.path import splitext

PATH = "C:\\Tools\\Anaconda3"

def nb_file_listdir( path, ext ):
  nb = 0
  for entry in scandir( path ):
    if entry.is_dir():
      nb += nb_file_listdir( entry.path, ext )
    else:
      r, e = splitext( entry.name )
      if e.lower() == ext:
        nb += 1
  return nb

print( "nb files=", nb_file_listdir( PATH, ".py" ) )
```

Cette nouvelle version est cinq fois plus rapide sur l'exemple choisi.

> Fstring vs string.format

En Python 2.X, pour formater une chaîne de caractères nous pouvons utiliser la fonction suivante :

```
nom = "Philippe"
age  = 46
text = "{nom} a {age} ans".format( nom=nom, age=age )
print( text )
```

L'introduction des Fstring en Python 3.6 permet de simplifier le code (de le rendre plus lisible) tout en étant plus efficace :

```
nom = "Philippe"
age  = 46
text = F"{nom} a {age} ans"
print( text )
```

> générateurs et itérateurs

Les API qui retournaient des listes en Python 2 retournent majoritairement des itérateurs en Python 3. Cela amènera probablement à repenser certains algorithmes.
Mais il est également possible de remplacer du code plus conséquent. Par exemple :

```
def frange( a, b, n ):
  h = ( b - a ) / n
  for i in range( n+1 ):
    yield a + i * h

for x in frange( 0., 1., 10 ):
  print( x )
```

que l'on peut remplacer par :

```
a = 0.
b = 1.
n = 10
h = ( b - a ) / n
for x in ( a + i * h for i in range( n+1 ) ):
  print( x )
```

> with

Une exécution contextuelle dans un bloc 'with' permet de simplifier le code tout en évitant les erreurs liées à un oubli de libération de ressources (notamment avec les verrous dans la programmation multi-threadée).

CONCLUSION

Les migrations sont comme des enfants, il n'y en a pas 2 identiques : chacune aura ses spécificités ou ses difficultés. Mais ne baissez pas les bras devant la difficulté, allez-y calmement, méthodiquement par étapes. Et à chaque étape : TESTEZ !
N'hésitez pas à demander conseil à des personnes expérimentées ou qui ont du recul par rapport à votre tâche afin de vous donner un regard neuf, des idées différentes de celles que vous avez déjà eues ou des recettes éprouvées. •

Claire Savinas
Professeur en lycée et formatrice en langage Python. Passionnée d'informatique Ingénieur en informatique

NumWorks & Python

L'algorithmique et la programmation prennent une place de plus en plus importante dans l'enseignement des mathématiques au lycée. Le programme actuel indique que « le choix du langage se fera parmi les langages interprétés, concis, largement répandus, et pouvant fonctionner dans une diversité d'environnements », ce qui correspondait très fortement au langage Python. Il est maintenant explicitement cité dans le projet de programme de la réforme du lycée.

Dans ces conditions, la présence de Python sur les calculatrices destinées au lycée est donc quasiment obligatoire. Il est toujours possible d'utiliser un ordinateur, mais celui-ci ne sera pas autorisé le jour du baccalauréat.

Rappels Python

Python a débuté son histoire comme un projet personnel du développeur néerlandais Guido van Rossum en 1989. Il n'a cessé depuis de croître en popularité et est depuis largement utilisé.
Il dispose d'une très grande variété de modules permettant de faciliter le développement d'applications. On peut par exemple citer : - NumPy pour le calcul scientifique - PyGame pour le développement de jeux - TkInter pour les interfaces graphiques - Matplotlib pour le tracé de graphiques.
La syntaxe de Python diffère des syntaxes descendantes du C par le fait que c'est l'indentation qui détermine les blocs fonctionnels, et il n'y a pas de terminateurs de ligne (le ; dans beaucoup de langages). Le traditionnel "Hello World !" en Python est donc :

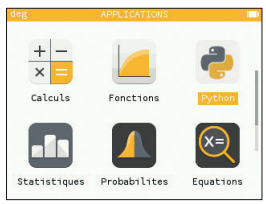
```
def hello():
 print("Hello World !")

hello()
```

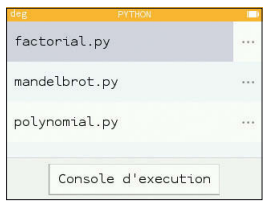
Le code Python doit donc être correctement indenté pour fonctionner, ce qui oblige à maintenir un minimum de lisibilité dans l'écriture du code.
Python dispose d'un interpréteur utilisable en mode interactif, le REPL qui permet de saisir des commandes et d'obtenir un résultat immédiatement. Cela permet de faciliter grandement la mise au point des programmes ci-dessus. Un exemple de session interactive, sur un ordinateur. Le ">>>" est le prompt Python, c'est à cet endroit que l'interpréteur attend les saisies au clavier de l'utilisateur.

```
$ python3
Python 3.6.7 (default, Oct 21 2018, 08:08:16)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>>
```

L'implémentation sur la NumWorks

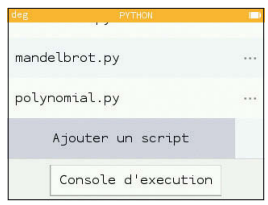
La calculatrice a été proposée très tôt avec un support du langage Python qui n'a cessé d'évoluer depuis. Nous allons donc nous baser sur ce qui est disponible dans le dernier firmware à ce jour (version 1.7.1).
Pour rappel, il est possible (et recommandé) de mettre à jour sur le site officiel : https://workshop.numworks.com/devices.
Le menu Python de la calculatrice nous amène sur cet écran :

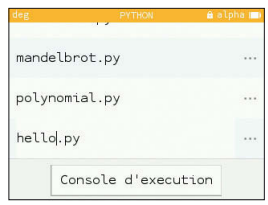


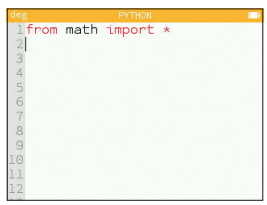


L'éditeur

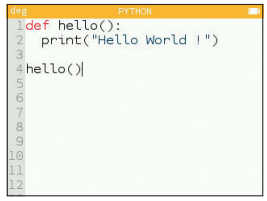
Pour créer un nouveau script, il suffit de descendre un peu dans le menu et de sélectionner "Ajouter un script", entrer son nom et enfin l'ouvrir :



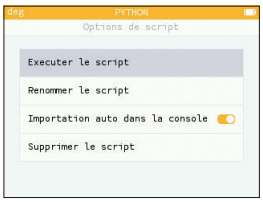




Reprenons le "Hello World" dans l'introduction :

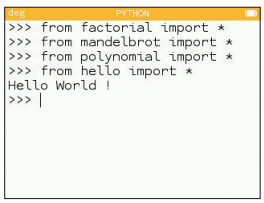


Pour exécuter le script, il faut revenir sur le menu précédent avec  (le script est sauvegardé automatiquement), aller sur le menu "…" et choisir "exécuter le script" :



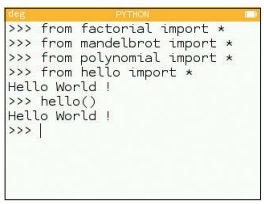
La console d'exécution

L'autre moyen d'appeler un script est de passer par la console d'exécution. Le menu se trouve au dessous de la liste des scripts :



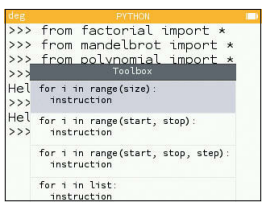
On peut constater que le hello world est exécuté automatiquement à l'ouverture de la console, car le script est importé automatiquement. Il est possible de modifier ce comportement dans le menu "…" à côté du nom du script.

Dans ce mode, on peut entrer du code Python et avoir le résultat directement. On peut appeler par exemple notre hello world :



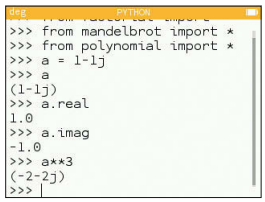
A noter qu'il y a des raccourcis disponibles pour ne pas avoir à tout taper au clavier. La touche  permet d'ouvrir la liste des fonctions connues de l'interpréteur :

 permet d'avoir des raccourcis sur les mots clés du langage et les modules présents dans la calculatrice :



Cela s'avère très pratique pour saisir des scripts directement sur la calculatrice.

Notons que les nombres complexes sont parfaitement gérés :



Les modules et exemples

Un certain nombre de modules sont présents pour étendre la fonctionnalité du langage de base :

math

Il contient les fonctions et constantes mathématiques usuelles : e pi sqrt pow exp expm1 log log2 log10 cosh sinh tanh acosh asinh atanh cos sin tan acos asin atan atan2 ceil copysign fabs floor fmod frexp ldexp modf isfinite isinf isnan trunc radians degrees erf erfc gamma lgamma

cmath

Il contient les fonctions et constantes mathématiques usuelles opérant sur les complexes :e pi phase polar rect exp log sqrt cos sin

random

Il contient les fonctions classiques concernant la génération de valeurs aléatoires : getrandbits seed randrange randint choice random uniform

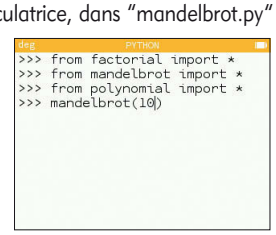
Bien que la calculatrice contienne un générateur de nombres aléatoires en hardware, le module Python utilise un algorithme "classique". La suite des nombres tirés sur deux calculatrices est donc identique. Une solution consiste à utiliser la fonction seed avec une valeur différente pour initialiser l'algorithme.

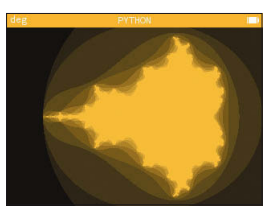
kandinsky

Ce module est spécifique à la calculatrice. Il est dédié à l'affichage en mode graphique sur l'écran de la calculatrice. Il compte les fonctions suivantes :

- color : permet de créer un objet représentant une couleur à partir des composantes rouge, verte et bleue, sous la forme d'entiers entre 0 et 255. Par exemple color(255,128,0) correspond à 
- get_pixel permet de récupérer la couleur affichée par un pixel, par exemple get_pixel(8,9) pour voir le pixel à x=8 et y=9. La surface utilisable représente 320 x 222 pixels, car la barre en haut de l'écran est toujours affichée. - set_pixel permet d'assigner une couleur à un pixel. Par exemple set_pixel(8,9,color(255,128,0)).
- draw_string permet d'afficher une chaîne de caractères. Par exemple draw_string("hello",8,9)

Un exemple complet d'utilisation est présent par défaut dans la calculatrice, dans "mandelbrot.py" :

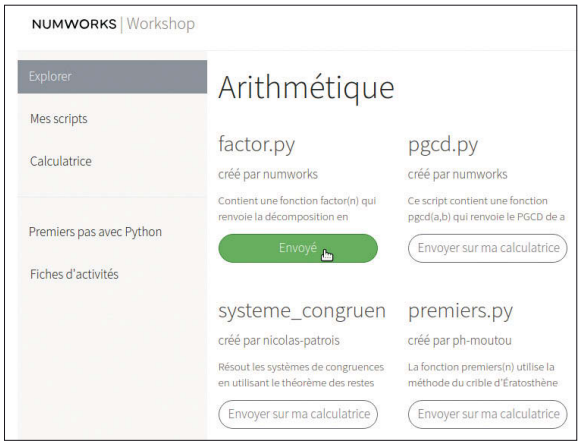




Le Workshop

Saisir des scripts sur la calculatrice n'est pas forcément le plus facile. Pour pallier ce problème, NumWorks a créé le "Workshop", accessible sur son site dédié : https://workshop.numworks.com/python

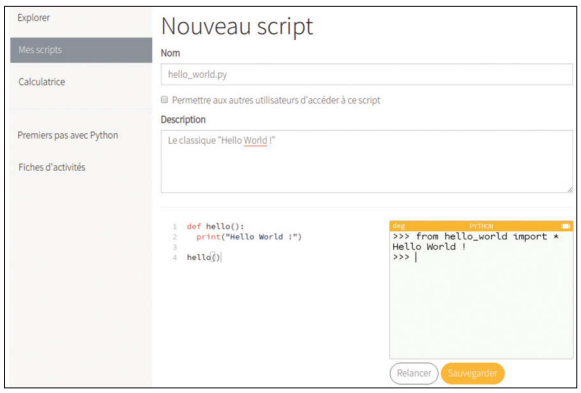
Celui-ci permet d'accéder aux scripts partagés par les autres utilisateurs, et de les envoyer en un clic sur la calculatrice :



Lors de la première utilisation, il faut suivre une procédure très bien décrite sur le site. Ensuite, il n'y a qu'à connecter la calculatrice pour qu'elle soit reconnue.

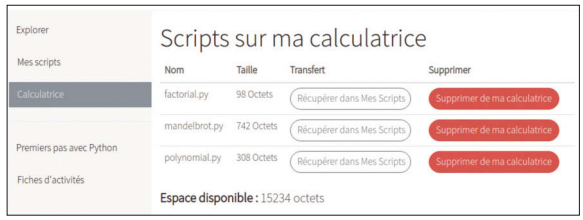
A noter, le Workshop ne fonctionne qu'avec le navigateur Google Chrome pour le moment.

L'autre fonctionnalité très importante du workshop est qu'il permet d'écrire et de tester les scripts directement dans le navigateur ! Pour cela, il suffit de se rendre dans la section "Mes scripts", et de choisir "Nouveau script". On se retrouve alors sur la page suivante, avec par exemple dans la capture, le hello world de l'introduction :



Un fois sauvegardé, on arrive sur la page suivante, qui permet de le tester, et de l'envoyer en un clic sur la calculatrice :

Une dernière fonctionnalité permet de gérer les scripts présents dans la calculatrice, et de les importer dans le Workshop.



Le Workshop permet donc de mettre au point, partager et transférer très facilement des scripts. Il est librement accessible depuis un navigateur web.

A noter, un émulateur "complet" de la calculatrice est aussi disponible sur le site de NumWorks : https://www.numworks.com/fr/simulateur/. Il permet de tester la calculatrice, faire des copies d'écran, et dispose aussi d'un mode "plein écran" pour faciliter la projection (en salle de cours par exemple).

Conclusion

L'interpréteur utilisé dans la NumWorks est basé sur MicroPython, qui est aussi utilisé dans le modèle Casio Graph 90+E et propose une très bonne compatibilité avec l'implémentation officielle du langage Python. Les modules sont adaptés pour l'usage dans la calculatrice, et le cœur du langage est suffisamment complet pour l'usage qui est visé. Des améliorations sont très certainement à prévoir dans le futur.

Il manque peut-être quelques fonctionnalités d'entrées/sorties : tracé de lignes ou de formes géométriques simples (rectangle, cercle, ...). Elles peuvent toujours être réalisées en Python et partagées sur le Workshop.

La calculatrice étant "open source", il est toujours possible de modifier soi-même le logiciel, pour inclure ses propres fonctionnalités, et de les proposer à NumWorks pour une éventuelle intégration : https://github.com/numworks/epsilon.

Yannick Grenzinger

David Panza

Du monolithe aux micro-services

Partie 1

OU CE QUE NOUS AURIONS AIMÉ SAVOIR AVANT DE NOUS LANCER

Dans cet article, l'objectif est de vous faire un retour d'expérience sur notre passage de deux applications "monolithiques" à une architecture micro-services en se concentrant sur ce que nous aurions aimé savoir avant de nous lancer.

Note importante : nos notes sont des articles à rechercher dans votre moteur de recherche favori.

niveau 200

Sortir du monolithe

Avant de démarrer, un peu de contexte. L'histoire est assez classique en fait, **deux applications, démarrées il y a plus de 10 ans** et gérées au sein d'un même département, **ont mal vieilli**. La maintenance évolutive a été externalisée. Les tech leads sur le sujet en France étaient principalement là pour valider la qualité a posteriori des développements. Le problème est alors bien connu : la dette technique s'accumule, ajouter de nouvelles fonctionnalités devient de plus en plus difficile et la frustration se crée aussi bien côté développement que métier. Il a donc fallu reprendre le contrôle et surtout **redonner la capacité de livrer de la valeur facilement et rapidement**.

Sur une des applications, tout a commencé en phagocytant petit à petit l'interface utilisateur vieillissante en superposant une couche technique moderne type SPA[1] (avec AngularJS et en rajoutant une couche REST). En parallèle, de nouveaux domaines fonctionnels sont ajoutés par l'intermédiaire de nouvelles applications (ces fameux micro-services). La transition, toujours en cours, doit être la plus transparente possible pour les utilisateurs. Une stratégie d'étranglement a donc a été mise en oeuvre afin d'externaliser petit à petit les fonctionnalités du monolithe. 1

Sur la deuxième application, la transition mêle stratégie d'étranglement, pour apporter de nouvelles fonctionnalités à l'existant, avec une refonte plus radicale du produit en découpant par cas d'utilisation. Cette refonte pose directement les bases des nouveaux micro-services en utilisant la connaissance de l'existant.

Cette transition de monolithe à des micro-services est riche en apprentissage et c'est cette expérience que nous aimerions vous partager avec **un guide sur les étapes nécessaires et les différents pièges que vous rencontrerez** dans cette aventure.

Nous avons décidé de construire ces étapes en quatre axes de maturité : la mise en pratique, le déploiement continu, la collaboration et la production. Ces quatre axes ont chacun leurs défis mais nous pensons que l'ordre représente une progression dans les difficultés que vous allez sûrement rencontrer.

Et pour commencer…

Les micro-services c'est quoi ?

Ce mot, vous l'avez sûrement entendu des centaines de fois. Si vous ne savez pas de quoi il s'agit, eh bien c'est très simple : **un micro-service est une application à responsabilité unique et totalement autonome**. Si on élargit, créer une plateforme de micro-services revient à créer plusieurs applications, chacune ayant une tâche bien spécifique et pouvant collaborer avec d'autres. Nous entendons par **responsabilité unique** qu'un micro-service **se concentre sur une et une seule chose**, comme gérer le panier d'un client, les commandes ou encore la facturation.

Nous entendons par **autonomie** qu'un micro-service doit être indépendant des autres. Cela implique **un couplage très faible avec d'autres micro-services ou leurs composants**. Il doit avoir sa propre configuration, infrastructure ou encore base de données. En cas de dysfonctionnement, un micro-service devrait avoir une influence minime sur les autres et donc sur le fonctionnement de l'ensemble.

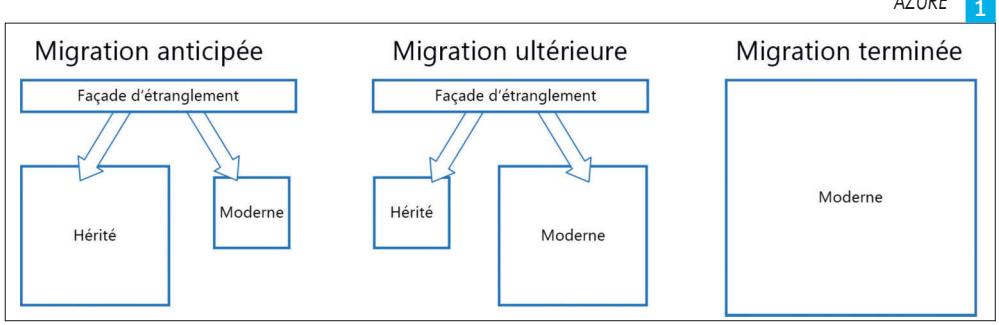
L'idée de ce type d'architecture **est de diviser pour mieux régner**[2], ce qui permet de grandement réduire la complexité d'un système. Et pour cause, il est beaucoup plus facile de raisonner sur de simples et petites problématiques plutôt que d'attaquer un problème dans son entièreté.

Et par opposition aux micro-services se trouve le méchant **monolithe**, un monstre logiciel venu des profondeurs du legacy, qui **centralise toutes les responsabilités et l'intelligence métier**. Une application qui les réunit toutes en somme. Vous en avez sûrement déjà rencontrées durant votre carrière. Lorsqu'un monolithe est mal conçu, il devient plus complexe à faire évoluer, dû au couplage, à la difficulté à tester, à maintenir, à maîtriser ou alors à déployer.

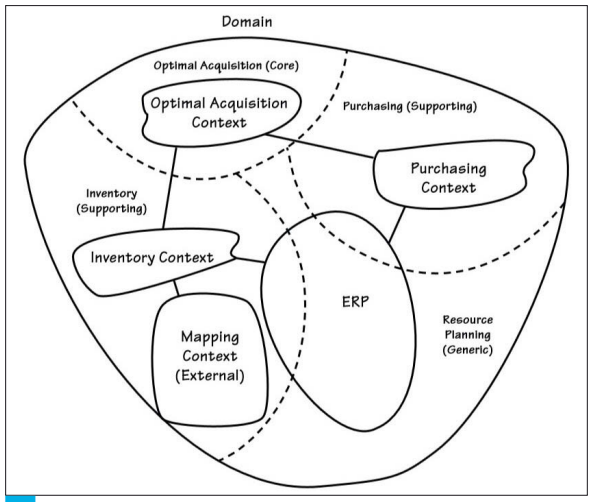
Ce dont il faut être conscient dans le développement logiciel, c'est que rien n'est figé. Les technologies s'adaptent, les styles de conception changent et **les besoins métier évoluent**. Le logiciel que vous êtes en train de concevoir répond sûrement à une problématique dans un contexte et une temporalité bien précis.

(1) Single Page Application

(2) Voir divide-and-conquer, coping with complexity, http://effectivesoftwaredesign.com



Stratégie d'étranglement, AZURE 1



2

Domaine et Bounded Context

Cette problématique évoluera elle aussi inévitablement et **votre logiciel devra**, sous peine de devenir obsolète, **s'adapter aux changements**. Il devient alors indispensable de penser votre code interchangeable et facilement malléable pour qu'il puisse s'adapter avec le temps (via les principes SOLID[3] par exemple).

Pour les plus anciens, vous aurez sûrement remarqué que diviser un système en plusieurs petites parties n'est pas une nouvelle pratique. Une telle architecture existait bien avant les micro-services : **SOA**[4]! Il est donc naturel de penser que les micro-services sont un sous type ou encore un exemple d'implémentation SOA. Eh bien pas tout à fait… Ces deux approches ont effectivement **une forte ressemblance dans leurs intentions de séparer la complexité mais sont profondément différentes dans leurs motivations**.

Les micro-services ont pour volonté de séparer pour mieux s'adapter aux changements. Par s'adapter, cela veut dire faire évoluer du code ou en jeter ! Si le besoin se manifeste, il est tout à fait acceptable de se séparer d'une partie de vos micro-services et ce sans mettre en péril votre système. Le couplage a donc une importance capitale !

A l'inverse, l'approche **SOA vise à séparer pour mieux réutiliser**. Il répond avant tout à une problématique de centralisation et de réduction des coûts. En augmentant la réutilisation d'un service, le couplage en devient que plus fort. Dès lors, il est difficile de faire évoluer votre système ou d'en changer une partie sans avoir un impact global. Une telle architecture répond davantage à une problématique d'urbanisation d'un SI.

Maintenant que vous savez ce qu'est un micro-service, nous allons pouvoir introduire nos quatre axes de maturité.

1ER AXE : Mise en pratique

Séparation des responsabilités

Comme vous le savez, les micro-services visent à séparer les responsabilités d'un système en services à responsabilité unique.

La première difficulté est donc d'**identifier les différentes responsabilités de votre système**. Naturellement en tant que développeur, vous allez tenter une séparation en analysant tout d'abord le code et ses dépendances. Une approche somme toute rapide et pragmatique, mais qui vous donnera un découpage peu viable à long terme, car trop technique.

L'autre approche se concentre davantage sur une analyse métier de votre logiciel. Il va d'abord falloir **connaître vos utilisateurs et leurs intentions**. Et pour ce faire, il existe une démarche qui tend à se populariser : la conception dirigée par le domaine ou encore **Domain Driven Design**[5] (DDD). L'un des premiers ouvrage à y faire référence est "Domain Driven Design, Tackling Complexity in the Heart of Software" d'Eric Evans paru en 2003. 2

Dans l'approche DDD, le but est d'extraire la connaissance métier afin de l'expliciter. Et l'un des points importants est de partager un même niveau de communication par un même langage, avec le moins d'ambiguïté possible (notion d'"Ubiquitous Language"). De multiples ateliers peuvent aider à acquérir ce langage et à extraire la connaissance métier : l'Event Storming[6], l'Exemple Mapping[7] ou encore la pratique BDDs[8]. Une fois extraite, elle fera partie intégrante de votre logiciel, et votre code en sera fortement imprégné, notamment dans le nommage des concepts. De plus, cela vous mènera justement à prendre des décisions stratégiques, avec l'émergence d'agrégat qui deviendront sûrement vos futurs micro-services ! Vous l'aurez compris, concevoir une bonne architecture micro-service ne nécessite pas seulement des compétences techniques, mais requiert une connaissance très fine du métier auquel il répond !

La collaboration entre micro-services

Par définition, **un micro-service ne rend qu'un seul service** et ne possède qu'une seule responsabilité. C'est donc généralement par **un ensemble de services** que l'on **construit une plateforme qui répond aux besoins utilisateur**. Pour ce faire, la collaboration entre eux est indispensable. A ce stade, vous avez deux possibilités :

La communication synchrone et directe : se basant sur le protocole **HTTP** et vous permettant d'appeler d'autres micro-services. Ce type de communication est **la plus naturelle et de loin la plus simple**. De plus, **les données récupérées** auront pour avantage d'être **les plus à jour**. **Néanmoins**, cette possibilité va induire **un couplage direct** ! Car si le service appelé est lent ou ne répond pas, alors l'appelant devra inévitablement le prévoir, via, par exemple, une stratégie de fallback (pattern Circuit Breaker[9]).

A l'inverse, **la communication asynchrone** passera par un protocole de **messages** fournis par des brokers type Kafka ou encore RabbitMQ. Ces messages peuvent être utilisés à **diffuser les données d'un service**, qui sera potentiellement **consommé par d'autres**. Ce type de communication a pour gros avantage le **découplage**. Donc plus de problème lié à l'indisponibilité d'un service ou de lenteur réseau ! Cependant, cette communication est **plus compliquée à mettre en oeuvre** et amène son lot d'inconvénients, comme la fraîcheur des données, la réémission ou la perte de messages.

En plus du moyen technique mis en oeuvre dans la communication entre micro-services, il vous faudra choisir une stratégie de collaboration. Et encore une fois, deux stratégies sont possibles :

L'orchestration, où un seul micro-service

(3) Voir SOLID principles made easy, http://hackernoom.com

(4) Service Oriented Architecture

(5) Voir Domain Driven Design sur https://domainlanguage.com/ddd/

(6) Voir Event Storming par Alberto BRANDOLINI

(7) Example Mapping

(8) BDD ou Behavior Driven Development

(9) Voir Circuit Breaker Pattern, Michael Nygard

coordonne un ensemble pour un cas d'usage bien précis comme un tunnel de commande. Toute la logique métier et de **contrôle** est **centralisée**. Si un problème survient, alors l'orchestrateur saura quoi faire pour remettre le système dans un état cohérent. Cette stratégie met en oeuvre un cerveau central qui, **en cas de problème, peut potentiellement paralyser toute une plateforme** (SPOF[10]).

A l'opposé, **la chorégraphie** s'articule sur un principe d'**autonomie**. Chaque service, par l'intermédiaire d'un évènement survenu dans le système, peut réagir en conséquence. L'idée est donc de mener la collaboration par **l'écoute d'événements**. Chaque partie étant autonome, le dysfonctionnement de l'un ne met pas en péril les autres. Mais avec ce type de stratégie, **il est possible que votre système puisse se retrouver dans un état incohérent**. Par exemple dans le cas d'un tunnel de commande, si le micro-service de livraison a déjà effectué la livraison alors que le service de facturation échoue. Ce concept de transaction déjà complexe à gérer, même dans un monolithe, le devient encore plus dans un système distribué. Il est toutefois possible d'y remédier par le pattern Saga[11].

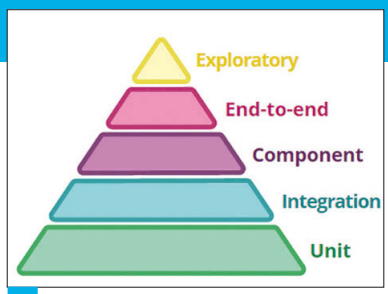
La technique au service d'un besoin

Si un micro-service est autonome et à responsabilité unique, il va de soi que son code, ainsi que son rôle soit unique et donc très spécifique. A cette spécificité, vous allez avoir la possibilité d'**adapter vos outils, langages** et autres. L'impact de ce choix devrait avoir une influence minime sur les autres micro-services, et cela vous donne plus de liberté afin de répondre au besoin métier.

Ainsi, choisir un langage objet ou fonctionnel, fortement typé ou non, utiliser un framework ou librairie plutôt qu'un autre ou encore une base de données orientée document plutôt que relationnelle, doit toujours se faire selon la nature du besoin. Et c'est là l'un des points forts de cette architecture, car au delà de remplir votre CV, **cela vous permettra d'expérimenter rapidement**, sans impact majeur, **afin de capitaliser vos expériences afin de la partager** avec d'autres équipes (CF 3ème axe de maturité - La collaboration).

(10) *Single Point Of Failure*

(11) *Voir Pattern saga sur microservices.io*



3 *Pyramide de tests, Martin Fowler*

Maîtriser la qualité de votre architecture

L'architecture micro-services se base avant tout sur **un système distribué**, et cela **amène son lot de complexité, notamment en termes de qualité**. Dans le cas d'un monolithe, où la base de code reste centralisée, il est plus facile de vérifier le bon fonctionnement de votre application avant de l'envoyer en production.

Un des avantages des micro-services est de développer plus petit et de manière isolée, ce qui raccourcira certainement le temps entre le développement et la mise en production. En théorie, cela vous permettra d'aller plus vite… enfin presque ! Comme tout bon développeur, vous vous assurerez que votre code marche et n'engendre pas de bugs pour vos utilisateurs. Et pour ce faire, une stratégie de test sera appliquée et inclura très certainement des tests automatisés à différents niveaux . Nous allons donc **assurer la qualité de nos développements par les tests** ! 3

La pyramide de tests nous aide à élaborer une stratégie en les répartissant à différents niveaux d'abstraction. Dans le cadre d'un monolithe, cela devient presque trivial. Mais dans le cas d'un système distribué, comment assurer que votre système marche toujours ?

L'une des solutions est d'ajouter des tests de haut niveau tels que **les tests d'interface ou end-to-end**. Ce type de test va couvrir votre système entier mais reste **extrêmement coûteux** à élaborer et surtout à maintenir **car très sensible aux changements**. Une autre solution est d'**ajouter des testeurs** qui vont manuellement vérifier la cohérence de votre application. **Pas très efficace surtout si l'on augmente la cadence de livraison**, il vous faudra plus de testeurs et/ou plus de temps pour tester notamment la régression. Il va donc falloir adapter votre stratégie de test en conséquence. **La complexité d'un système distribué se situe dans la collaboration** entre les micro-services. Les tests de contrat, plus connus sous le nom de **Consumer Driven Contract**[12] (CDC), permettent de tester la collaboration entre deux micro-services (avec des outils tels que PACT[13] ou encore Spring Cloud Contract[14]). 4

Une autre manière d'assurer qu'un système distribué ne comporte pas de bugs est de **simuler le comportement de vrai utilisateur**. Plus complexe à mettre oeuvre, le test de simulation (ou *Simulation Testing*[15]) va vous permettre de **trouver des anomalies** sur de vrais cas d'usage. Un pattern qui se base sur la génération automatique de tests, qui vérifieront la cohérence d'un système **via des invariants** (connu sous le nom de ***Property Based Testing***[16]).

2ÈME AXE : Le déploiement continu

"Quand il y en a un, ça va. C'est quand il y en a beaucoup qu'il y a des problèmes" - Un homme politique français parlant des micro-services.

L'intégration continue

Rapidement, vous allez avoir besoin de livrer vite et rapidement de nombreuses applications. Il est donc nécessaire d'**avoir**

(12) *Voir Consumer-Driven Contracts: A Service Evolution Pattern, Martin Fowler*

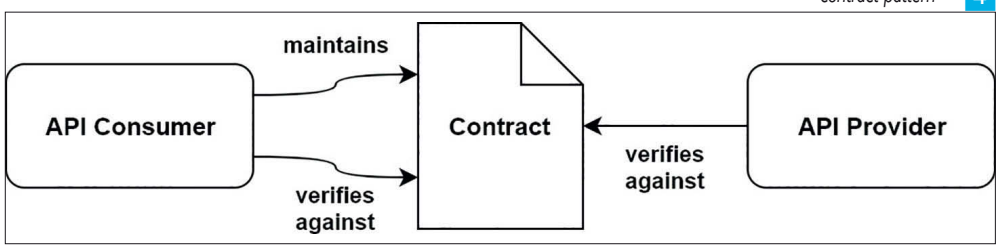
(13) *Voir PACT,* https://docs.pact.io/

(14) *Voir Spring Cloud Contract,* https://cloud.spring.io/spring-cloud-contract/

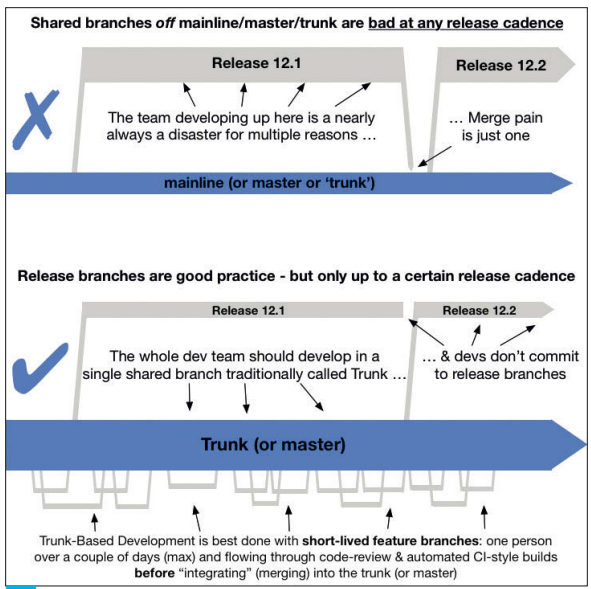
(15) *Voir Simulation Testing, Michael Nygard*

(16) *Voir Domain Invariants & Property-Based Testing for the Masses, Romeu Moura*

Consumer driven contract pattern 4



5 *Pipelines Jenkins*



6 *Trunk based development*

un CI[17] robuste. Bien sûr on imagine que vous avez déjà une instance de votre outil de CI, au hasard Jenkins. Cette unique instance, tout à fait raisonnable dans le cadre de quelques applications, va devenir rapidement un fardeau dans le cadre des micro-services. Tout d'abord la fréquence des builds est largement multipliée par le nombre d'application ce qui peut ralentir votre CI et donc la capacité de livraison des équipes. A cela s'ajoute potentiellement l'utilisation de Pull Request[18] ayant chacun leur branche et donc leur build. La charge mémoire et CPU sur votre CI va augmenter drastiquement. Et si cela ne suffisait pas, des problèmes déjà ennuyeux comme des conflits de ports ouverts lors des tests d'intégration vont devenir encore plus désagréables. Si vous voulez que la file d'attente des builds reste raisonnable et éviter trop de faux positifs, **il va falloir mettre à l'échelle votre CI**. Dans notre cas, nous avons pu tirer profit d'un passage sur Docker Enterprise avec Jenkins lançant chaque build dans un conteneur, le tout avec des pipelines[19] codés en groovy et versionnés sur Git.

Vous pourriez aussi utiliser le tout nouveau Jenkins X permettant de tirer profit de Kubernetes, des concurrents comme Concourse ou GoCD ou encore des solutions de type SaaS comme CloudBees. L'important est de pouvoir mettre rapidement à l'échelle votre CI en fonction de vos besoins. 5

Trunk-Based Development

Souvent pour livrer des fonctionnalités, on a pour habitude de créer des branches, ce qui provoque souvent de douloureux moments lors du merge sur la branche principale. Cette douleur est proportionnelle au nombre de branches et à leur durée de vie. A cela, vous rajoutez le nombre d'applications et le nombre de développeurs, ce qui impacte considérablement votre capacité à livrer. Afin d'éviter cela, on utilise **la technique dite du "Trunk Based Development" ou TBD[20]** . Cette pratique est considérée comme obligatoire par les auteurs du livre Continuous Delivery[21] pour avoir une vraie intégration continue. De notre côté, nous sommes moins extrêmes. Nous faisons des branches ayant une durée de vie courte qui permettent de faire des revues de code (technique connue sous le nom de "GitHub flow"), ce qui offre une solution proche ayant ses avantages. Dans les deux cas, cela demande beaucoup de rigueur car **il faut être prêt à livrer votre code à n'importe quel moment** si possible sans tirer une branche de bugs fixes (encore moins avoir une branche "prod" et une branche "dev"). 6

Comment être capable de livrer à n'importe quel moment cette unique branche ? **En utilisant les feature toggles[22] (ou "Feature Flags") qui vous permettent de livrer du code de fonctionnalités non finalisées** ou encore en test avant l'ouverture à l'ensemble des utilisateurs. Cette capacité à activer ou désactiver des fonctionnalités directement en production est aussi un excellent moyen de livrer très régulièrement sans de longues campagnes de QA car il reste possible de désactiver du code si celui-ci provoque des erreurs.

Et conjointement à la pratique TBD, l'ajout de feature toggles vous permet de livrer du code, même non finalisé ou encore en test avant l'ouverture à l'ensemble des utilisateurs (vérifiez tout de même que votre code compile !). Cette capacité à activer ou désactiver des fonctionnalités directement en production est aussi un excellent moyen de livrer très régulièrement, car il reste possible de désactiver du code si celui-ci provoque des erreurs. 7

La livraison, un non-évènement !

Les feature toggles sont aussi un des éléments permettant de **transformer les livraisons en prod en des "non évènements"** : un non évènement peut se faire sans soucis à 17h un vendredi par exemple. Pour cela, il va falloir **mettre en place une infrastructure**

(17) Continuous Integration

(18) Afin de permettre des revues de code

(19) Voir article "Top 10 Best Practices for Jenkins Pipeline Plugin" sur le site cloudbees.com

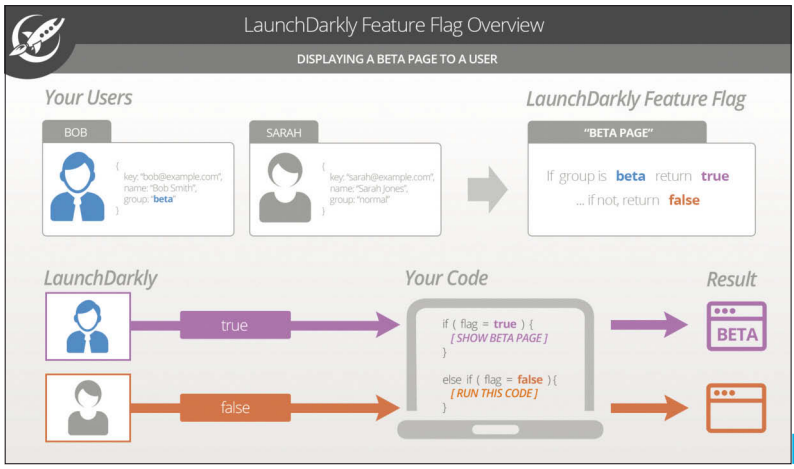
(20) voir le site trunkbaseddevelopment.com

(21) voir le livre Continuous Delivery par Jez Humble et David Farley, 2010

(22) voir l'article sur les Feature Toggles sur le site martinfowler.com ou le site featureflags.io

as code[23] permettant de gérer la création des environnements, l'installation et le démarrage de l'application ou encore les mises à jour du schéma de vos bases de données. L'objectif est que vos livraisons sur vos différents environnements soient totalement automatisées et exécutables via un simple clic. Si vous en êtes encore à éteindre votre serveur d'application, y mettre votre package puis redémarrer l'ensemble manuellement, ne pensez même pas faire des micro-services. Commencez par mettre en place un outil d'automatisation comme Ansible, Puppet ou Chef permettant au passage de décrire et versionner vos environnements. Dans notre cas, nous utilisons Docker et ses images pour créer des services très facilement déployables. Et pour gérer nos environnements, nous utilisons des fichiers de description "docker compose" versionnés sur Git en association avec Vault pour la gestion des secrets (mots de passe, certificats ...). Jenkins nous permet de lancer la construction du service puis de l'image Docker mais aussi son déploiement en un clic.

(23) Voir l'infrastructure as code ou encore l'article "Infrastructure as Code: A Reason to Smile" sur le site thoughtworks.com



7 *Launch darkly*

DevOps

Sans le savoir, nous avons parlé d'un élément organisationnel indispensable d'une architecture micro-services : **une culture DevOps. Si vous voulez réussir cette automatisation, il va falloir travailler en collaboration avec la production.** Et si vous pensez que faire des JIRAs sera suffisant pour mettre en place l'outillage ou le déploiement, eh bien vous vous trompez. Des pratiques DevOps fortes sont les clés d'une capacité à livrer rapidement et régulièrement sans aucun souci. Les plus avancés sont même capables de livrer directement en prod sans aucun environnement de tests ou d'intégrations (oui c'est possible ![24]) •

(24) Voir la présentation "quality production" sur le site infoq.com

La suite de cet article dans le prochain numéro

Christophe Villeneuve
Consultant IT pour Ausy, Mozilla Rep, auteur du livre « Drupal avancé » aux éditions Eyrolles et auteur aux Éditions ENI, PHPère des elePHPants PHP, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora...

IFTTT dans la domotique

niveau 100

IFTTT signifie IF This Then That. C'est un service web gratuit. Il permet d'interconnecter des applications et des matériels pour les faire communiquer. Le but étant de créer des usages et expériences.

IFTTT est un service d'automatisation. Il est disponible directement en ligne. Il utilise le principe des scénarii, c'est à dire lorsqu'un événement se produit avec un service alors un évènement, appelé « déclencheur », se produit, suivi d'une action automatique.

Par exemple vous suivez un flux d'actualité comme votre magazine préféré « Programmez » et vous souhaitez relayer les actualités sur vos différents réseaux de communications pour en faire profiter vos abonnés. IFTTT va répondre à ce besoin de relais au lieu de retaper ou refaire les messages.

Par défaut, IFTTT embarque une compatibilité avec un grand nombre de services et de sites, dont la liste augmente régulièrement. Vous trouverez des sites comme Twitter, Facebook, Wikipedia, Pinterest… De plus il est compatible avec de nombreux objets connectés de type « connected home » :

- Le système d'irrigation connecté de GreenIQ ;
- Les objets connectés GreenWave ;
- Le système d'alarme iSmartAlarm ;
- Le système de sécurité et domotique MyFox HomeControl ;
- Les stations météo Netatmo ;
- Les sondes pour plantes Parrot Flower Power.
- Etc.

Contourner IFTTT

Vous pouvez aller plus loin en contournant IFTTT car il permet de répondre aux attentes des makers, bidouilleurs souhaitant réaliser eux-mêmes la connexion avec des objets non listés.

Depuis l'apparition de la domotique, de nombreux développeurs ont trouvé des astuces pour utiliser ce réseau à partir des méthodes existantes pour mettre en lien IFTTT et un serveur domotique tout en utilisant les canaux existants. Toutes les fonctionnalités ne sont pas disponibles par défaut.

La technologie développée à l'époque utilisait le principe de l'API avec un accès externe et public. En effet, le serveur domotique avec lequel vous souhaitiez établir la connexion possédait un système pour interroger son API par une adresse HTTP, avec des identifiants pour éviter que l'API ne soit considérée comme personne non autorisée. Vous trouverez de nombreux articles en ligne montrant différentes manières de pallier l'absence d'un canal dédié aux API sur IFTTT. Nous présentons certaines de ces solutions que vous pouvez reproduire.

Elément déclencheur : Google Drive

Zibase est une box domotique compatible avec de nombreux protocoles Radio comme il en existe de nombreux modèles équivalents tels que Vera Lite, MyFox Home, Zipabox, Thomson Thombox, Somfy, Figaro Home, ...). Avec la Zibase, l'utilisation du mode balance Withings a été rendue possible en modifiant une action IFTTT : au lieu d'envoyer un fichier sur Google Drive en précisant l'URL d'un fichier, c'est l'URL utilisée qui sera envoyée vers la connexion Zibase, une commande brute.

Le résultat obtenu se décompose de la façon suivante : Zibase reçoit une commande à partir d'une URL appelée et est prise en compte. Du côté Google Drive, rien n'est envoyé puisque l'adresse n'envoie pas de fichier quand elle est sollicitée. Cette technique permet d'interfacer n'importe quel canal IFTTT sur une box domotique. Il va récupérer les données appelées « Ingrédients » pour alimenter les valeurs d'une sonde virtuelle.

Source : http://missquellegoule.blogspot.fr/2015/02/balance-withings-et-zibase-via-ifttt-et.html

Une feuille de calcul

La feuille de calcul est un autre moyen de communiquer avec un objet domotique, c'est-à-dire une feuille de calcul avec un script, pour construire une URL, qui enverra les informations à l'API pour communiquer les informations au serveur domotique.

Il faudra héberger la feuille de calcul sur Google Drive. Il aura aussi la charge de gérer la partie authentification, qui sera nécessaire pour communiquer avec votre box.

Du côté des mises à jour, vous devez enregistrer les identifiants de l'API IFTTT qui peut mettre à jour cette feuille quand c'est nécessaire. Toutefois, pour la partie authentification, les identifiants de l'API seront déportés de votre box vers Google Drive pour éviter par exemple de les rendre accessibles vers tous les internautes.

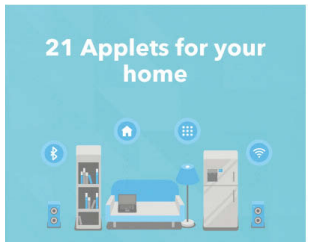
Source : http://www.abavala.com/ifttt-2-api-lien-entre-ifttt-et-les-api-de-votre-systeme-domotique/

Le canal domotique

IFTTT a répondu aux nombreuses demandes des utilisateurs en créant différentes collections comme la maison (Home), l'Internet des Objets (IoT), la voiture (car), les makers… Le but est d'avoir une interface directe entre IFTTT et les API sans utiliser de méthodes détournées. Ainsi, vous gérez directement les requêtes entrantes et sortantes. 1 2

À partir de l'interface

L'article montrera la décomposition du canal « maker » sur lequel vous pourrez vous appuyer pour utiliser les autres collections. Nous



1

2

utilisons une prise connectée pour allumer automatiquement un appareil électrique comme une cafetière ou une bouilloire, ainsi lorsque vous sortez du lit le matin, votre boisson chaude sera prête. 3

La majorité des appareils communiquent en socket, c'est pourquoi il faut configurer l'application de votre appareil pour qu'elle puisse recevoir des instructions IFTTT. Ainsi vous pourrez générer un jeton (code PIN IFTTT) pour la connexion.

L'étape suivante passera par le site IFTTT, en créant une nouvelle Applet à partir de l'objet que vous avez choisi pour arriver à un écran qui se décomposera en 2 éléments :

- Un déclencheur : appelé à partir d'un « IF »
- Une action : qui exécute après le « THEN » 4

Le déclencheur « IF » se comporte comme une adresse spécifique à atteindre. Vous avez la possibilité d'ajouter plusieurs arguments (maximum 3) à l'URL publique, ainsi que la date et l'heure qui sera l'élément d'exécution. L'action « THEN » reste identique à la technique mise en place depuis l'origine du site, c'est-à-dire l'envoi d'une requête vers l'URL enregistrée avec éventuellement les arguments saisis. Il est souvent demandé à cette étape de saisir le token de votre appareil, que vous avez généré précédemment pour que celui-ci puisse accepter la requête envoyée.

Après la sauvegarde, la connexion devient opérationnelle.

Construire son déclencheur

IFTTT propose un canal « maker » pour l'intégrer à vos projets personnels. Nous utiliserons un Raspberry PI comme un objet connecté pour vous montrer le principe de communication.

Préparation

Pour commencer, la création d'applets compatibles avec tous les périphériques ou applications pouvant effectuer ou recevoir une requête web, passera par le lien suivant https://ifttt.com/maker_webhooks pour obtenir une URL et une clé indispensable pour le bon fonctionnement. 5

À partir du menu « settings », vous obtenez un écran comprenant une URL avec une clé secrète déjà renseignée qu'il faudra ajouter à l'interface de votre appareil. Il est possible de personnaliser quelques informations à partir de l'interface, comme le nom de l'évènement et ses arguments. 6 7

Comme le montre la capture, vous pouvez non seulement déclencher une URL, mais également une commande « curl » en ligne de commande.

Il est important de noter la clé qui commence par gV4x41Z

Quelle que soit la méthode utilisée, vous pouvez envoyer jusqu'à 3 arguments au format JSON. Après avoir effectué les modifications, vérifiez la (bonne) connexion avec un test.

Votre objet connecté

La configuration de votre appareil s'effectue en mode terminal, c'est-à-dire que vous devez vous connecter à celui-ci en SSH (sous Linux ou Mac) ou avec Putty (sous Windows).

Nous transformons notre Raspberry Pi en Home Assistant, en installant l'application :

```
$ pip3 Install homeassistant
```

À partir de votre navigateur, tapez l'URL suivante pour vérifier que celui-ci est correctement installé

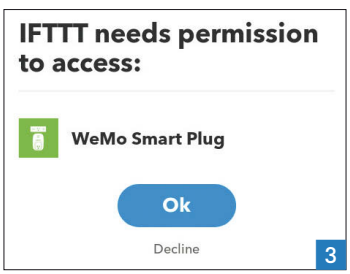
http://IP_De_votre_PI:8123/states

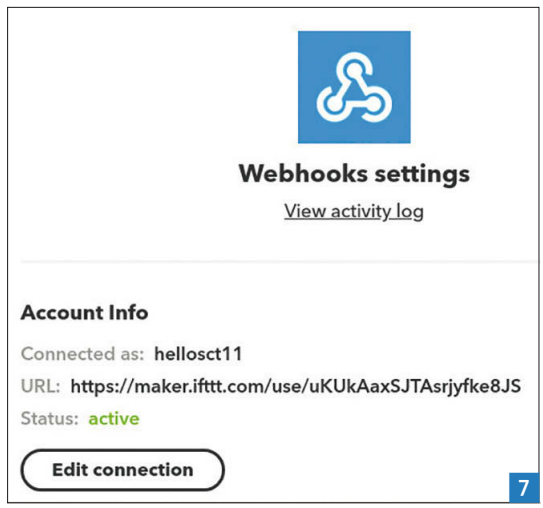
La connexion entre votre Raspberry PI et IFTTT s'effectue en éditant le fichier suivant :

```
$ nano ~/.homeassistant/configuration.yaml
```

Ajoutez les lignes suivantes :

```
ifttt :
Key : VOTRE_CLE_API
```





PS : La clé est celle que vous avez notée précédemment
Relancez le programme home-assistant

```
hass –open-ui
```

Votre Applet

La création de votre applet correspond à la création d'une application pour communiquer entre IFTTT et votre Home-assistant (Raspberry PI). Utilisez le lien suivant https://ifttt.com/my_applets 8
À partir du bouton « + this », choisissez le service **Webhooks** 9
L'opération s'effectuera en 6 étapes
Tout d'abord en créant un nom de projet. Nous choisissons homeassistant 10
Après avoir cliqué sur « create trigger », vous choisirez « +that » que vous sélectionnez 11
Le service de destination sera dans notre exemple Google Drive 12
Il vous sera demandé quelques informations complémentaires comme la connexion à votre compte et d'accepter les permissions IFTTT auprès de ce compte.
Vous compléterez le formulaire avec :

- un nom de projet,
- les colonnes dont vous avez besoin,
- Le chemin d'accès au fichier. 13

Bien entendu, chaque ligne peut accepter des éléments dynamiques supplémentaires, ce qui sera la dernière étape de configuration.

Le fonctionnement

Le fonctionnement permet de tester si la connexion entre IFTTT et votre Raspberry PI se déroule sans problème. Vous pouvez appeler le service que vous venez de créer avec « CALL SERVICE ».
Si votre test de connexion est OK, vous avez la possibilité de configurer un mode automatique de votre Raspberry PI, c'est-à-dire à chaque action de celui-ci, il enverra une information à IFTTT qui utilisera le service en sauvegardant dans Google Drive.

```
automation:
- alias: Démarrage
trigger:
platform: homeassistant
event: start
action:
service: ifttt.trigger
data: {« event »:"HASS_START_UP », « value1 »:"Start up »}
```

Autre possibilité

Comme la connexion a fonctionné dans un sens, il est possible de déclencher une opération à partir d'une action web. Pour cela, la procédure reste identique avec la même technique et beaucoup de choix possibles. Lorsque vous aurez choisi l'élément déclencheur avec le « + this », vous choisirez le « +that » de la manière suivante : 9

- Vous effectuez la recherche pour trouver le service **Webhooks** dont vous renseignerez les différents champs
- URL : https://votre_ip:8123/api/services/homeassistant/restart?api_password=Votre_Mot_Passe
- Methode : POST
- Éventuellement un message

L'évolution sera prise en compte après le reboot du home assistant

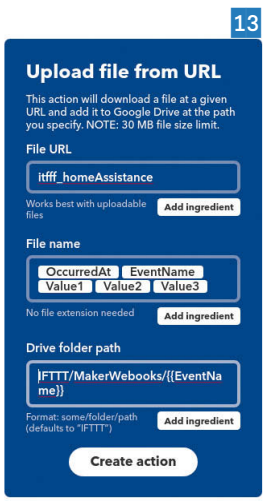
Pour aller plus loin

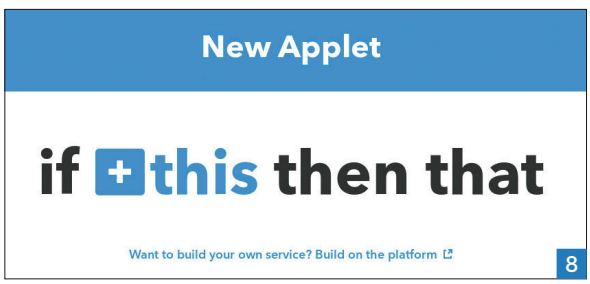
Le site IFTTT propose de nombreuses API, gérées par des flux de données automatiques, appelés Channels, composés de liens et de règles décrivant leurs interactions. En associant les uns aux autres, on arrive à créer de nombreux cas de combinaisons de fonctionnalités et d'automatismes, appelés Applets. C'est pourquoi, une documentation est disponible pour utiliser l'API dans vos propres projets Site : https://platform.ifttt.com/docs

Conclusion

Le site IFTTT a ouvert son API pour vous proposer de nombreux services, aussi bien dans un sens que dans l'autre. Ainsi, quelle que soit la domotique que vous possédez, ou en utilisant le projet Home assistant, prêt à l'emploi ou en le construisant, vous pouvez générer des actions, obtenir des informations à distance sans que vous soyez proche de l'appareil.
À vous de jouer maintenant. •

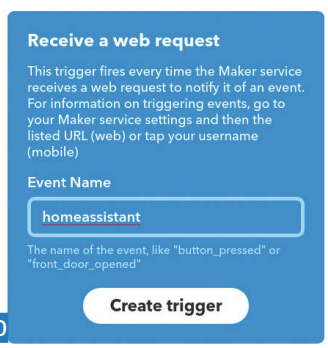
13



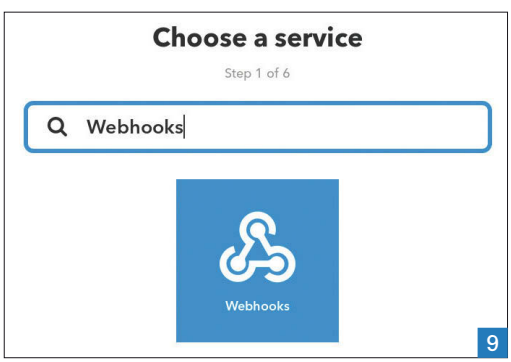


8

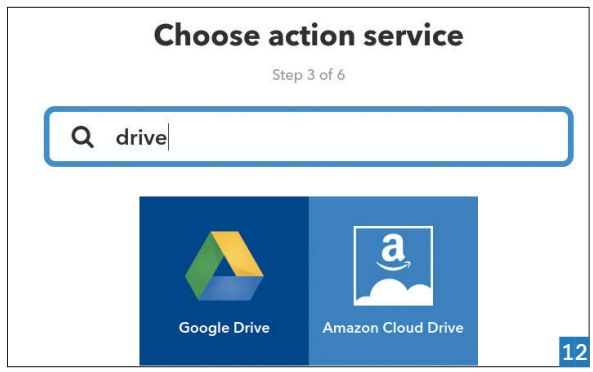
11



10



9



12

Jordan NOURRY
Développeur
@La combe du lion vert

Fouad JADOUANI
Développeur
@Société Générale

Refactoring de Legacy Code avec Programmation Fonctionnelle en pur JavaScript

Partie 3

La programmation fonctionnelle est devenue un sujet très tendance ces dernières années. Complètement méconnue des développeurs débutants, pour la plupart concentrés sur la programmation orientée objet, elle permet de rendre le code plus concis, plus simple et plus expressif.
Comment appréhender ce style de programmation ? Quels sont ses points forts ? Et surtout comment « refactorer » du code legacy en y ajoutant de la programmation fonctionnelle ? Pour étayer nos propos nous allons nous appuyer sur le langage JavaScript, dont la communauté a énormément travaillé ces 3 dernières années à l'intégration de la programmation fonctionnelle dans notre environnement professionnel !

niveau 200

Suite et fin.

LES MONOIDES

Les monoïdes sont avant tout une structure mathématique, comme beaucoup de choses de très bon goût en informatique. Un monoïde c'est un ensemble muni d'une loi de composition interne associative et d'un élément neutre.
En informatique cela peut être représenté par une fonction prenant en paramètre un élément d'un type puis un autre élément du même type. En résultat de l'opération, la fonction retourne un élément du même type. Cela ne laisse pas de place au hasard !
Par exemple, il existe des monoïdes naturels, comme les entiers :

```
const increment = num => num + 1
const square = num => num * num
const modulo10 = num => num % 10
let result = modulo10( square( increment(2) ) ); // <- 9
```

Cela s'appelle la clôture d'une opération, **l'ensemble est fermé sur l'opération et c'est donc la première propriété d'un monoïde.**
La deuxième propriété est la composition avec des éléments du même ensemble, cela s'appelle **l'Associativité**. C'est à dire que nous pouvons mettre les parenthèses où nous voulons, cela ne change rien.

```
const result = (3 + 5) + 2;
// est identique à
const result = 3 + (5 + 2);
```

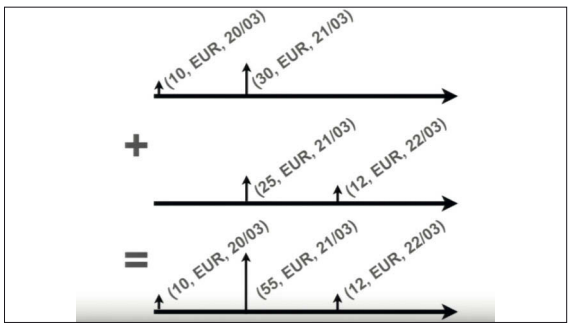
A ne surtout pas confondre avec la commutativité qui elle change le résultat d'une opération.
La troisième propriété est l'élément neutre. L'« absence », le « un » et le « plusieurs » sont donc encapsulés dans la structure.
Appliquer cette structure sur la diversité des cas réels métiers permet de baisser le niveau de complexité de l'ensemble. **Ceci en réunissant les éléments de cet ensemble au sein d'une même structure monoïdale.**

On passe d'un produit cartésien à gauche, à un produit de cas unique à droite, on reste donc sur un cas systématique. Le bénéfice est qu'on peut scaler en complexité plus facilement si on met en place cette structure algébrique dans notre code. **L'objectif est donc de les repérer et de les mettre en place.** Donc, comme nous avons vu précédemment il y a des monoïdes naturels :

```
Les entiers :   int + int : int
  (3+5) + 2 = 3 + (5+2)
  Element neutre  0
List :
  (.) + (.;.) = (.;.;.)
  (a) + (,c) = (a, b) + (c)
  Element neutre ()
Strings:
  "hello" + "world"
  "jor" + "dan" = "jorda" + "n"
  Element neutre ""
```

Même si les monoïdes semblent simples au premier abord, **ils permettent de monter en complexité, en volume de données et en fréquence de données**. Car c'est la clé pour pouvoir traiter des comportements complexes. On peut **pressentir les monoïdes dans notre base code,** car ils sont fortement liés aux notions de ***reduce, merge, add, compose, combine***. Ils sont aussi présents quand on parle de calcul partagé sur plusieurs machines qu'on doit rassembler ensuite. Pratique aussi quand on a une notion de temporalité et qu'on veut précalculer sur une certaine plage de temps et consolider avec le calcul d'une autre plage de temps. Voici des exemples un peu plus « métier », qu'on peut facilement retrouver dans des contextes financiers, bancaires ou assurances :

```
Money Quantity: (25, EUR) + (25, EUR) = (50, EUR)
Cash Flow: (25, EUR, today)
Cash sequence:
```



L'utilisation des monoïdes sur des cas « métier » nous permet d'accéder à une arithmétique d'objet.
D'autres structures mathématiques sont disponibles, avec des propriétés diverses et variées, comme par exemple, **les Vector Space ou encore les Cyclic Group**, qu'on vous invite à regarder.
SOURCE : https://www.youtube.com/watch?v=_jr8E5GVnBA

FUNCTORS

En mathématique pure, un « functor » est une fonction qui prend une valeur et une fonction ; la valeur peut être un tableau par exemple. Le functor est donc responsable de sortir les valeurs du tableau individuellement et de les passer un à un à la fonction passée en paramètre. Puis de créer une nouvelle structure pour ces valeurs et de la retourner quand toutes les valeurs ont été traitées. Suivant cette définition, *filter* et *map* seraient des functors, mais pas *forEach* qui ne retourne rien. Pour autant cette définition n'est pas vraie dans le monde du développement, car cette définition en fait un synonyme du concept de high-order function. Donc dans un langage de programmation fonctionnelle, **un functor est un objet implémentant map**. Vous avez sûrement déjà utilisé les functors sans vous en rendre compte. Un exemple bien connu est *Array* qui permet d'itérer sur ses items à travers la méthode *map*.

```
[1, 2, 3, 4].map(x => x * 2); //    [2, 4, 6, 8]
```

L'utilisation de functor ne s'arrête pas à leurs utilisations sur des tableaux, il peut être utilisé sur plus de structures, comme les : object, valeur, fonction observable…
Pour aller plus loin dans l'explication à travers des exemples concrets on vous conseille de lire cet article*(11)* qui référence d'autres ressources intéressantes sur les **functors**.

EXEMPLE CONCRET

Dans l'exemple ci-dessous vous trouverez plusieurs fonctions qui permettent d'avoir des informations sur une liste de voitures données en entrée. Vous avez la version "Legacy" en commentaire et la version « refactorée » en version plus fonctionnelle :

```
/*function maxPrice(cars) {
  var maxPrice = 0;
  for (var i = 0; i < cars.length; i++) {
    if (cars[i].price > maxPrice) {
      maxPrice = cars[i].price;
    }
  }
  return maxPrice;
}*/

const maxPrice = cars => cars.reduce((maxPrice, car) => car.price > maxPrice
 ? car.price : maxPrice, 0);
;
/*function minPrice(cars) {
  var minPrice = 0;
  for (var i = 0; i < cars.length; i++) {
    if (minPrice === 0 || cars[i].price < minPrice) {
      minPrice = cars[i].price;
    }
  }
  return minPrice;
}*/

const minPrice = cars => cars.reduce((minPrice, car) => minPrice === 0 ||
 car.price < minPrice ? car.price : minPrice, 0);
;
/*function averagePrice(cars) {
  var total = 0;
  var nbCars = cars.length;
  for (var i = 0; i < cars.length; i++) {
    total = total + cars[i].price;
  };
  return total / nbCars;
}*/
;
const averagePrice = cars => cars.reduce((total, car) => total += car.price, 0)
 / cars.length;
;
/*function averagePriceByBrand(cars, brand) {
  var total = 0;
  var nbBrandCars = 0;
  for (var i = 0; i < cars.length; i++) {
    if (cars[i].brand === brand) {
      nbBrandCars = nbBrandCars + 1;
      total = total + cars[i].price;
    }
  }
  return total / nbBrandCars;
}*/

const nbBrandCars = cars => brand => cars.filter(car => car.brand ===
 brand).length;

const totalByBrand = cars => brand => cars.filter(car => car.brand ===
 brand).reduce((total, car) => total += car.price, 0);
;
const averagePriceByBrand = cars => brand => totalByBrand(cars)(brand) /
 nbBrandCars(cars)(brand);
;
describe('My cars collection compute indicators', () => {
  const cars = [
```

(11) https://medium.com/@dtinth/what-is-a-functor-dcf510b098b6

```
    {brand: 'Renault', model: 'Renault Twizy', price: 7540},
    {brand: 'Alfa Romeo', model: 'Mito', price: 15490},
    {brand: 'Volkswagen', model: 'Golf', price: 21990},
    {brand: 'Porsche', model: '718', price: 55040},
    {brand: 'BMW', model: 'Serie 1', price: 23200},
    {brand: 'Audi', model: 'A4', price: 33670},
    {brand: 'Porsche', model: 'Macan', price: 58835},
    {brand: 'Volkswagen', model: 'Polo', price: 14430},
];

it('max price', () => {
    maxPrice(cars).should.be.deep.equals(58835);
});

it('min price', () => {
    minPrice(cars).should.be.deep.equals(7540);
});

it('average price', () => {
    averagePrice(cars).should.be.deep.equals(28774.375);
});

it('average price for Porsche car only', () => {
    //averagePriceByBrand(cars, 'Porsche').should.be.deep.equals(56937.5) ;
    averagePriceByBrand(cars)('Porsche').should.be.deep.equals(56937.5);
});
});
```

Dans cet exemple on peut remarquer **une implémentation plus concise de presque 90% en programmation fonctionnelle, par rapport à la version impérative.**

VÉRIFIER LE CODE JS AVEC HASKELL

Nous ne pouvions pas finir cet article sans parler de Haskell, qui est un langage de référence dans le monde de la programmation fonctionnelle. Nous ne pouvions pas non plus finir cet article sans parler d'une personne qui a réussi à réunir le monde JS et celui d'Haskell sur le point de la programmation fonctionnelle dans un superbe "Live Coding" que vous pouvez retrouver ici :
https://www.youtube.com/watch?v=lQ1kDpGeoCk
Vous y verrez notamment que grâce à un peu de sucre syntaxique, nous pouvons très bien traduire notre code JS en code HASKELL. Ce qui est pour DETANT Xavier la preuve que ce style de programmation qu'on vous propose aujourd'hui est très proche du fameux langage pur.

CONCLUSION

A travers cet article nous avons pu découvrir les quelques notions de la programmation fonctionnelle, ce qui permet de démystifier et de se familiariser avec ce paradigme de programmation.
Nous avons vu au travers d'exemples que nous pouvons tirer plusieurs avantages de cette méthode de développement qui permet notamment d'obtenir un code :

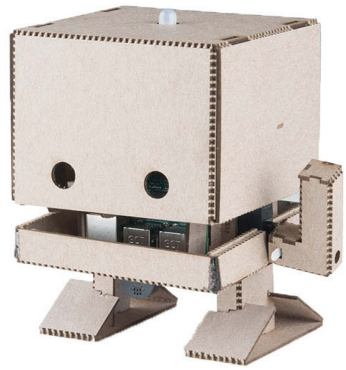
- **plus lisible ;**
- **Facile à réutiliser et à composer ;**
- **Plus sécurisé;**
- **Facile a tester.**

Bien sûr, la lecture de cet article ne fait pas de vous un développeur en programmation fonctionnelle, car d'autres notions restent à découvrir et à approfondir. Pour aller plus loin dans cet apprentissage, il est nécessaire de s'immerger et d'apprendre avec un langage purement fonctionnel tel qu'Haskell. Voici un livre pour débuter Haskell "Learn You A Haskell For Great Good" et sa traduction en français (http://lyah.haskell.fr/). Une fois les quelques notions abordées dans cet article acquises, nous vous invitons à commencer à intégrer de la programmation fonctionnelle dans vos codes. **L'apprentissage de ce style de programmation vous aidera aussi fortement à écrire du meilleur code en programmation objet**. D'ailleurs, la programmation fonctionnelle et la programmation objet ne sont pas ennemies, il faut trouver le juste milieu entre l'utilisation des deux styles dans vos projets. **L'Architecture Hexagonale est d'ailleurs une grande alliée pour trouver ce juste milieu !** •

Clémence Lebrun
Developer & Developer Advocate Europe
IBM
Clemence.lebrun@fr.ibm.com

TJBot

Un robot cognitif, programmable et fun !



TJBot est le premier projet open source "Do It Yourself" de la collection IBM Watson Maker Kits. C'est un robot programmable qui utilise les APIs IBM Watson, idéal pour faire ses premiers pas dans le monde de l'intelligence artificielle.

niveau 100

Les curieux pourront apprendre comment facilement intégrer des capacités cognitives dans des objets connectés et ce, de façon ludique. Tout est à faire soi-même, de l'assemblage des composants électroniques à la programmation logicielle en passant par la fabrication du robot, en carton ou en plastique selon la méthode choisie. TJBot peut parler, converser, analyser du texte et des images mais aussi interagir en bougeant le bras ou en allumant sa LED.

Le projet :

TJBot est né fin 2016 au sein d'IBM Research, d'une volonté de créer un projet à des fins éducatives et démonstratives utilisant les différents services IBM Watson. Il a été créé par Maryam Ashoori, PhD IBM Research AI. Le nom « TJ Bot » est un hommage à Thomas J.Watson, le premier Président et CEO d'IBM. C'est un projet en constante évolution qui bénéficie des apports de la communauté open source. Il existe aujourd'hui de nombreux tutoriels en ligne et articles qui traitent le sujet du TJBot et proposent différentes versions évoluées.

Assembler le TJBot :

Afin d'assembler un TJBot classique, vous aurez besoin de matériel. Sachez que cette liste n'est pas exhaustive. Elle permet de créer un TJBot avec ses fonctions basiques, mais certains tutoriels nécessiteront du matériel additionnel.

- **Raspberry Pi 3** + carte SD avec NOOBS préinstallé ;
- **Microphone USB ;**
- **Mini enceinte Bluetooth**. Toute enceinte avec une entrée audio jack 3.5 mm ou Bluetooth fonctionnera. Notez que si vous souhaitez utiliser le jack, vous aurez aussi besoin d'un adaptateur USB Audio pour éviter les interférences audio avec la LED ;
- **Servo Motor**. Equipé d'une prise 3 contacts, on retrouve 3 câbles : un rouge (celui du milieu), un marron ou noir qui correspond à la MASSE/GND (borne négative de l'alimentation), et un orange, jaune, blanc ou bleu pour la commande de position du Servo Motor. Le Servo Motor permet le mouvement du bras du TJBot ;
- **NeoPixel RGB LED (8mm) :** Si vous utilisez un autre type de LED, vous pourriez avoir besoin d'une résistance, celle-ci n'en a pas besoin ;
- **Câbles Femelle - Femelle** pour les connections ;
- **Raspberry Pi Camera** : 5MP ou 8MP ;
- **Pour contrôler le Raspberry Pi, il est conseillé d'avoir à disposition** : un clavier USB, une souris USB ainsi qu'un écran avec une prise HDMI.

Il existe des revendeurs qui proposent des kits contenant toutes les pièces nécessaires ainsi que le corps du TJBot cartonné prédécoupé. Si vous optez pour cette option, il vous suffira donc de procéder ensuite aux étapes de pliage et à l'assemblage électronique en suivant les vidéos explicatives.

Chez Sparkfun : https://www.sparkfun.com/products/14515
Chez Adafruit : https://www.adafruit.com/product/3462

Vous pouvez aussi vous procurer vous-même les pièces électroniques nécessaires et fabriquer le corps de TJBot en le personnalisant selon vos envies :

Option 1 : avec une découpe Laser

Pour effectuer vous-même votre découpe laser, on recommande d'utiliser un carton « Chipboard », plus mince et plus résistant que le carton ondulé, il est aussi plus facile à découper avec un cutter laser. Vous pourrez ainsi faire des perforations qui faciliteront le pliage du carton. 1

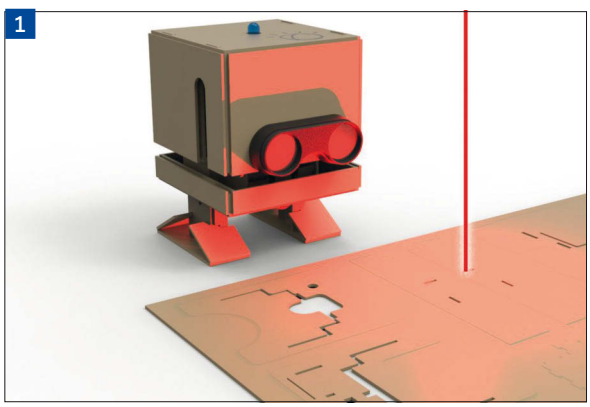
Si vous n'avez pas vous-même accès à un cutter laser, sachez qu'il existe des services en ligne pour le faire. Des cutter laser peuvent aussi être mis à disposition dans des associations ou des clubs de Makers par exemple. Les dimensions recommandées du chipboard sont 18" x 18" x .080" (457mm x 457mm x 2mm). 2

Trois formats différents sont proposés au téléchargement : https://ibmtjbot.github.io/images/TJBotLasercutLines.zip 3

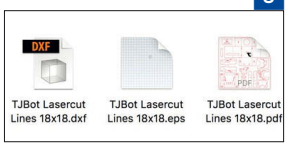
Etapes de pliages :

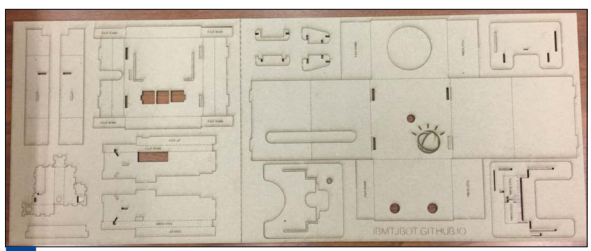
Une fois la découpe faite, vous allez pouvoir procéder au pliage. Armez-vous de patience pour cette étape car une erreur est vite arrivée. Prenez le temps d'effectuer les étapes les unes après les

1



3



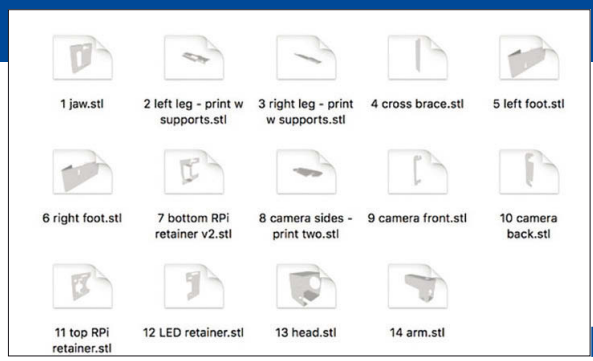


2

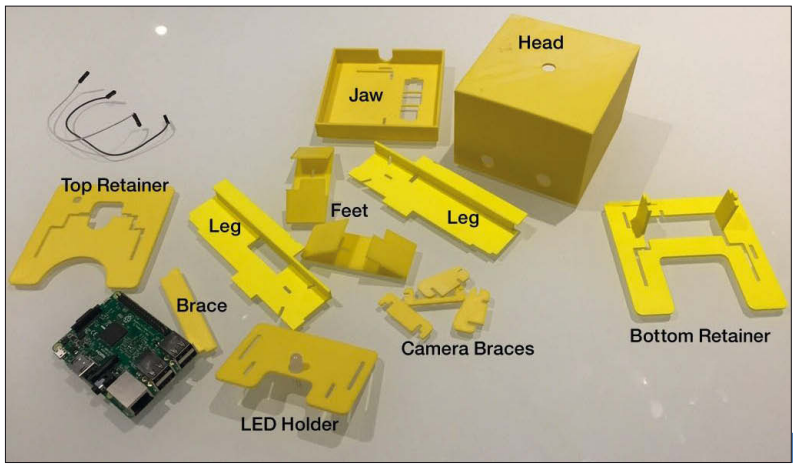
autres. Un mauvais pliage pourrait endommager le carton et il faudrait alors utiliser du ruban adhésif pour réparer ou bien recommencer la découpe.
Il est recommandé d'utiliser le bord d'une table ou bien une règle en métal comme levier pour le pliage. Evitez de mettre trop de pression sur le pliage pour ne pas déchirer le carton. Assurez-vous de plier dans le bon sens avant de commencer. Les fichiers à télécharger, les vidéos ainsi que le détail des étapes de pliages sont disponibles ici :
https://www.instructables.com/id/Build-TJ-Bot-Out-of-Cardboard/

Option 2 : avec une imprimante 3D

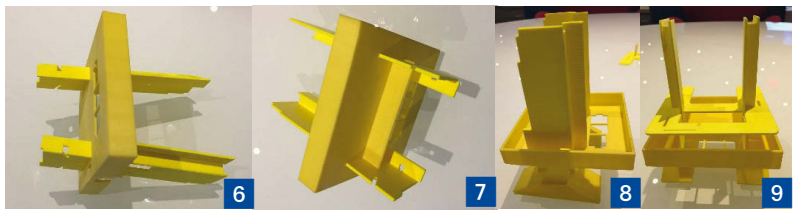
La seconde option est d'imprimer le corps du TJBot avec une imprimante 3D. De même que pour la découpe laser, vous n'avez pas forcément besoin de posséder votre propre imprimante, vous pouvez faire ce travail dans un club de Makers par exemple ou encore utiliser un service en ligne pour imprimer vos pièces. Notez que l'impression demande plusieurs heures (le temps d'impression peut varier en fonction de votre imprimante) mais vous pouvez imprimer les différentes pièces en plusieurs fois.
Vous avez accès aux fichiers sources de conception qui sont en libre téléchargement. https://ibmtjbot.github.io/images/TJBot3DPrintFiles.zip 4
Il y a un fichier STL par pièce mais vous pouvez imprimer plusieurs pièces à la fois : 5
Attention, certaines pièces doivent être imprimées en utilisant des supports.
En faisant le choix de l'impression 3D, vous vous laissez davantage de liberté quant au choix des couleurs et de l'aspect de chaque pièce du TJBot. Sur les communautés en ligne, de nombreux curieux ont posté leur travail et certains ont su faire preuve d'une grande créativité !
Une fois les pièces imprimées, vous pouvez procéder à l'assemblage. Vous pouvez avoir besoin de raboter certaines pièces selon la qualité de votre impression, assurez-vous d'avoir un cutter à disposition pour cette étape.
Commencez par insérer les deux jambes dans la mâchoire, dans le trou en forme de « L », par le haut de la mâchoire. Il est possible que vous ayez à forcer un peu, ou à raboter le trou avec votre cutter. (6) Pièces : Leg (x2), Jaw.
Insérez ensuite le fixateur des jambes sous la mâchoire qui permet de maintenir les jambes parallèles. Deux trous rectangulaires sont prévus à cet effet dans les jambes. Cette pièce fixatrice doit être en contact avec la mâchoire. (7) Pièce : Brace.
Insérez ensuite les pieds dans les encoches qui se situent au bas des jambes. La pièce s'insère depuis l'extérieur de la jambe. (8) Pièces : Feet (x2).
Vous pouvez maintenant mettre le TJBot debout !
C'est le moment de placer votre Servo Motor dans le trou sur la gauche de la mâchoire. De cette façon vous pourrez facilement emboîter la pièce suivante.
Insérez ensuite par le haut des jambes la pièce de consolidation (les bras de support vers le bas) au-dessus de la mâchoire et faites-la glisser par le bas, de façon à ce qu'elle s'emboîte sur le haut du Servo Motor ainsi que dans la mâchoire. (9) Cette pièce servira également de support pour la caméra. Pièce : Bottom Retainer.
Connectez la caméra Pi sur le port prévu sur le Raspberry Pi (10).
Il faut maintenant placer votre Raspberry Pi dans le corps du TJBot.

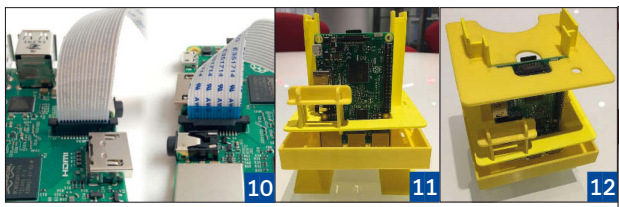


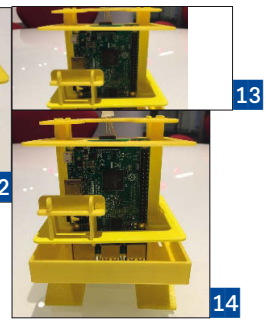
4



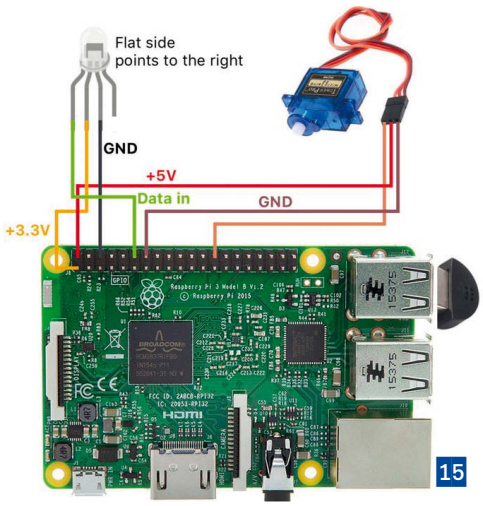
5


6 7 8 9

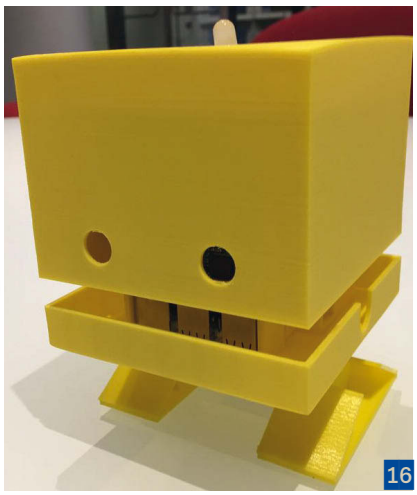

10 11 12


13 14

Insérez le Raspberry Pi en orientant les ports USB vers le bas. Ceux-ci s'emboîtent dans les trous prévus dans le bas de la mâchoire. Assurez-vous que les broches sont orientées vers la face avant du TJBot comme sur la photo (11).
Vous allez maintenant fixer la caméra à l'aide des 4 petites pièces de support. Faites glisser les deux pièces de support latérales dans les encoches de la pièce de consolidation puis faites glisser votre caméra. Fixez ensuite les deux pièces de supports supérieurs de façon à ce que la caméra soit droite et immobile. Pièces : Camera Braces.
Faites glisser la seconde pièce de consolidation au-dessus du Raspberry Pi à travers les trous en forme de L. Assurez-vous que le petit trou est à votre droite quand vous regardez le TJBot de face. Descendez la pièce jusqu'à ce qu'elle soit en contact avec le Raspberry Pi (12). Pièce : Top Retainer.
Saisissez la troisième pièce de consolidation et insérez la LED dans le trou du milieu (13). Faites glisser cette pièce par le haut des jambes de façon à ce que seule une petite partie des jambes ne dépasse (14). Pièce : LED Holder.



15


16

Avec les câbles femelle-femelles vous pouvez maintenant connecter la LED et le Servo Motor. Branchez également le micro USB. Reportez-vous au schéma suivant (15) :

Vous pouvez installer la tête du TJBot, attention à faire passer soigneusement la LED par le trou sur le haut de la tête : 16

Le Servo Motor que vous avez installé et connecté dépasse sur le côté du TJBot et vous pouvez maintenant fixer le bras sur le Servo Motor à l'aide d'un tournevis (de préférence aimanté) avec la vis fournie avec le Servo Motor.

Retrouvez toutes les étapes de l'impression et de l'assemblage détaillées ici : https://www.instructables.com/id/Build-a-3D-Printed-TJBot/

Votre TJBot est assemblé !

Donnez vie au TJBot !

Configuration

Maintenant que votre matériel est prêt, il vous faut vous assurer de configurer correctement votre Raspberry Pi et d'installer les dépendances nécessaires. Pour contrôler votre Raspberry Pi, il est possible d'utiliser un moniteur avec une prise HDMI (une télé ou un écran classique), une souris et un clavier. Vous pouvez les connecter directement sur votre TJBot puisque toutes les entrées sont accessibles. Dans la plupart des kits Pi, la carte SD est déjà préchargée avec une image du système d'exploitation Raspberry Pi.

Insérez la carte SD dans le Raspberry Pi, allumez le Raspberry Pi en branchant son alimentation puis suivez les instructions à l'écran pour terminer l'installation du système d'exploitation.

Toutes les étapes de démarrage du Raspberry Pi sont disponibles sur le site officiel : https://www.raspberrypi.org/learning/hardware-guide/ .

Assurez-vous de connecter votre Raspberry Pi au Wifi et de connecter votre speaker Bluetooth ou USB.

Lorsque vous démarrez votre Pi pour la première fois, vous devez effectuer une configuration initiale.

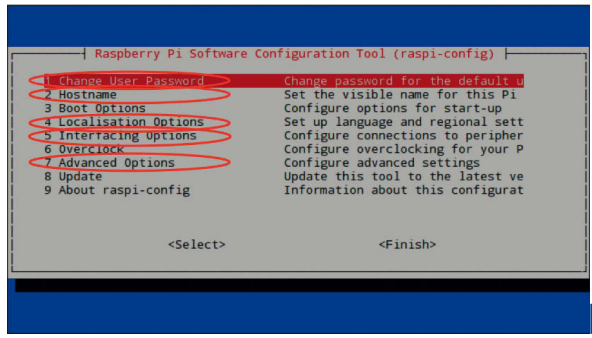
Si vous préférez utiliser une connexion ssh depuis votre ordinateur, plutôt que d'utiliser un clavier et un moniteur, c'est possible :

Ainsi, dans une fenêtre de terminal sous Linux ou MacOS, la commande est la suivante :

```
> ssh pi@raspberrypi.local
```

Le mot de passe par défaut est « *raspberry* ».

Une fois que vous avez accès à la ligne de commande, exécutez la commande suivante :



17

18

```
> sudo raspi-config
```

L'utilitaire de configuration apparaît : 17

Si vous préférez utiliser l'interface graphique, vous pouvez également accéder à l'outil de configuration du Pi : 18

Dans l'outil de configuration, vous devez :

- Changer le mot de passe pour l'utilisateur standard Pi. La valeur par défaut est « raspberry », mais vous devez la modifier pour empêcher tout accès non autorisé à votre Raspberry Pi ;
- Modifier le nom d'hôte, si nécessaire. La valeur par défaut est raspberrypi et vous devriez la changer si vous êtes susceptible de lancer votre Pi sur un réseau contenant d'autres Raspberry Pi ;
- Définir le fuseau horaire, les paramètres régionaux et le clavier ;
- Régler la résolution VNC, il est conseillé de choisir 1280x720 ou plus ;
- Activer les interfaces suivantes : Caméra, SSH, VNC, I2C.

Une fois votre Raspberry Pi configuré, vous pouvez redémarrer.

Mises à jour

Pour mettre à jour votre Pi, vous devez avoir un accès en ligne de commande. Que ce soit en utilisant la connexion ssh ou le terminal directement sur le Pi, effectuez les commandes suivantes :

```
> sudo apt-get update
>  sudo apt-get upgrade –y
```

Avec la commande « apt-get install », installez des packages supplémentaires pour améliorer le fonctionnement du Raspberry Pi.

Netatalk fera apparaître le Raspberry Pi en tant que périphérique réseau dans l'application Finder sur un Mac (cela peut être utile si vous oubliez le nom d'hôte) et le smbclient vous aidera avec le réseau sous Windows :

```
> sudo apt-get install -y netatalk smbclient
```

Dans la suite du projet, nous utiliserons l'outil de développement Node-RED, assurez-vous d'installer les dernières versions de Node-RED et de Node.js :

```
> update-nodejs-and-nodered
```

La commande « sudo systemctl enable nodered.service » configure le système pour qu'il exécute Node-RED au démarrage du système :

```
> sudo systemctl enable nodered.service
```

Redémarrez le Pi :

```
> sudo reboot
```

La dépendance libasound2-dev est requise pour la compilation de node-red-contrib-speakerpi

```
> sudo apt-get install -y libasound2-dev
```

Exécutez les commandes suivantes pour installer les nœuds Node-RED nécessaires.

```
> cd ~/.node-red
> npm install node-red-contrib-speakerpi node-red-contrib-micropi node-red-node-pi-neopixel
> npm install node-red-node-watson node-red-dashboard node-red-contrib-camerapi
> npm install node-red-node-pi-gpiod
```

La dernière modification nécessaire consiste à s'assurer que PiGPIOd est démarré sur le Pi.

```
> sudo nano /etc/rc.local
```

Editez le fichier /etc/rc.local puis ajoutez la ligne / usr / bin / pigpiod -l à la fin du fichier, mais avant la dernière ligne « exit 0 » :
Puis ajoutez la ligne :

```
/usr/bin/pigpiod -l
```

Redémarrez le Pi :

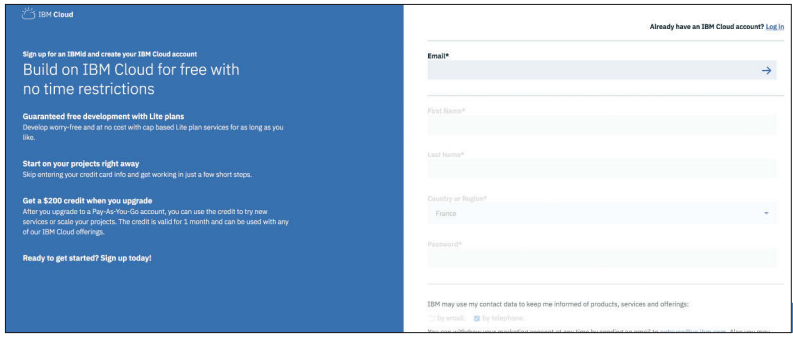
```
> sudo reboot
```

En cas de problème avec votre configuration audio, reportez-vous à : https://github.com/binnes/tobyjnr/wiki/Getting-Sound-to-work-on-the-Raspberry-Pi

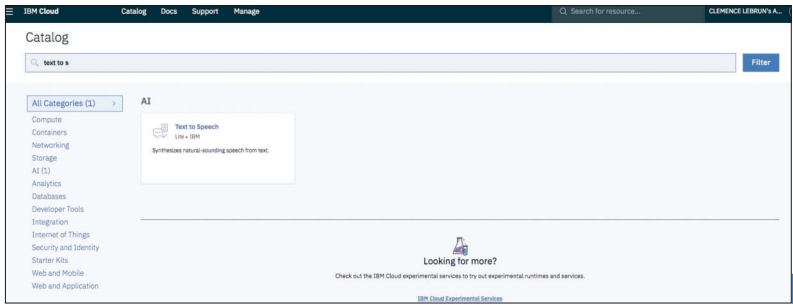
Créez vos services sur IBM Cloud

Afin d'utiliser les services Watson comme Text To Speech, Speech To Text, Visual Recognition, Watson Assistant, Language Translator, vous devez commencer par instancier ces services sur votre compte IBM Cloud.
Si vous n'avez pas encore de compte, créez en un sur : https://console.bluemix.net/registration/ 19
En créant ce compte Lite gratuit, vous accédez à de nombreux services, dont les services Watson qui sont utilisés pour le TJBot.
En parcourant le catalogue, vous pouvez commencer par créer les services suivants :

- Visual Recognition ;
- Text To Speech ;
- Speech To Text ;
- Language Translator ;
- Tone Analyzer.



19



20

Programmez avec Node-RED

Node-RED est un environnement de développement graphique permettant un développement rapide d'applications. Node-RED fonctionne sur Node.JS, il est donc disponible sur le Raspberry Pi, il peut fonctionner en local ou sur le Cloud. Node-RED dispose d'une vaste bibliothèque de « nœuds » qui donnent accès à de nombreuses fonctionnalités, telles que le contrôle des périphériques Raspberry Pi par exemple.
Depuis votre Raspberry Pi, lancez Node-RED soit depuis la ligne de commande, soit depuis le menu des outils de développement puis ouvrez dans un navigateur votre application Node-RED et accédez à l'éditeur Node-RED s'exécutant en local sur le Raspberry Pi.
Vous pouvez utiliser un navigateur depuis votre ordinateur, à condition qu'il puisse communiquer avec le Raspberry Pi via votre réseau local. Le navigateur n'a pas besoin de tourner sur le Raspberry Pi.
L'éditeur utilise le port 1880.
Dans la section Configuration, vous avez défini un nom d'hôte de Raspberry Pi. Pour accéder à Node-RED, vous pouvez donc utiliser « http://votre-nom-d-hote.local: 1880 »
Si ça ne fonctionne pas, vous pouvez simplement utiliser l'adresse IP de votre Raspberry Pi, par exemple http://192.168.0.10:1880, en remplacement de 192.168.0.10 par l'adresse IP de votre Pi (que vous pouvez obtenir en exécutant « hostname –I » en ligne de commande sur le Raspberry Pi).
Avec Node-RED, vous pouvez connecter facilement les périphériques du Rasberry Pi, tels que la caméra, le micro, le speaker, la LED et des APIs comme les services Watson. Il vous suffit de glisser/déposer des nœuds depuis la boîte à outil sur votre espace de travail, de paramétrer les nœuds puis de les connecter les uns avec les autres. Faire un « Deploy » chaque fois que vous voulez déployer votre code. En accédant à votre application, vous découvrez la « boîte à outils » qui contient notamment les bibliothèques de nœuds que vous venez d'installer : 21
Pour utiliser des nœuds de service IBM Cloud, vous devez récupérer les informations d'identification du service sur votre tableau de

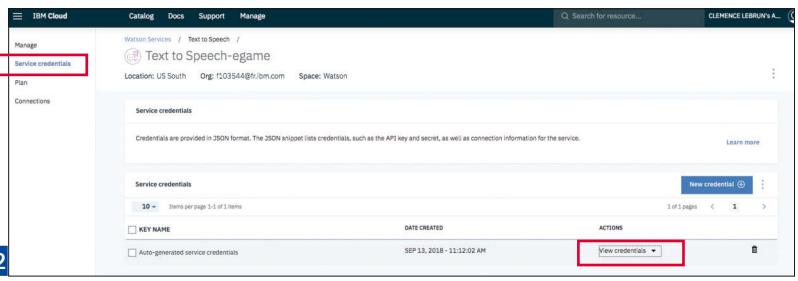
bord IBM Cloud, en sélectionnant le service : 22
Double cliquez sur le nœud concerné et fournissez le username/password ou l'API Key nécessaire à l'identification : 23

Scénarios

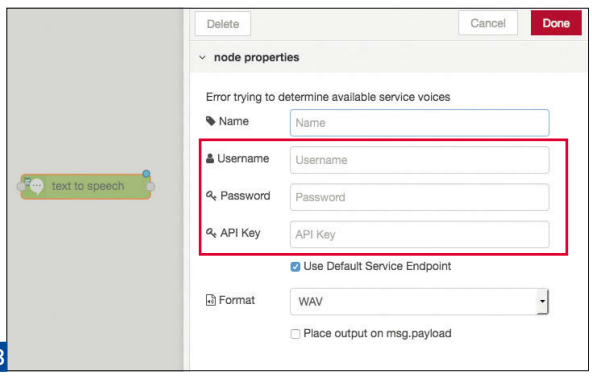
Voici quelques exemples de scénarios que vous pouvez créer avec Node-RED pour expérimenter les fonctionnalités de votre TJBot :

- Analyse de tons sur des tweets (fait clignoter les couleurs de la LED et mouvement du bras du TJBot en fonction de la classification du tweet) ;
- Contrôle de la couleur de la LED avec la voix (Speech To Text) et mouvement du bras du TJBot ;
- Bulletin météo en interrogeant des APIs de prévision météo (Weather Company Data sur IBM Cloud par exemple) et annonce par le TJBot (Text To Speech) ;
- Classification de photos prises avec la caméra Pi du TJBot utilisant Watson Visual Recognition et extraction de texte ;
- Conversation utilisant Watson Assistant pour dialoguer avec le TJBot. Notez que les nœuds node-red-dashboard peuvent être utilisés pour créer une interface utilisateur pour votre TJBot.


21



22



23

Exemples

Ce flow écoute un flux Twitter, appelle Tone Analyzer, lit les tweets avec une synthétisation vocale, fait clignoter les différentes couleurs de la LED en fonction des tons identifiés, et agite les bras du TJBot sur des tweets identifiés comme joyeux : 24

TJBot peut reconnaître des objets dans des images, détecter des visages dans une photo et estimer l'âge et le sexe d'une personne à l'aide de la reconnaissance visuelle et de l'extraction de texte de Watson : 25

Retrouvez des tutoriaux pour démarrer avec TJBot et Node-RED : https://github.com/johnwalicki/TJBot-Node-RED

Vous pouvez aussi récupérer les flow Node-RED et les importer directement dans votre application Node-RED sous forme de fichier JSON. Notez que si vous ne souhaitez pas utiliser Node-RED, il existe aussi d'autre types de tutoriaux : https://github.com/ibmtjbot/tjbot/tree/master/recipes .

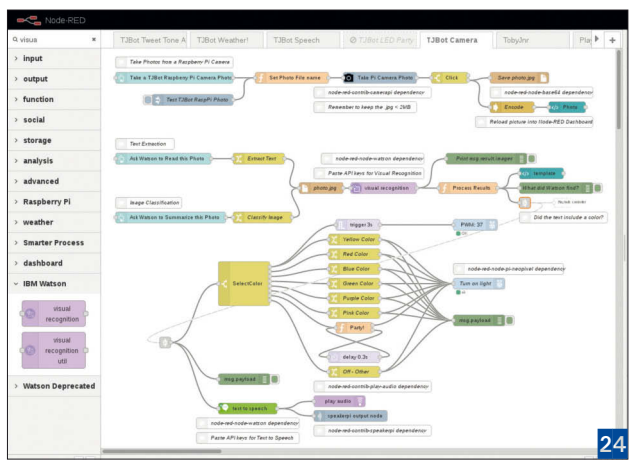
Vous avez maintenant toutes les clés en main pour créer vos propres scénarios et vous amuser avec votre TJBot !

Contribuez au projet

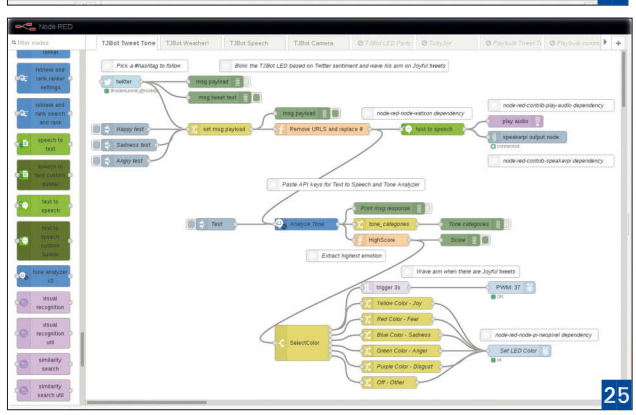
TJBot est open source donc si vous avez créé votre propre scénario, n'hésitez pas à le partager avec la communauté ici : https://github.com/ibmtjbot/tjbot/tree/master/featured .

Références :

Créer votre compte gratuitement : https://console.bluemix.net/registration/
https://ibmtjbot.github.io/
https://www.research.ibm.com/tjbot/
https://github.com/binnes/tobyjnr/wiki



24


25

Nicolas Decoster
@nnodot
Informaticien et scientifique chez Magellium
Co-fondateur et animateur à la Compagnie du Code

Juliane Blier
@tactless7
Développeuse JavaScript chez SchoolMouv

Tour d'horizon de Vue

Partie 2

Vue est un framework web créé pour la construction d'interfaces utilisateur. Il a été conçu pour une prise en main facile et une adoption progressive. Ce positionnement et des choix techniques bien pensés en font un framework de qualité, gagnant en popularité et se distinguant des autres frameworks majeurs que sont Angular et React.

niveau 200

Les Composants avec Vue

Vue permet de définir de nouveaux composants utilisables de la même manière que les éléments HTML standards. Par exemple, on peut définir un nouvel élément <github-contributors> contenant le tableau des collaborateurs présenté dans le n°223 et en spécifiant le nom du repository avec l'attribut repo :

```
<github-contributors repo="vuejs/vue"></github-contributors>
```

La définition d'un composant peut se faire de plusieurs manières. Par exemple on peut tout simplement utiliser l'instruction :

```
Vue.component('github-contributors', { /* data, etc. */ })
```

On peut également regrouper le template, le style et la logique dans un seul fichier dont l'extension est .vue. Dans le cas de notre composant GitHub, voici à quoi pourrait ressembler ce fichier (nous n'avons gardé que l'essentiel permettant de comprendre le mécanisme de définition d'un composant dans un fichier .vue) :

```
<template>
<div>
 <h1> {{ project.name }} </h1>
 <h4> {{ project.description }} </h4>
 ...
</div>
</template>

<script>
module.exports = {
 props: ['repo'],
 data: () => ({
  project: {}, ...
 }),
 async created () {
  // fetch data for this.project etc.
 },
 computed: { ... },
 watch: { ... }
}
</script>

<style scoped>
 .coredev { ... }
 .selected { ... }
</style>
```

Et voici un exemple d'interface utilisant ce composant pour afficher trois tableaux de contributeurs pour trois projets GitHub différents. La partie HTML :

```
<div id="my-app">
 <github-contributors
  v-for="repo in repos"
  v-bind:repo="repo" >
 </github-contributors>
</div>
```

et la partie JavaScript :

```
import GithubContributors from './GithubContributors.vue'
new Vue({
 el: '#my-app',
 data: function () {return {
  repos: [
   'vuejs/vue',
   'nuxt/nuxt.js',
   'VueVixens/website' ]}},
 components: {
  'github-colaborators': GithubContributors
 }
})
```

Les outils de build de Vue génèrent à partir du fichier .vue tout ce qui est nécessaire pour la définition et l'utilisation du composant.

Utiliser le state avec Vuex

Comme vu précédemment, Vuex est un outil de gestion d'état développé par l'équipe de Vue. Il fait partie de ces fameuses briques à ajouter au framework pour pousser un peu plus les

MIGRATION D'UN PROJET EXISTANT

La philosophie de progressivité de Vue se retrouve également dans la capacité de migration de projets existants vers l'utilisation de Vue. Il est possible de commencer à introduire Vue sur certaines parties isolées d'une interface pour l'étendre progressivement à toute l'application. Vue peut coexister avec les codes et les frameworks historiques d'un projet, et les différents outils de l'univers Vue (Vuex…) peuvent être introduits progressivement. Les difficultés résideront bien sûr dans les adhérences éventuelles du projet aux autres technos que doit remplacer Vue.

possibilités. Très utilisé dans les SPA (Single Page Application), le state permet de modifier les données affichées sur la vue, sans recharger la page. On conserve ainsi l'état de l'application. Il est éphémère dans le sens où si on la recharge, il disparaît et doit être reconstruit soit par des appels à l'API, soit par des actions utilisateurs.
Reprenons où nous en étions avec notre découpage en composants. L'utilisation de Vuex dans une vue se fait de cette manière :

```
const store = new VuexStore({
 state: {
  projects: {}
 }
})
new Vue({
 el: '#my-app',
 store, ...
})
```

L'idée ici est de "sauvegarder" la donnée récupérée par les appels d'API dans le state. De cette manière, nous pourrons utiliser cette donnée dans les différents composants. Et en cas de donnée différente commitée dans le state (nous verrons plus loin ce que peut bien être un commit dans le state), Vue utilisera ses propriétés de rendu réactif pour mettre à jour les composants.

Dans notre store, nous créons un objet projects dans lequel nous ajouterons chaque projet récupéré via l'appel API GitHub : nous ferons le commit d'une mutation dans le state. La mutation est la seule manière de modifier et mettre à jour le state. Voyons comment procéder :

CAS D'USAGE

Encore une fois, l'aspect progressif de Vue est un atout en termes de variété de cas d'usage. Le framework peut bénéficier à des projets d'ampleur et d'ambition variées. Vue est parfait pour le développement de petits prototypes ou de petits outils : un éditeur, la création d'un fichier HTML, l'insertion de Vue depuis un CDN et on peut coder rapidement et de manière élégante une petite application. L'outil pourra être utilisé en local ou être déployé sur un serveur statique. En introduisant progressivement les outils de l'univers Vue, on peut passer à des projets de plus en plus complexes couvrant tout le spectre des développements avec des technos web : Single Page Application, site statique avec contenu dynamique côté client (i.e. avec Vue tournant dans le navigateur), site statique généré en amont sans dynamique côté client (Server Side Rendering), site vitrine, application web complexe (GitLab utilise Vue), blog, site de commerce, site et application mobile, etc.

```
const store = new VuexStore({
 ...
 mutations: {
  addProject(state, project) {
   state.projects[project.name] = project;
  }
 }
})
```

L'utilisation de cette mutation se fait donc en "commitant", avec le payload retourné par l'appel à l'API GitHub dans notre cas :

```
fetch(`https://api.github.com/repos/${this.repo}`)
.then(project => {
 store.commit('addProject', project)
})
```

Ce faisant, chaque fois que ce morceau de code est appelé, l'objet récupéré est commité dans le store, alimentant ainsi le state. Il nous faut dorénavant indiquer à nos composants d'utiliser les informations provenant du state. Il y a deux solutions. Pour une

PERFORMANCES

Il est délicat de parler performances tant les benchmarks sont divers et que la moindre condition peut changer significativement le résultat final.

La documentation de Vue apporte un bon nombre d'informations sur les performances. Elle compare d'ailleurs celles de Vue à celles de React, framework reconnu pour ses bonnes performances. La manière dont sont rendus les composants diffère cependant de son concurrent. En effet, Vue garde un mapping précis de son arbre de composants. Cette spécificité lui permet de nativement rafraîchir uniquement les composants concernés par le changement, évitant ainsi aux développeurs d'optimiser le rendering.
Face à AngularJS, Vue semble s'en sortir assez facilement gagnant à nouveau grâce à son traçage des composants et de ses dépendances. AngularJS, lui, nécessite que tous les observateurs soient réévalués en cas de changement dans son scope, ce qui altère les temps de réaction lorsqu'il y en a beaucoup. Mais face à la nouvelle mouture d'AngularJS, Angular, les performances sont très similaires.

Pour des métriques un peu plus objectives, on peut regarder du côté du benchmark très complet mené sur un grand nombre de framework JS par le créateur du repo Github *js-framework-benchmark*. Vue y a de très bonnes performances, assez proches des deux autres frameworks les plus utilisés. La différence entre React, Angular et Vue est très faible et ne devrait donc pas forcément être un critère de décision à part entière mais plutôt un bonus. D'autre part, en comparant les poids des fichiers Gzip, Vue s'en sort largement devant avec seulement 23K contre 43K pour React et 143K pour Angular.

VUE, UN FRAMEWORK PROGRESSIF

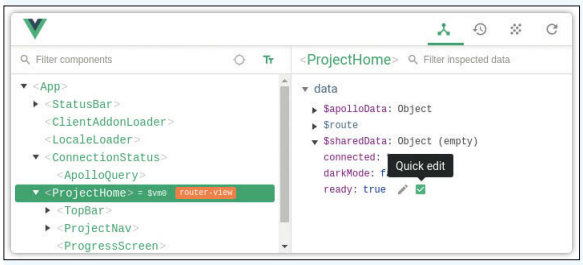
Pourquoi dit-on de Vue qu'il est un framework progressif ? Comme vu plus haut, l'écosystème de Vue est riche d'outils pouvant s'ajouter à la bibliothèque de base suivant les besoins. Chacun d'eux permet d'aller un peu plus loin dans l'utilisation du framework.

Vue Router

En plus de la gestion de state avec Vuex (cf. partie dédiée), les SPA (Single Page Application) développées en Vue utilisent bien souvent le Vue Router. Ce package permet de gérer les URL et d'adapter les composants ou les pages à afficher selon l'URL tapée. Il donne ainsi l'impression de naviguer sur l'application web en modifiant l'URL, mais sans recharger la page et donc en conservant le state.

Vue Devtools

L'équipe de Vue a pensé à tout, même au débogage ! Elle a en effet développé un plugin de navigateur (disponible sur Firefox et Chrome). Dans Vue Devtools, on peut visualiser l'état de chaque composant (cf. la figure ci-dessous). Il est aussi possible d'accéder en temps réel au state de l'application (cf. partie sur Vuex) et on a même accès à l'historique des mutations le modifiant au cours du temps. On peut ainsi revenir exactement avant cette fichue mutation qui a altéré nos données !



A gauche, l'arbre de composants de l'application. A droite, les données du composant sélectionné

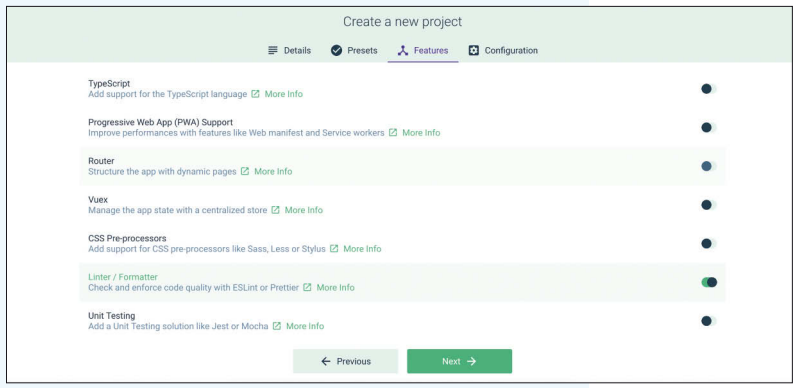
Vue CLI

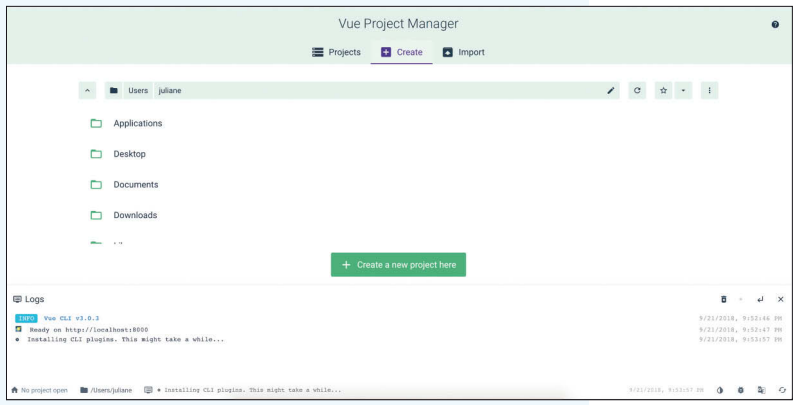
Avec Vue CLI, on peut démarrer très rapidement un projet en Vue presque totalement personnalisé. Avec la version 3 de Vue CLI et en une ligne de commande, l'outil nous propose linters, outils de tests, préprocesseurs, paquets tels que Vuex, Vue Router, etc. Ajouté à l'architecture déjà prête et la configuration webpack qu'il est possible de surcharger, vous avez tout ce qu'il faut pour démarrer un projet en Vue. La nouveauté de cette version 3.0 réside aussi dans la possibilité d'upgrader le projet vers la nouvelle version de l'outil n'importe quand, et en une commande.

```
Vue CLI v3.0.3
? Please pick a preset: Manually select features
? Check the features needed for your project:
 ◉ Babel
 ◯ TypeScript
 ◯ Progressive Web App (PWA) Support
 ◉ Router
❯◉ Vuex
 ◯ CSS Pre-processors
 ◉ Linter / Formatter
 ◯ Unit Testing
 ◯ E2E Testing
```

Les nombreuses possibilités de personnalisation du projet grâce à Vue CLI

En outre, l'équipe de développeurs a aussi embarqué une interface sobre mais efficace permettant de faire les réglages de la création d'un projet. Tout est là, la possibilité d'importer ses projets pour les gérer, celle d'en créer, la configuration… Et ce, associé à un système de logs en bas de page.





Vue ui

Le SSR avec Vue

Le SSR (Server-Side Rendering) a un intérêt en termes de performance (temps d'accès au site réduits) et SEO (apparition dans les moteurs de recherche). Vue propose des solutions pour pratiquer le SSR. Tout d'abord, lorsque l'intérêt est purement SEO et que le nombre de pages à indexer dans les moteurs de recherche est moindre, il est possible de s'orienter vers un simple plugin webpack (prerender-spa-plugin) qui construit les pages concernées et les envoie aux robots parcourant la page.

Il y a ensuite Nuxt.js, un framework complet basé sur Vue qui a été créé par les français Sébastien et Alexandre Chopin. Cet outil fait abstraction de la relation client-serveur et tout est pensé pour se concentrer sur l'UI et faire fi de la configuration.

Enfin, Vue propose de la documentation expliquant la création à la main d'un projet en SSR, avec l'intégration d'un serveur Node.js. C'est un peu plus laborieux mais aussi plus adaptable aux spécificités du projet. Cependant, un plugin permettant la création d'une architecture adaptée au SSR est en cours de création par Guillaume Chau, un membre de la core team Vue. Bien que ce package vue-cli-plugin-ssr n'en soit qu'à ses début, il reste très prometteur.

application utilisant beaucoup de variables du state, on privilégiera le helper mapState qui permet de mapper une variable du state à une variable de composant. Pour des projets plus petits, nous pouvons nous contenter d'utiliser une propriété computed qui récupèrera l'information du store de cette manière :

```
computed: {
 projects: () => {
  return this.$store.state.projects
 }
}
```

Ainsi, si la description du repository change, le prochain appel à l'API Github mettra automatiquement à jour les informations dans le state via le commit de la mutation 'addProject', et ce changement sera appliqué et affiché de manière réactive à tous les composants utilisant cette information.
Pour finir, on peut choisir de placer l'appel à l'API dans la définition du store pour en faire une action. Les actions peuvent être asynchrones, alors que les mutations sont toujours synchrones. Les actions ne modifient pas le state, mais commitent une mutation qui va, elle, modifier le state. Pour cela, on ajoute juste un champ action à la définition du store :

```
actions: {
 getProject({commit}, repo) {
  fetch(`https://api.github.com/repos/${repo}`).then(project => {
   store.commit('addProject', project)
  })
 }
}
```

Enfin, pour connecter tout cela à notre composant (cf. la partie sur les composants), il nous suffit d'appeler l'action getProject dans notre fonction created pour qu'à la création du composant les informations sur le projet soient récupérées :

```
async created () {
 store.dispatch('getRepo', this.repo)
}
```

Pour finir, signalons que Vuex propose une notion de module permettant de diviser le state lorsqu'on travaille sur de gros projets. Il est possible de créer un fichier par module et ainsi de segmenter le store de notre application pour plus de lisibilité et de maintenabilité. •

PRISE EN MAIN

Voyons concrètement comment on développe un projet avec Vue, en partant d'un environnement vierge (enfin, on suppose quand même que Node.js et un éditeur de texte sont déjà installés) jusqu'au déploiement d'un site statique. Pour le développement, le plus simple est d'utiliser le CLI (même si on peut tout à fait utiliser Vue et le configurer sans le CLI) qui s'installe avec la commande :

```
$ npm install -g @vue/cli
```

Tout d'abord le CLI offre la possibilité de tester des choses rapidement. Cela permet de prototyper ou de découvrir Vue sans devoir créer un projet node à part entière. Supposons que l'on crée un fichier App.vue qui contient :

```
<template>
 <h1>Hello!</h1>
</template>
```

Pour voir le résultat, il suffit de lancer la commande :

```
$ vue serve
```

et d'ouvrir l'adresse proposée (a priori http://localhost:8080/) avec un navigateur. Toute modification du fichier App.vue sera prise en compte immédiatement et l'affichage dans le navigateur sera mis à jour. Pour générer un site statique à partir de ce fichier, on utilise la commande :

```
$ vue build
```

Cela crée un dossier dist contenant tous les fichiers nécessaires pour le site. Cela fonctionne très bien pour des petits projets, mais dès que l'on veut construire quelque chose de plus conséquent en gérant proprement les dépendances, il est préférable de passer à la création d'un projet Vue. Pour cela on peut encore une fois utiliser le CLI :

```
$ vue create hello
```

Et se laisser guider par les questions qui sont posées. Cette commande crée un dossier avec tout ce qui est nécessaire pour le développement de notre projet Vue. Une fois que l'on a créé nos fichiers sources, on peut voir le résultat avec la commande :

```
$ npm run serve
```

Qui se comporte comme la commande vue serve ci-dessus. De même, il y a une commande pour créer les fichiers statiques du projet pour un usage en dehors du contexte de développement :

```
$ npm run build
```

Pour déployer le résultat il n'y a plus qu'à déplacer le contenu du dossier dist sur le serveur qui va gérer l'accès au site.

Par exemple, voyons ce qu'il faut faire pour déployer un projet Vue en tant "GitLab pages" à l'aide de GitLab CI. Pour cela il suffit de créer à la racine du projet GitLab un fichier .gitlab-ci.yml avec ce contenu :

```
pages: # Le nom du job doit être "pages"
 image: node:latest
 stage: deploy
 script:
  - npm run build
  - rm -r public  # GitLab Pages déploie le contenu d'un dossier
  - mv dist public # dont le nom est obligatoirement "public"
 artifacts:
  paths:
   - public
 only:
  - master
```

Le site sera déployé en tant que GitLab pages à chaque push dans la branche master du repository.
La documentation officielle de Vue CLI contient un section déploiement qui donne des exemples de déploiement détaillés pour divers hébergeurs classiques : GitLab Pages, comme dans notre exemple, mais aussi GitHub Pages, Netlify, Amazon S3, Firebase, Now, Surge et Bitbucket Cloud.

COMPARAISON AVEC REACT ET ANGULAR

La documentation officielle de Vue comporte une section très détaillée et bien faite qui compare Vue à d'autres solutions dont Angular et React. C'est un exercice toujours délicat et l'équipe de Vue a essayé de rester la plus objective possible. Citons également l'excellent article "Angular vs. React vs. Vue : A 2017 comparison" par Jens Neuhaus (@jensneuhaus), qui, même s'il date un peu, aborde la question de manière posée et pragmatique, et permet de se poser les bonnes questions pour choisir.

Les avantages de Vue les plus marquants sont que ce framework est très bien documenté, que la courbe d'apprentissage est douce et que l'on peut intégrer les différentes fonctionnalités de manière progressive. Son défaut majeur est que la communauté est beaucoup plus petite que celles de React et Angular. Une autre différence, qui peut être vue soit comme un avantage, soit comme un inconvénient, est que Vue est porté par la communauté et non par un acteur majeur du web (Facebook pour React et Google pour Angular). C'est un avantage en termes d'indépendance et de pérennité (il ne risque pas d'être abandonné par décision stratégique d'un acteur privé), et un inconvénient en termes de moyens mobilisés pour son développement.

Voyons quelques éléments de comparaison entre ces trois frameworks du point de vue du développeur.

Au niveau du rendu, chacun offre une solution technique spécifique : Angular et Vue utilisent une syntaxe de template intégrée à HTML et React se base sur JSX qui est un langage pour spécifier du code HTML qui s'intègre à JavaScript. Même si chaque framework recommande sa technique de rendu, il est toujours possible d'utiliser une méthode alternative. Soit en fournissant son propre moteur de rendu soit en en utilisant un existant. Il est ainsi tout à fait possible d'utiliser JSX avec Vue.

Pour donner une petite idée de certaines similitudes et différences entre les syntaxes de rendu de ces frameworks, voici une implémentation basique d'une liste de todos. D'abord en React :

```
<ul> {
props.todos.map((todo, index) => (
 <li key={index}>
  <div>
   <label> { todo.text } </label>
   <button onClick="{() => this.remove(todo)}">remove</button>
  </div>
 </li>
))
} </ul>
```

puis en Angular :

```
<ul>
 <li *ngFor="let todo of todos">
  <div>
   <label> {{ todo.text }} </label>
   <button (click)="remove(todo)">remove</button>
  </div>
 </li>
</ul>
```

et enfin en Vue :

```
<ul>
 <li v-for="(todo, index) in todos" v-bind:key="index">
  <div>
   <label> {{ todo.text }} </label>
   <button v-on:click="remove(todo)">remove</button>
  </div>
 </li>
</ul>
```

Une particularité d'Angular est qu'il est fortement recommandé d'utiliser TypeScript pour en profiter pleinement, alors que Vue et React s'utilisent indifféremment avec JavaScript ou TypeScript.

Une autre spécificité d'Angular est qu'il s'agit d'un très gros framework alors que React et Vue sont plutôt de petites bibliothèques. Angular offre beaucoup d'outils par défaut et propose une vision très spécifique sur la manière de coder et d'architecturer son application, en particulier avec une vision très orientée objet. React et Vue sont plus focalisés sur des fonctionnalités simples et offrent plus de souplesse.

En termes de gestion de l'état, les framework n'imposent rien mais certains font des recommandations. Pour React, un des outils classiques est Redux qui est maintenu en dehors du projet, pour Vue il s'agit de Vuex, qui est comparable à Redux mais qui va un peu plus loin, et qui est maintenu par la core team en cohérence avec le reste. L'architecture d'Angular se concentre sur d'autres aspects, il n'y a pas vraiment de recommandation et il est bien sûr possible d'utiliser des outils séparés ou dont l'intégration a été développée par la communauté. C'est le cas de ngrx qui bénéficie d'une certaine popularité et qui permet d'utiliser la bibliothèque réactive RxJS pour gérer l'état d'une application Angular.

Au niveau des outils de développement, ils bénéficient tous les trois de CLI proposant des fonctionnalités globalement comparables et s'intègrent tous plus ou moins finement aux DevTools des navigateurs.

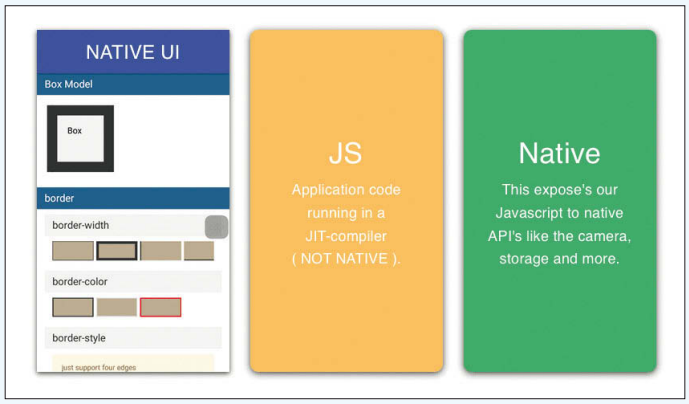
Au final, ce n'est pas toujours facile de choisir un framework. Pour vous aider, dans l'article cité plus haut, Jens Neuhaus (@jensneuhaus) vous propose une petite liste de recommandations en fonction de qui vous êtes, de vos contraintes ou de vos envies. Elle n'est peut-être pas complètement à jour (l'article date de 2017) mais nous trouvons qu'elle synthétise bien les différences entre les frameworks. Voici ce que Jens Neuhaus propose comme réponses à la question "Que dois-je choisir ?" :

Si vous travaillez chez Google : **Angular**
Si vous aimez TypeScript : **Angular (ou React)**
Si vous aimez la programmation orientée objet : **Angular**
Si vous avez besoin d'être guidé : **Angular**
Si vous travaillez chez Facebook : **React**
Si vous aimez la flexibilité : **React**
Si vous aimez les gros écosystèmes : **React**
Si vous aimez choisir parmi des douzaines de packages : **React**
Si vous adorez JavaScript et l'approche "tout en JavaScript" : **React**
Si vous aimez le code propre : **Vue**
Si vous voulez la courbe d'apprentissage la plus douce : **Vue**
Si vous voulez le framework le plus léger : **Vue**
Si vous voulez de la *separation of concerns* dans un seul fichier : **Vue**
Si vous travaillez seul ou dans une petite équipe : **Vue (ou React)**
Si la taille de votre application tend à être importante : **Angular (ou React)**
Si vous voulez construire une application avec react-native: **React**
Si vous voulez disposer d'un grand choix de développeurs : **Angular ou React**
Si vous travaillez avec des designers et avez besoin de fichiers HTML propres : **Angular ou Vue**
Si vous aimez Vue mais que vous avez peur de l'écosystème limité : **React**
Si vous n'arrivez pas à vous décider, apprenez d'abord **React**, puis **Vue** et enfin **Angular**

QUID DU MOBILE AVEC VUE.JS ?

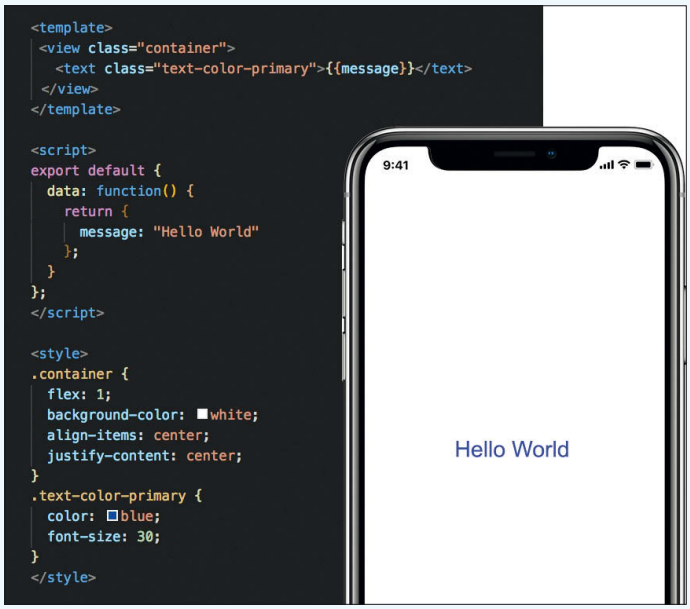
Ce n'était pas le cas il y a encore un an, mais il existe à ce jour plusieurs solutions viables pour développer une application mobile avec Vue.js.

Weex



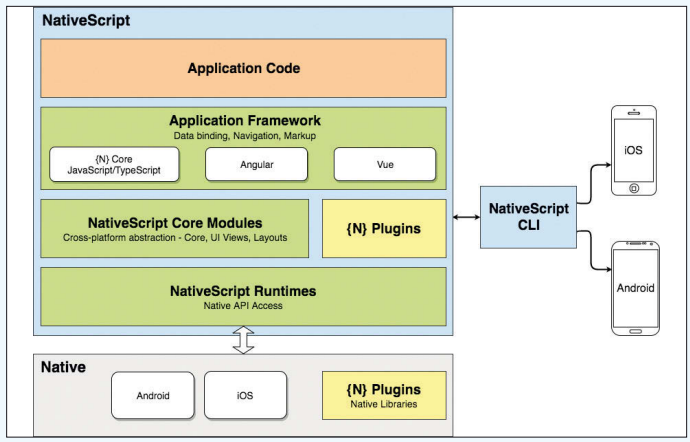
Projet porté par le géant chinois Alibaba, Weex est un framework permettant de développer des applications Android, iOS et même des applications web avec une seule base de code. Même si sa documentation évolue régulièrement, elle reste à l'heure actuelle peut être encore un peu trop succincte pour espérer pouvoir mettre en production une application développée avec Weex. D'autant plus que, Alibaba oblige, la grande majorité des ressources sont écrites en chinois.

Vue Native



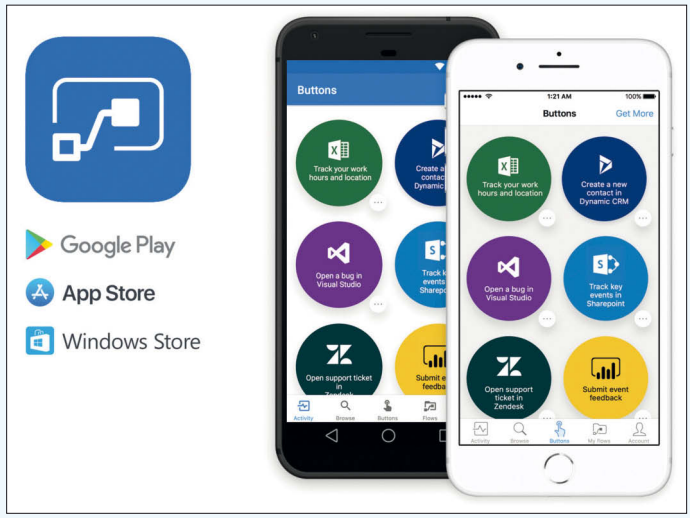
Pour ceux n'ayant pas eu l'occasion d'en entendre parler, React Native est un framework permettant de développer des applications Android et iOS natives en se basant sur React.js. Une équipe de développeurs a récemment créé un plugin permettant d'arriver aux mêmes objectifs mais en écrivant du Vue.js. Celui-ci compile le code en React Native qui lui-même sera compilé en code natif pour la plateforme visée. Les composants natifs sont les mêmes que pour React Native, seule la manière de coder diffère. On pourrait d'ailleurs penser que la double compilation laisse à désirer en termes de performance, mais étonnamment, les perfs sont très proches d'une app React Native.

NativeScript Vue



Dans la famille mobile en Vue.js je voudrais le petit dernier ! NativeScript Vue est arrivé début 2018 dans sa version 1.0. Depuis, la version 2.0 a eu le temps de voir le jour. Comme les deux autres, il permet de construire des applications natives avec une seule base de code et en utilisant des composants natifs. En revanche, celui-ci est un plugin de NativeScript, framework open-source permettant de développer des applications Android et iOS, et par conséquent, il a l'avantage de l'expérience de NativeScript et surtout un bon nombre de la librairie de composants inhérente à cette technologie. Il est donc tout à fait possible de développer une app mobile avec NativeScript Vue, même si le framework manque de quelques fonctionnalités bien pratiques telles que le hot reload (liveSync uniquement disponible lors du développement sur simulateur Android).

Ionic



Ionic est un framework Javascript longuement éprouvé depuis 2015 et qui en est actuellement à sa 4ème version. Basé sur Cordova, il encapsule une webview dans une application, qu'elle soit Android ou iOS. La manière de coder est très similaire au web et de nombreux plugins sont disponibles pour faciliter le développement. Ici encore, une seule codebase suffit, même si les plugins ne sont pas tous compatibles avec les deux plateformes. Auparavant cantonné au langage Angular, la nouvelle version 4 d'Ionic permet dorénavant l'utilisation de Vue dans sa codebase.
A l'heure où nous écrivons ces lignes, Ionic 4 n'est malheureusement pas encore sorti officiellement. Difficile donc de songer à l'utiliser en production.

Jérôme GIACOMINI
Consultant Infinite Square
http://blogs.infinitesquare.com/
@jeromegiacomini

INFINITE SQUARE

niveau 100

ASP.Net Core et la compression

Il peut être utile de supporter la compression des réponses lorsqu'elles sont volumineuses cela peut réduire considérablement le temps de téléchargement de celles-ci.

Activer la compression gzip

Pour activer la compression gzip c'est très simple le framework ASP.NET Core fournit des classes toutes faites.
Dans le fichier Startup il suffit d'activer la compression des réponses de la manière suivante :

```
using Microsoft.AspNetCore.ResponseCompression;
...

public ConfigureServices(IServiceCollection services)
{
   services.AddResponseCompression();
}
```

Il est à noter que vous pouvez aussi configurer le niveau de compression Gzip de la manière suivante :

```
services.Configure<GzipCompressionProviderOptions>(o => o.Level =
System.IO.Compression.CompressionLevel.Optimal);
```

Si votre site est en https, vous pouvez activer la compression de la manière suivante :

```
services.AddResponseCompression(o =>
{
   o.EnableForHttps = true;
});
```

Attention activer la compression avec le protocole HTTPS peut mener à des failles telles que BREACH et CRIME.

Puis dans la méthode Configure vous devez appeler la méthode d'extension suivante :

```
app.UseResponseCompression();
```

Activer la compression Brotli

ASP.NET Core ne fournit pas de *compression provider* pour ce format. Il faut donc l'écrire soi-même.
Compression provider Brotli :

```
public class BrotliCompressionProvider : ICompressionProvider
{
  private readonly CompressionLevel _compressionLevel;
  public BrotliCompressionProvider(CompressionLevel compressionLevel = CompressionLevel.Fastest)
  {
    _compressionLevel = compressionLevel;
  }

  public string EncodingName => "br";
  public bool SupportsFlush => true;
  public Stream CreateStream(Stream outputStream)
  {
    return new BrotliStream(outputStream, _compressionLevel);
  }
}
```

Activer la compression Deflate

Compression provider Deflate :

```
public class DeflateCompressionProvider : ICompressionProvider
{
  private readonly CompressionLevel _compressionLevel;
  public DeflateCompressionProvider(CompressionLevel compressionLevel = CompressionLevel.Fastest)
  {
    _compressionLevel = compressionLevel;
  }
  public string EncodingName => "deflate";
  public bool SupportsFlush => true;
  public Stream CreateStream(Stream outputStream)
  {
    return new DeflateStream(outputStream, _compressionLevel);
  }
}
```

Attention la classe DeflateStream ne respectent pas bien la norme Deflate donc si vous utilisez un client autre que .Net il ne pourra pas décoder votre archive.

Ajout des ICompressionProvider custom

Puis, il faut modifier la méthode AddResponseCompression de la manière suivante :

```
services.AddResponseCompression(o =>;
{
  o.Providers.Add(new BrotliCompressionProvider());
  o.Providers.Add(new DeflateCompressionProvider());
  o.EnableForHttps = true;
});
```

Récupérer une réponse compressée

Maintenant que nous avons une réponse compressée il n'y a qu'à l'utiliser.

Pour avoir un contenu compressé le client devra envoyer le header *Accept-Encoding* avec comme valeur *gzip*, ou *deflate*, ou *br*. Il peut aussi envoyer les 3 "*gzip,deflate,br*" pour dire au serveur qu'il supporte les 3 formats.

Mais les clients ne décompressent pas automatiquement les réponses :

- heureusement ma libraire Tiny.RestClient le fait **automatiquement** sans besoin de configurer quoi que ce soit.
- La classe *HttpClient* permet de décompresser automatiquement un contenu compressé.

Mais son utilisation n'est pas évidente car il faut lui passer un *handler* spécifique :

```
HttpClientHandler handler = new HttpClientHandler()
{
  AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate
};


using (var client = new HttpClient(handler))
{
  ....
}
```

Attention, pour l'instant la compression brotli n'est pas encore gérée par HttpClient.

Quand faut-il activer la compression ?

La documentation précise qu'il ne faut pas compresser des fichiers inférieurs à 1000 octets car cela peut produire un fichier compressé plus volumineux que le fichier non compressé.

Il est aussi à noter que la compression rajoute de la charge de calculs côté serveur.

Happy coding. •

Laurent Ellerbach
Principal Software Engineer Manager,
Microsoft
laurelle@microsoft.com,
http://github.com/ellerbach

Accès au matériel avec .NET Core ou le retour de DllImport

niveau 100

.NET Core est un framework multiplateforme, totalement Open Source, hérité de .NET. Le code source est disponible sur https://github.com/dotnet/core. La version actuelle est .NET Core 2.1. Les contributions communautaires sont volontiers acceptées. Il est donc possible de faire du code qui fonctionnera aussi bien sur un Raspberry Pi que dans un container Docker dans Azure, sur un Mac ou bien entendu sur Windows.

Parce que .NET Core est multiplateforme, il est difficile de supporter directement dans le framework des spécificités matérielles comme les ports USB car ceux-ci sont intimement liés à l'OS et à l'architecture des drivers sous-jacents. Le support matériel a donc un prix : il faudra dans la plupart des cas l'implémenter soi-même.

La bonne nouvelle c'est qu'il est bien sûr possible de le faire assez facilement. Dans cet article, je prendrai l'exemple du port série. Même si on peut se dire que c'est assez standard et maintenant connu depuis des décennies, il n'y a pas encore de support natif pour le port série dans .NET Core. Pour un de mes projets personnels, j'ai eu besoin du support du port série sous Linux. J'ai en effet un Raspberry Pi qui tourne dans mon jardin, plus exactement dans ma serre, qui est connecté par port série à un Arduino bardé de capteurs tels que température, humidité du sol, de l'air, vitesse, direction du vent.

Je voulais vraiment garder Linux sur le Raspberry et utiliser .NET Core. C'était avec la version 1.0 et une façon pour moi de tester dans des conditions réelles le nouveau framework sur une plateforme Arm. Les données récoltées sont postées dans Azure IoT Hub.

Le secret pour accéder au matériel et à n'importe quel composant de l'OS ou librairie existe depuis une éternité : DLLimport. C'est un héritage de Visual Basic 1.0. Oui, vous lisez bien, c'est bien aussi vieux que cela et surtout ça a toujours été présent dans tous les frameworks .NET. Y compris bien sûr à l'époque sur Mono qui permettait déjà de faire tourner du code du framework riche .NET sur d'autres plateformes telles que Linux et Mac.

Son utilisation est simple : il suffit de référencer une DLL, un SO ou toute autre librairie exportant des points d'entrées comme par exemple avec la libc sous Linux. Si vous souhaitez importer sous Windows une DLL, vous utiliserez un nom de type *malibrarie.dll*. Et si vous utilisez des librairies non-système sous Linux ou Mac, le nom ressemblera à *malibrairie.so*.

```
using System.Runtime.InteropServices;
namespace System.IO.Ports {
  public static class Libc {
    [Flags]
    public enum OpenFlags {
      O_RDONLY = 0,
      O_WRONLY = 1,
      O_RDWR = 2,
      O_NONBLOCK = 4, }
    [DllImport("libc")]
    public static extern int open(string pathname, OpenFlags flags);
    [DllImport("libc")]
    public static extern int read(int fd, IntPtr buf, int count);
}}
```

La première fonction *open* permet l'ouverture d'un port série. La seconde, *read*, permet de lire les données du port s'il y en a de disponible. Nous verrons un peu plus loin comment les utiliser.

Ce qu'il est nécessaire d'avoir c'est bien entendu la définition exacte de la fonction que vous voulez importer. Attention, il va y avoir du marshaling, conversion de type, et il est important de bien comprendre les mécanismes de conversion.

Dans le cas de la première, nous avons une énumération de type Falgs qui sera automatiquement convertie dans le bon type.

Les type *String* seront eux aussi convertis en char* dans ce cas. Le type *IntPtr* est un pointeur et sera aussi converti par le framework pour correspondre à un pointeur sous Linux. Il existe des mécanismes de conversion pour tous les types et il est possible de forcer certains types. Dans l'exercice d'importer des librairies, c'est la principale difficulté que de bien comprendre quel est le type attendu et comment le représenter en .NET Core.

Les 2 fonctions et l'enum ci-dessus ne sont qu'un sous ensemble des fonctions nécessaires pour accéder au port série. L'intégralité du code source de ce projet se trouve sur mon GitHub https://github.com/Ellerbach/serialapp.

Ensuite, l'utilisation des fonctions se fait de la même façon que pour n'importe quelle classe .NET :

```
int fd = Libc.open(portName, Libc.OpenFlags.O_RDWR | Libc.OpenFlags.O_NONBLOCK);
if (fd == -1)
{
  throw new Exception($"failed to open port ({portName})");
}
```

Ces quelques lignes permettent l'ouverture du port série. A noter que *portName* est le nom du port série. Sous Windows, il ressemblerait à COM3 mais sous Linux ou Mac, il portera plutôt le doux nom de /dev/ttyS2 ou /dev/ttyUSB0 par exemple.

La seconde difficulté est maintenant d'utiliser correctement les fonctions et dans le bon ordre. Pour cela, je conseille de regarder

comment elles sont utilisées sur la plateforme native. Des exemples d'accès au port série avec des codes en C++ m'ont permis de bien comprendre comment les utiliser. La documentation des drivers spécifiques doit aussi permettre de bien comprendre l'utilisation spécifique. En règles général, soyez curieux et cherchez dans GitHub ou StackOverflow l'utilisation spécifique des librairies qui vous intéressent.

Attention également aux droits qui peuvent être nécessaires à l'utilisation de librairies. Sous Linux par exemple, les ports séries nécessitent une élévation de privilèges.

L'ouverture du port série est un peu plus compliquée que le code ci-dessus car il faut aussi associer la bonne vitesse pour le port. Le code complet est disponible sur mon GitHub.

Ci-dessous un autre exemple pour lire les données :

```
int res = Libc.read(fd.Value, readingBuffer, READING_BUFFER_SIZE);
if (res != -1) {
  byte[] buf = new byte[res];
  Marshal.Copy(readingBuffer, buf, 0, res);
  OnDataReceived(buf); }
```

Cet exemple est intéressant car il permet de voir le mécanisme de marshaling en action. S'il y a des données disponibles sur le port série, il sera nécessaire de les stocker dans une variable *safe* de .NET, ici un tableau de byte *buf* qui a précédemment été créé. Comme vu dans la définition, nous récupérons un pointeur *readingBuffer*. Cette copie des données est possible grâce à la fonction *Marshal.Copy*. Vous vous doutez qu'il faudra faire l'exercice inverse pour l'écriture :

```
IntPtr ptr = Marshal.AllocHGlobal(buf.Length);
Marshal.Copy(buf, 0, ptr, buf.Length);
Libc.write(fd.Value, ptr, buf.Length);
Marshal.FreeHGlobal(ptr);
```

Dans ce cas de figure, il faut d'abord créer un pointeur, ce qui est permis par *Marshal.AllocHGlobal*. La copie des données s'effectue comme dans le cas précédent et il faut enfin libérer le pointeur une fois l'appel à la fonction effectué. C'est ce que permet de faire la fonction *Marshal.FreeHGlobal*. Ces mécanismes d'allocation et désallocation de mémoire avec des pointeurs sont très familiers aux développeurs C/C++ mais ne le sont en général pas du tout pour les développeurs C#/Java/Python et autres languages comprenant des mécanismes de *garbage collector* et effectuant ce travail en tâche de fond pour vous. Il est critique de bien effectuer cette allocation/désallocation sinon la mémoire de votre application peut très rapidement arriver à saturation. Et qui dit copie de données dit aussi lenteurs potentielles, donc attention à toutes les fonctions utilisant intensément les pointeurs et marshaling, elles peuvent rapidement engendrer quelques ralentissements. Si cependant vous avez beaucoup d'opérations de ce type à effectuer, vous pouvez le faire en mode *unsafe*. Ce mode vous permet un accès direct aux pointeurs, donc à la mémoire, permettant des transferts faciles de données dans le monde *unsafe* en .NET Core 2.1. La programmation de ce type est alors très proche de celle en C/C++. Plus d'informations sur https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/unsafe-code.

Toutes les fonctions permettant le marshaling se trouvent dans le namespace *System.Runtime.InteropServices*. La documentation se trouve sur https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.marshal?view=netcore-2.1.

L'utilisation de DLLimport permet donc d'accéder à tout ce qui peut être exposé par l'OS sous-jacent, quel qu'il soit. Il vous suffit de connaître les librairies et leur utilisation. Si vous souhaitez réaliser des composants multiplateforme, vous serez obligé d'importer des librairies très différentes suivant les OS mais aussi architectures. Et vous devrez tester également la plateforme sur laquelle vous tournez pour appeler le code spécifique à la plateforme. Dans cet exemple de support du port série, la classe est minimale et n'implémente pas toutes les fonctions que vous trouverez dans le namespace *System.IO.Ports*. Il est bien sûr possible de le faire, avoir une couverture complète des classes et fonctions. Mon besoin était simple et je suis allé à l'essentiel.

Au-delà du code, un Nuget est également disponible NetCoreSerial https://www.nuget.org/packages/NetCoreSerial. Bon DLLimport et profitez de tout ce que l'OS cible de votre application peut vous offrir ! •

Richard Clark
Microsoft MVP .NET depuis 2002, il est l'animateur du 1er site .NET français www.c2i.fr et coanime le podcast devapps.be.

Blazor : sauce WebAssembly

niveau 100

Je vais vous parler dans cet article d'un projet expérimental lancé par des membres de l'équipe ASP.NET de Microsoft nommé Blazor. A ma connaissance c'est la première fois que Microsoft aborde un projet de cette façon, c'est-à-dire un POC sur un nouveau Framework et disponible en Open Source. Un peu comme en 2007 quand ils ont travaillé sur les premières maquettes d'ASP .NET MVC, mais dans le cas présent, ils sont complètement transparents et nous livrent leurs idées au fur et à mesure du développement et nous incitent à y participer.

Alors ils le disent ouvertement : « on essaye, on va voir ce que cela donne, si c'est cool et que vous trouvez ça cool, on continuera officiellement, sinon on laissera tomber ». Donc pas question pour l'instant d'imaginer de mettre en production une quelconque application reposant sur ce Framework. Donc je vous aurais averti :

WARNING, WARNING : pour le fun uniquement pour l'instant.

WebAssembly

Tout d'abord, Blazor repose sur WebAssembly, un standard du W3C supporté par les principaux navigateurs. 1

Ce standard lancé en 2015 seulement, permet d'exécuter du byte-code dans le navigateur dans une sandbox. Donc au lieu d'exécuter du code Javascript, vous avez un code binaire qui théoriquement, s'exécutera bien plus rapidement tout en restant sécurisé. Tout comme en Java ou en .NET, vous avez une sorte de code intermédiaire (qui peut être décompilé au format texte) qui sera compilé à la volée par la machine virtuelle du compilateur. Si vous voulez en savoir plus sur ce format (et avaler des pages et des pages de spécifications techniques), sachez que les spécifications sont Open Source (https://webassembly.org/).

La bonne nouvelle (pour Microsoft), c'est que l'équipe Mono s'est penchée sur cette technologie et a crée un compilateur (mono-wasm) qui compile votre code IL en WebAssembly. En appliquant ce compilateur au .NET Core Framework, vous avez donc théoriquement tout le Framework .NET qui est disponible dans votre navigateur.

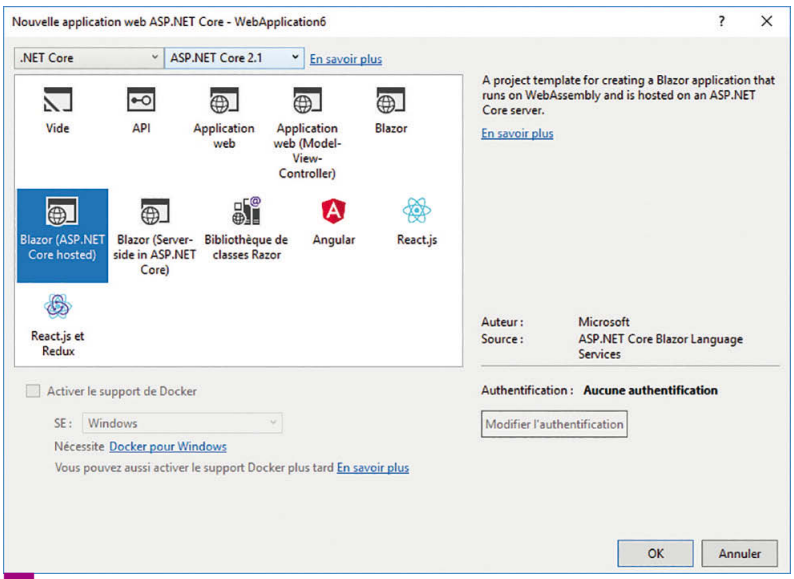
Qu'est-ce que Blazor ?

Maintenant, quelle est l'idée de Blazor ? Blazor est la contraction de Browser et Razor (Brazor aurait été plus logique mais bon…) donc repose sur Razor. Le moteur de Razor utilisé par ASP.NET MVC depuis de nombreuses années permet, à partir d'un code mélangeant du Html et du code C#, de créer des classes .NET qui généreront le code Html envoyé au navigateur.

Voilà donc l'astuce : à partir du code Razor, on génère une classe que l'on compile en code IL, le code IL est recompilé en WebAssembly par mono-wasm et le tout est envoyé au navigateur. Le code interprété par le navigateur génère le code Html affiché.

Simple non ? Presque. Dans la théorie cela semble séduisant et facile, mais il faut enrichir Razor pour parvenir au but final. Prenons l'exemple simple d'un appel Ajax dans votre code : il n'est pas question, pour des raisons de sécurité que ce soit votre WebAssembly qui se charge de cet appel, seul le navigateur doit en être responsable. Autre exemple : l'interaction entre votre code et le code Javascript qui peut être présent dans votre page ? Bref, il y a du pain sur la planche. Regardons dans un premier temps les principaux concepts de Blazor.

1



2

Concepts fondamentaux

Blazor est basé sur des composants (héritant de BlazorComponent) qui sont des fragments de l'interface. Une page est un composant, qui peut contenir des composants, etc. La décomposition d'une page en composants est importante car Blazor gère la mise à jour du DOM affiché au niveau du composant.

Comme une application Blazor est une application SPA (Single Page Application), il possède un moteur de routage, et, tout comme .NET Core, d'un moteur d'injection de dépendance.

Enfin, comme tout Framework SPA qui se respecte, il permet d'interagir avec du code Javascript (dans les deux sens).

Notre première application Blazor

Si vous voulez vous amuser avec ce Framework, il y a d'abord quelques impératifs :

- Visual Studio doit être en version 15.7 minimum ;
- Le .NET Core 2.1 SDK doit être installé ;
- Et via le VS Marketplace, vous devez avoir le Blazor Language Services extension.

Les quelques gigaoctets téléchargés et installés, dans Visual Studio, créez une application web .NET Core. Vous avez alors le choix : 2

Dans notre exemple nous allons choisir tout de suite le modèle Blazor (ASP .NET Core hosted) qui est le plus intéressant et on est des warriors chez Programmez !

Ce modèle crée une solution avec trois projets :

- Un projet ASP .NET MVC « classique » Server dans lequel on écrira nos apis REST ;
- Un projet librairie .NET classique nommée Shared ;
- Et un projet Blazor nommé client.

On voit ici l'un des gros points forts de Blazor : on peut avoir du code partagé entre notre serveur REST et l'application SPA cliente, le code d'une entité peut être utilisé (et donc non réécrite) dans les 2 projets.

Dans le modèle on a une entité WeatherForcast dans le projet Shared :

```
public class WeatherForecast
{
  public DateTime Date { get; set; }
  public int TemperatureC { get; set; }
  public string Summary { get; set; }
  public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```

Cette entité est utilisée par notre api REST dans l'application serveur :

```
[HttpGet("[action]")]
public IEnumerable<WeatherForecast> WeatherForecasts()
{
  // le code n'est pas important
  ...
}
```

Et côté application SPA :

```
WeatherForecast[] forecasts;

protected override async Task OnInitAsync()
{
  forecasts = await Http.GetJsonAsync<WeatherForecast[]>("api/SampleData/WeatherForecasts");
}
```

Si vous êtes un développeur Full Stack, c'est un gros avantage, un gain de temps indéniable.

Anatomie d'une page

Regardons maintenant comment est constituée une page Blazor (extension cshtml car c'est une page Razor). C'est une version un peu modifiée du modèle de base :

```
@using WebApplication5.Shared
@page "/fetchdata"
@inject HttpClient Http

<h1>Weather forecast</h1>

@if (forecasts == null)
{
  <p><em>Cliquez sur Refresh</em></p>
}
else
{
  <table class="table">
    <tbody>
      @foreach (var forecast in forecasts)
      {
        <tr>
          <td>@forecast.Date.ToShortDateString()</td>
          <td>@forecast.TemperatureC</td>
          <td>@forecast.TemperatureF</td>
          <td>@forecast.Summary</td>
        </tr>
      }
    </tbody>
```

```
  </table>
}
<button class="btn btn-primary" onclick="@RefreshAsync">Refresh</button>

<div><Copyright Title="c2i.fr"></Copyright></div>


@functions {
  WeatherForecast[] forecasts;

  private async void RefreshAsync()
  {
    forecasts = await Http.GetJsonAsync<WeatherForecast[]>("api/SampleData/WeatherForecasts");
  }
}
```

La directive @page "/fetchdata" indique que l'on a un composant de type page accessible via l'url /fetchdata. (http://localhost:2345/fetch-data).
La directive @inject HttpClient http indique que l'on veut faire une injection de dépendance pour le service HttpClient.
Puis l'on a une page Razor « classique » avec les directives @if, @foreach, @functions. Et c'est dans @functions que l'on va définir le code behind de notre page. Ce code contient juste une méthode asynchrone qui utilise le service HttpClient pour appeler l'api REST de notre serveur. Cette méthode est appelée quand on clique sur le bouton de notre page (directive onclick="@RefreshAsync").
Enfin, on a un nouveau tag Html, Copyright, qui en réalité est un autre composant Blazor, donc la définition ne contient que quelques lignes (fichier nommé Copyright.cshtml) :

```
<div>@Title</div>

@functions {
  [Parameter]
  string Title { get; set; }
}
```

En tant que développeur C#, il n'y a rien de bien sorcier et l'on est tout de suite à l'aise. C'est l'avantage de Blazor.

Quelques remarques

Interop Javascript

Blazor permet l'interaction avec du code Javascript et dans les deux sens, c'est-à-dire qu'un code C# peut appeler une méthode Javascript et une méthode Javascript peut appeler une méthode (d'instance depuis la v0.6) de votre code C#. Pour qu'un code C# puisse appeler un code Javascript, il faut pour cela enregistrer dans un dictionnaire cette méthode (je n'entrerai pas dans le détail ici). Et c'est ce qu'il fait quand il doit faire un appel http.
Je vous l'avais dit plus haut, pour des raisons de sécurité, vous n'avez pas le droit d'utiliser les classes HttpClient du Framework

.NET. Dans Blazor, on a donc des méthodes Javascript qui font cet appel et une classe HttpClient qui se charge d'invoquer ces méthodes Javascript. C'est pourquoi on utilise l'injection de dépendance dans cet exemple pour bien utiliser le HttpClient de Blazor et non celui de .NET.

Binding

Le binding entre le code Html et les variables C# est dans l'exemple unidirectionnel (quand on met à jour la variable C#, l'affichage se met à jour). Il y a bien entendu la possibilité de le faire en bidirectionnel avec les tags input.

Débogage

C'est le gros point noir pour l'instant de Blazor : le débogage est très limité mais je vous rassure, ils travaillent dessus. Pour l'instant, l'astuce que j'ai trouvée est de créer un code Javascript qui fait du console.log et que j'invoque grâce à l'interop.

Performance

L'un des gros points avancés comme avantage de Blazor ce sont les performances. Théoriquement, il est vrai qu'une WebAssembly a tout pour séduire mais pour l'instant, on est encore dans le flou le plus complet sur les réelles performances de WebAssembly et de Blazor en particulier.

Conclusion

Blazor est un projet expérimental intéressant. Dire qu'il est l'avenir, je ne saurais trop le dire. Il est indéniablement intéressant pour un développeur full stack C# mais quand on voit la simplicité et la puissance des Framework Javascript/Typescript comme Angular, il lui reste beaucoup de travail à faire, surtout côté outillage.

Si vous voulez en savoir plus, rendez-vous donc sur la page du projet Blazor (https://blazor.net/) et/ou sur son projet Github (https://github.com/aspnet/Blazor) ou alors, sur l'émission de notre podcast devApps.be consacré à cette technologie : http://devapps.be/podcast/blazor-webassembly/ (on a un logo pour Blazor qui a quand même plus de gueule que le logo officiel) ou sur mon site web où j'ai consacré une série d'articles (http://www.c2i.fr/articles/commencons-avec-les-webassembly-et-blazor). •

Denis Duplan,
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Zoom hardware sur Amiga

Partie 1

Le choc de l'arrivée de la Super Nintendo fin 1991 pour les amateurs de micros 16 bits, ce fut le Mode 7. La console était capable d'appliquer une rotation et un redimensionnement à un bitmap de grande taille dans la trame, et il était possible de trafiquer pour produire un effet de perspective.

niveau 300

L'Amiga 500 était doublement handicapé pour parvenir à produire de tels effets : pas de circuit spécialisé pour cela, et une organisation des données affichées sous forme de bitplanes exigeant, pour y parvenir, de nombreuses opérations au CPU et/ou au Blitter. Toutefois, cela n'empêcha pas certains de produire des effets du genre, par exemple le zoom au démarrage de la démo *World of Commodore 92* de Sanity (Figure 1).

Parmi tous ces effets de zoom avec ou sans perspective, la plupart se sont appuyés sur le zoom vertical hardware, découvert dès les débuts de l'Amiga, et certains sur le zoom horizontal hardware, qui l'a été bien plus tard.

Comment procéder à l'un et l'autre de ces zooms avec le hardware ? En particulier, quel est ce fameux « *$102 trick* » régulièrement évoqué dans les forums de demomakers, souvent succinctement, et parfois abusivement ? Et dans quelle mesure est-il possible de zoomer ainsi ? Tout cela et plus encore dans ce qui suit.

Vous pouvez télécharger l'archive contenant le code et les données des programmes présentés ici à l'URL suivante :

http://www.stashofcode.fr/code/zoom-hardware-amiga/code.zip

Cette archive contient plusieurs sources :

- `zoom0.s` pour dissimuler par zoom vertical hardware des lignes avant, au milieu et en bas d'une image et recentrer cette dernière verticalement à l'écran ;
- `zoom1.s` pour identifier quand utiliser le Copper pour modifier la valeur de BPLCON1 afin de dissimuler un certain nombre de colonnes formées des derniers pixels d'un groupe de 16 pixels ;
- `zoom2.s` pour visualiser le résultat produit par le zoom horizontal hardware reposant sur la technique précédente généralisée pour dissimuler de 1 à 15 colonnes de pixels d'une image ;
- `zoom3.s` pour tester un zoom reposant sur la technique précédente pour réduire une image de 306 à 15 pixels de largeur ;
- `zoom4.s` pour tester un zoom combinant zoom horizontal hardware et zoom vertical hardware pour réduire une image de 306 x 256 pixels à 15 x 15 pixels.

Comme toujours depuis qu'il en est question dans ces colonnes, vous pouvez pratiquer la programmation sur Amiga à l'aide de l'assembleur ASM-One tournant sur une émulation d'Amiga 500 dans WinUAE, en vous appuyant au besoin sur l'*Amiga Hardware Reference Manual*. Vous trouverez les références à la fin de cet article. Référez-vous au premier article de la série portant sur la programmation d'un sine scroll (*Programmez !* #213 à #217), pour plus d'informations. Alternativement, vous pouvez vous reporter au blog de l'auteur.

NB : Cet article se lit mieux en écoutant l'excellent module *Helmet for sale* composé par Jason / Kefrens pour *R.A.W.* #2, mais c'est affaire de goût personnel…

1

Zoom dans la démo World of Commodore 92 de Sanity.

Le zoom vertical hardware, un usage banal des modulos

Comme cela a été expliqué dans un article précédent, le hardware affiche une image composée de bitplanes dont les adresses lui sont fournies via les couples de registres BPLxPTH / BPLxPTL. Une fois qu'il a lu et affiché les données d'une ligne, le hardware rajoute l'équivalent en octets à ces adresses – 40 octets pour un écran de 320 pixels de large –, puis il y rajoute un modulo. Ce modulo est stocké dans BPL1MOD pour les bitplanes impairs (1, 3, etc.), et BPL2MOD pour les bitplanes pairs (2, 4, etc.).

Sachant que le hardware lit donc dans des registres que le Copper permet de modifier (au moins) à chaque ligne de l'écran, on conçoit facilement comment tirer profit de ce fonctionnement. Prenons l'exemple d'une image de 320 x 256 pixels en 16 couleurs, donc 4 bitplanes.

Par exemple, il est possible de demander au Copper d'attendre le début d'une ligne et d'écrire dans les registres BPLxPTH/L pour modifier l'adresse de la ligne qui sera affichée. Cette adresse sera déterminée en fonction du zoom, pour sauter ou répéter des lignes, ou simplement passer à la ligne suivante. La Copper list contient alors notamment un bloc suivant par ligne N, ici présenté en pseudo-code pour en faciliter la lecture :

```
WAIT <début de ligne N>
MOVE <valeur>, BLTP1PTH
MOVE <valeur>, BLTP1PTL
MOVE <valeur>, BLTP2PTH
MOVE <valeur>, BLTP2PTL
MOVE <valeur>, BLTP3PTH
MOVE <valeur>, BLTP3PTL
MOVE <valeur>, BLTP4PTH
MOVE <valeur>, BLTP4PTL
```

Qui est certain de l'alignement des données en mémoire peut se dispenser d'écrire dans BPLxPTH sachant que sa valeur ne changera pas quel que soit l'offset rajouté pour progresser dans les bitplanes, mais peu importe : c'est le principe qu'on illustre ici.
Il est aussi possible de simplement d'écrire dans BPL1MOD et BPL2MOD. Dans ce cas, c'est l'adresse de la ligne suivante, et non l'adresse de la ligne courante, qui sera impactée, le hardware utilisant les modulos en fin de ligne. La Copper list contient alors notamment un bloc suivant par ligne N, ici encore présenté en pseudo-code pour en faciliter la lecture :

```
WAIT <début de ligne N-1>
MOVE <valeur>, BPL1MOD
MOVE <valeur>, BPL2MOD
```

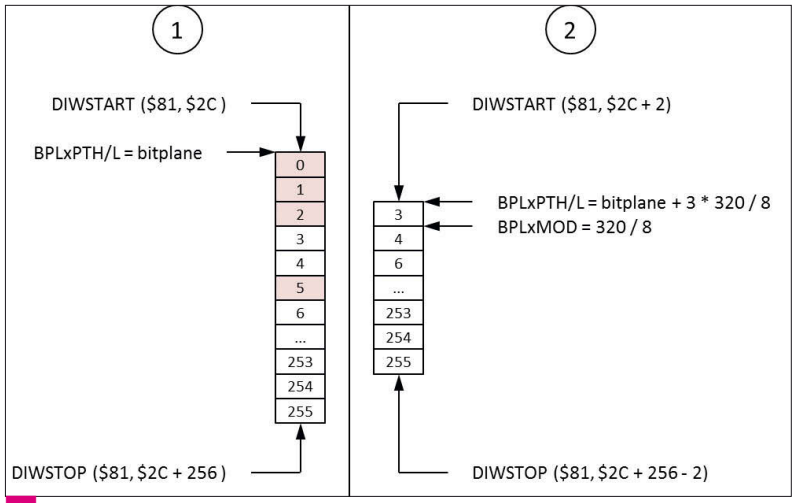
Utiliser BPLxMOD affecte tous les bitplanes impairs et/ou tous les bitplanes pairs, sans discrimination. Toutefois ce n'est pas gênant, car le zoom à réaliser est généralement le seul effet à produire dans les bitplanes. Par ailleurs, utiliser ces registres est plus économique qu'utiliser BPLxPTH/L.
En effet, il ne faut pas oublier que le facteur de zoom variant d'une trame à l'autre, la Copper list devra être modifiée au CPU ou au Blitter pour mettre à jour les valeurs que le Copper écrit dans les registres utilisés. Or le calcul est vite fait :

- avec BPLxPTH/L, il faut modifier les valeurs de deux MOVE par bitplanes, soit 8 valeurs au total (éventuellement 4 en jouant sur l'alignement en mémoire) ;
- avec BPLxMOD, il faut modifier les valeurs de deux MOVE tout cours, soit 2 valeurs au total.

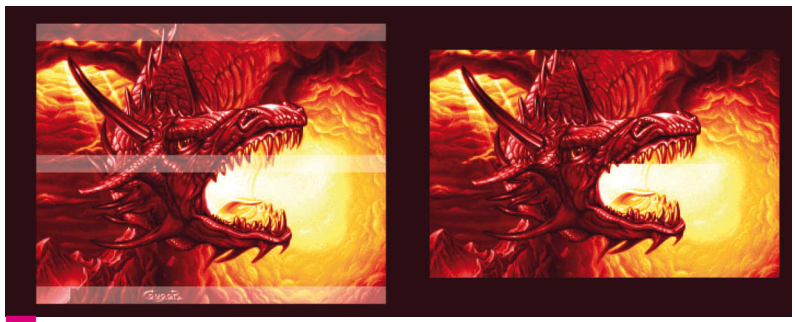
Toutefois, il ne suffit pas de dissimuler des lignes pour produire un zoom : à toute étape, il faut que l'image reste centrée à l'écran. Or modifier les modulos entraîne le tassement de l'image vers le haut de l'écran. Pour compenser il faut repousser vers le bas l'image de la moitié du nombre de lignes dissimulées. C'est possible grâce au moins à deux solutions :

- modifier la position verticale de départ de l'affichage dans le registre DIWSTRT, et la hauteur de cet affichage dans le registre DIWSTOP ;
- modifier la position verticale d'un WAIT du Copper, position à partir de laquelle des MOVE modifient les registres BPLxPTH/L pour pointer sur les adresses de départ des bitplanes de l'image zoomée.

Dans l'un et l'autre cas, il est inutile de modifier la position verticale des WAIT qui indiquent au Copper à quelle ligne il doit attendre avant d'exécuter le MOVE qui, en écrivant une valeur dans BPLxMOD, entraîne ou non la dissimulation d'une ou plusieurs lignes. En effet, à chaque étape du zoom, il suffit d'actualiser directement dans le code de la Copper list les valeurs que ces MOVE écrivent dans BPLxMOD, en tenant compte de la nouvelle position verticale de l'image pour sélectionner ces MOVE. Bref, s'il faut réduire la hauteur d'une image de 256 à 0 pixels en N étapes, il suffit de construire une Copper list qui contient toujours 256 WAIT suivis de MOVE sur BPLxMOD, et de modifier à chaque étape les valeurs que ces MOVE écrivent dans BPLxMOD pour animer le zoom. Dans le programme final `zoom4.s`, c'est la première solution qui a été adoptée par simplicité. Il faut noter que cela implique de



2 *Principe du zoom vertical hardware.*



3 *Zoom vertical hardware d'une image.*

modifier BPLxPTH/L pour parvenir à dissimuler une ou plusieurs des premières lignes. En effet, le modulo est une valeur que le hardware ajoute à l'adresse de la ligne qui vient d'être affichée. Or par définition, la première ligne ne vient après aucune autre ligne. Toutefois, c'est le seul cas où ces registres sont modifiés pour zoomer.
Par ailleurs, il faut noter que le faisceau d'électrons ne trace donc rien au-dessus et en-dessous de l'image zoomée. Dans ces conditions, il est impossible d'afficher des bandeaux encadrant l'image zoomée. Pour ce faire, il faut opter pour la seconde solution.
La figure 2 récapitule ce qui se passe dans le cas où les lignes 0, 1, 2 et 5 de l'image doivent être dissimulées :

- Comme il s'agit de dissimuler quatre lignes, DIWSTART est modifié pour que l'affichage démarre deux lignes plus bas, assurant ainsi que l'image reste verticalement centrée à l'écran.
- En conséquence, DIWSTOP est aussi réduit pour que le hardware n'affiche pas du *garbage* après avoir affiché la 256ème ligne de l'image.
- Pour dissimuler les trois premières lignes, impossible d'utiliser BPLxMOD. C'est donc BPLxPTH/L qui est modifié avant le début de l'affichage de la ligne 3.
- Pour dissimuler la ligne 5, BPLxMOD est passé à l'équivalent d'une ligne en octets, soit 40 octets, avant la fin de l'affichage de la ligne 4.

Le programme `zoom0.s` constitue un exemple de base où les 16 premières lignes, les 16 lignes médianes et les 16 dernières lignes d'une image sont ainsi dissimulées – pour la beauté du geste, c'est l'image Dragon Sun de Cougar / Sanity qui est utilisée (Figure 3).

A ce stade, il reste un point essentiel à régler : comment identifier les lignes à dissimuler à une étape du zoom donnée ? Cette question, qui se pose presque dans les mêmes termes pour le choix des colonnes à dissimuler, sera abordée plus loin.

Le zoom horizontal hardware, un détournement surprenant du scrolling

L'histoire dira à qui revient le mérite d'avoir trouvé l'astuce (on évoque le génial Chaos / Sanity). Comme chacun sait, il est possible de demander au hardware de retarder l'affichage de l'image à l'écran d'un nombre de pixels allant de 0 à 15. La valeur de ce retard doit être spécifiée dans le registre BPLCON1, qui comporte 4 bits (PF1H3-0) pour le retard des bitplanes impairs (1, 3, etc.) et 4 bits (PF2H3-0) pour le retard des bitplanes impairs (2, 4, etc.). C'est le scrolling hardware.

Ainsi, une valeur de $005F dans BPLCON1 va retarder l'affichage des bitplanes pairs de 5 pixels et celui des bitplanes impairs de 15 pixels. A moins d'utiliser le dual-playfield ou de chercher à manipuler distinctement les bitplanes pairs et impairs, les retards spécifiés pour ces bitplanes seront identiques. Il s'agira de produire un scrolling horizontal en jouant BPLCON1 et sur les couples BPLxPTH / BPLxPTL – on ne s'étendra pas sur ce sujet trivial ici.

Or le hardware lit les données qu'il va afficher par groupe de 16 pixels. A chaque fois, il tient compte des décalages spécifiés dans BPLCON1 pour retarder plus ou moins l'affichage de ce groupe. Que se passerait-il si la valeur de ces décalages devait être réduite d'un groupe à un autre ? En particulier, le hardware n'afficherait-il pas plus tôt le second groupe, écrasant les derniers pixels du premier ?

C'est exactement ce qui se passe (Figure 4). Et comme dissimuler des colonnes de pixels dans l'image produit une réduction de la largeur de cette dernière, il y a lieu de parler de zoom horizontal hardware.

Encore faut-il régler une question pratique : quand et comment modifier la valeur des décalages dans BPLCON1 ? Comme cela vient d'être suggéré, il faut que l'écriture dans BPLCON1 soit synchronisée sur le cycle de lecture et d'affichage des groupes de pixels par le hardware.

Chacun sait qu'il est toujours possible d'attendre une position du faisceau d'électron (le raster) à l'écran, et de modifier le contenu d'un registre hardware à cet instant. Cette manœuvre peut être réalisée au CPU, mais elle mobilise ce dernier. S'il fallait réaliser le zoom hardware de cette manière, il serait impossible de rien faire d'autre durant une trame.

Fort heureusement, le Copper est là, qui exécute sa Copper list parallèlement. Pourquoi ne pas utiliser ses instructions WAIT et MOVE, pour parvenir au résultat souhaité ?

Le programme `zoom1.s` en donne une illustration. L'idée va être d'attendre la position à laquelle on souhaite dissimuler un certain nombre des derniers pixels d'un groupe donné.

Mais où attendre ? A vrai dire, il y aurait deux solutions : utiliser un WAIT puis un MOVE pour modifier BPLCON1. Ou alors, comme dans le cas d'un effet plasma, enchaîner les MOVE en sachant qu'un MOVE prend 8 pixels en basse résolution pour être exécuté, dont un MOVE pour modifier BPLCON1. Dans les faits, seule la seconde solution est praticable.

Pourquoi ? Parce que c'est comme ça. Tous les coders qui utilisent le zoom horizontal hardware le font ainsi. Pourquoi, encore ? Parce qu'il faudrait vraiment, mais alors vraiment, se casser la tête pour parvenir à calculer précisément la position horizontale à spécifier dans le WAIT pour que le MOVE soit exécuté au bon moment – et encore, il n'est pas dit que ce serait possible. A ceux qui se lamenteraient de ce manque de rigueur, disons que pour aller au fond des choses, il faudrait savoir expliquer à tout instant le rapport entre la position horizontale du faisceau d'électron telle qu'elle est formulée dans un WAIT du Copper et la valeur horizontale telle qu'elle figure dans DDFSTRT. Or si c'est certainement possible, cela reste à faire…

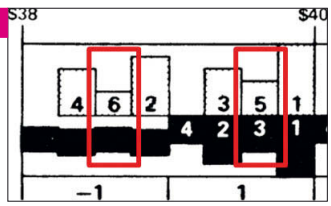
Ainsi le Copper est programmé pour attendre le début de chaque ligne à la position horizontale `$3D`, empiriquement déterminée, et procéder à 40 MOVE dont certains vont modifier la valeur de BPLCON1 en réduisant le retard initial, eux aussi empiriquement déterminés.

Il faut noter qu'une telle sollicitation du Copper contraint le nombre de bitplanes, donc le nombre de couleurs à l'écran. C'est qu'au-delà de 4 bitplanes, le hardware vole des cycles au Copper, si bien que ce dernier perd la possibilité d'exécuter autant de MOVE par ligne (Figure 5).

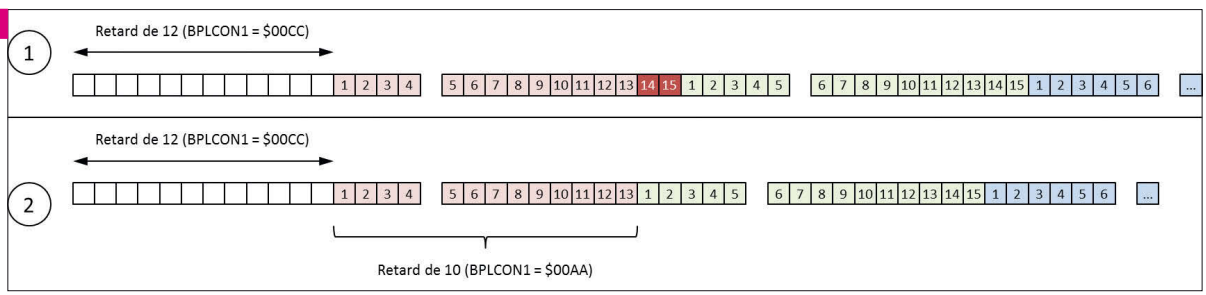
Dans `zoom1.s`, la partie de la Copper list concernant le zoom (après une initialisation de BPLCON1 à `$00FF` toutefois) se présente ainsi :

```
;Zoom

move.w #ZOOM_Y<<8,d0
move.w #ZOOM_DY-1,d1
_zoomLines:
```

5 *Le hardware vole des cycles vidéo au Copper pour afficher plus de 4 bitplanes.*



4 *Dissimulation des 2 derniers pixels d'un groupe en réduisant le scrolling hardware de 2.*




```
    ;Attendre le début la ligne

    move.w d0,d2
    or.w #$00!$0001,d2
    move.w d2,(a0)+
    move.w #$8000!($7F<<8)!$FE,(a0)+

    ;Initialiser BPLCON1 avec une retard de 15 pixels ($00FF)

    move.w #BPLCON1,(a0)+
    move.w #$00FF,(a0)+

    ;Attendre la position sur la ligne correspondant au début de l'affichage (position horizontale $3D dans un WAIT)

    move.w d0,d2
    or.w #ZOOM_X!$0001,d2
    move.w d2,(a0)+
    move.w #$8000!($7F<<8)!$FE,(a0)+

    ;Enchaîner des MOVE qui ne font rien jusqu'à celui qui doit passer le retard à ZOOM_BPLCON1

    IFNE ZOOM_MOVE   ;Car ASM-One plante sur un REPT dont la valeur est 0...
    REPT ZOOM_MOVE
    move.l #ZOOM_NOP,(a0)+
    ENDR
    ENDC

    ;Modifier BPLCON1 pour passer le retard à ZOOM_BPLCON1

    move.w #BPLCON1,(a0)+
    move.w #ZOOM_BPLCON1,(a0)+
```

```
    ;Enchaîner des MOVE qui ne font rien jusqu'à la fin de la ligne

    IFNE 39-ZOOM_MOVE       ;Car ASM-One plante sur un REPT dont la valeur est 0...
    REPT 39-ZOOM_MOVE
    move.l #ZOOM_NOP,(a0)+
    ENDR
    ENDC

    ;Passer à la ligne suivante de la bande de lignes zoomées

    addi.w #$0100,d0
    dbf d1,_zoomLines

    ;Réinitialiser BPLCON1 ($00FF) pour la fin de l'écran

    move.w #BPLCON1,(a0)+
    move.w #$00FF,(a0)+
```

La Copper list contient alors notamment un bloc suivant par ligne N, ici encore présenté en pseudo-code pour en faciliter la lecture :

```
WAIT ($00, N)
MOVE #$00FF, BPLCON1
WAIT ($3D, N)
REPT ZOOM_MOVE
MOVE #$000, ZOOM_NOP
ENDR
MOVE #ZOOM_BPLCON1, BPLCON1
REPT 39-ZOOM_MOVE
MOVE #$000, ZOOM_NOP
ENDR
```

La suite de cet article dans le prochain numéro

Le Dieu du Random décidera

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Nos experts techniques : D. Duplan, C. Pichaud, Y. Bennani, D. Nicolet, P. Boulanger, C. Savinas, Y. Grenzinger, D. Panza, C. Villeneuve, J. Nourry, F. Jadouani, C. Lebrun, N. Decoster, J. Blier, J. Giacomini, L. Ellerbach, R. Clark
Couverture : © D.R. - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, novembre 2018

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Abonnez-vous

www.solutions-numeriques.com/abonnement/

4 sites thématiques, pour répondre à vos besoins d'information

- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ?
  2 sites répondent à votre profil
- La cybersécurité vous concerne ?
  Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc
- L'emploi, les salaires, les formations, les offres vous intéressent ?
  Le site sur l'Emploi dans le numérique est à votre disposition

www.solutions-numeriques.com

PROLONGATION JUSQU'AU 21 DÉCEMBRE

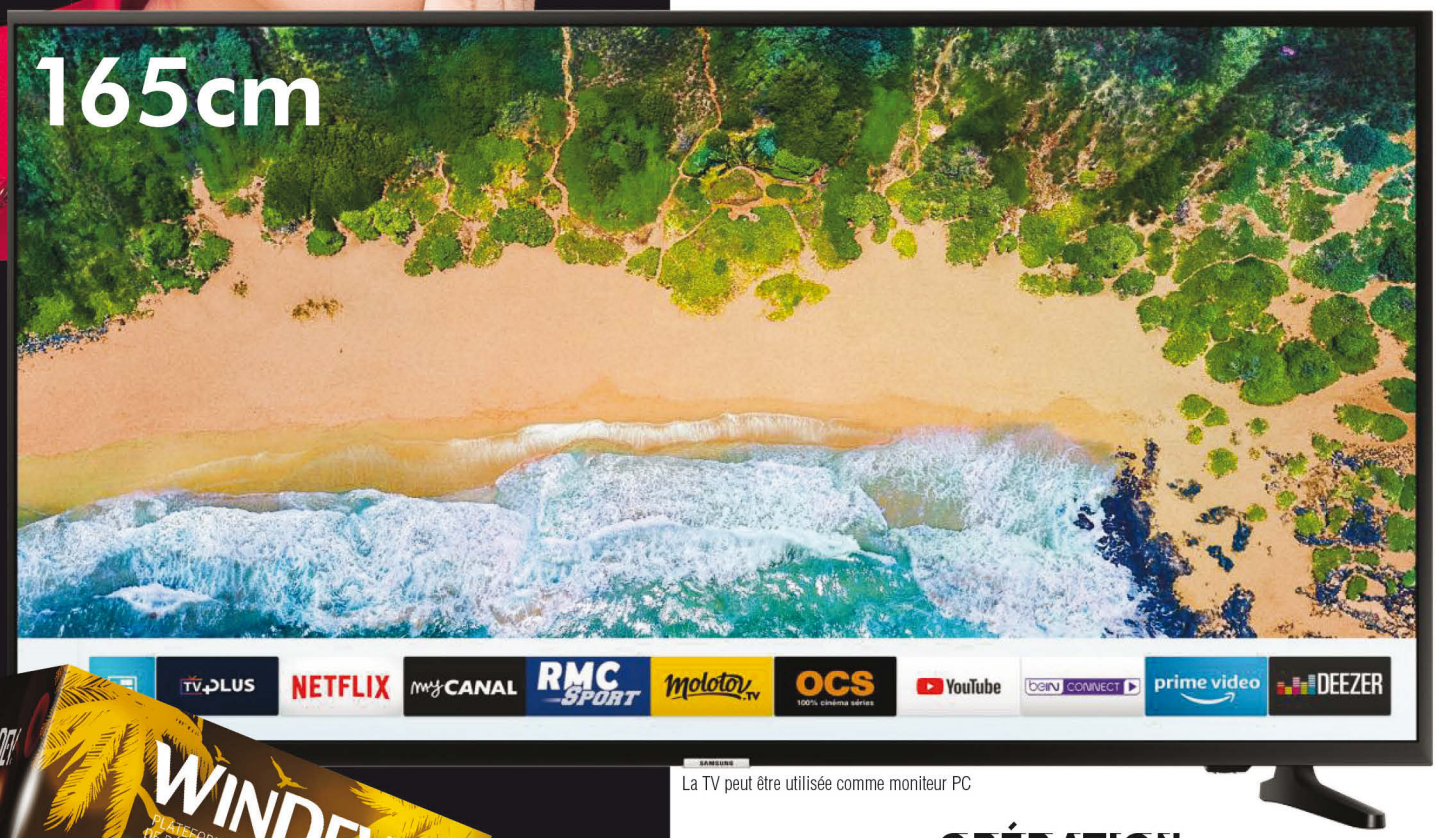
COMMANDEZ WINDEV 24

OU WEBDEV 24 OU WINDEV MOBILE 24

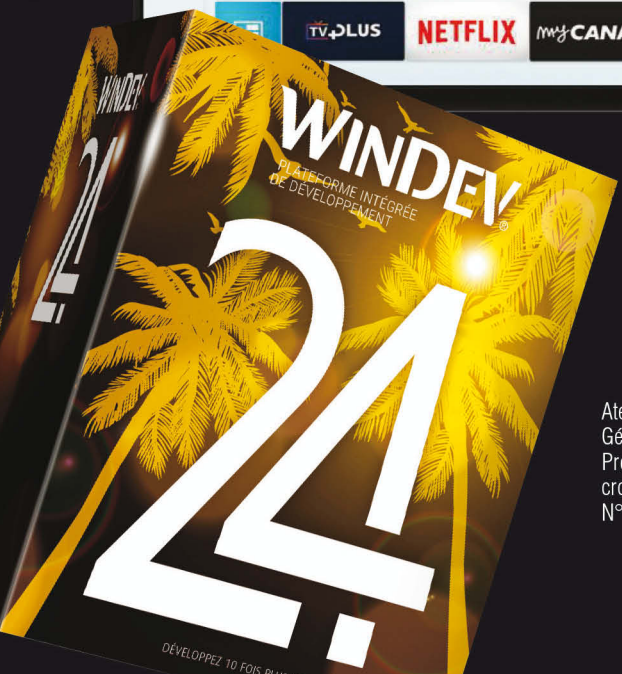
ET RECEVEZ UNE SUPERBE SMART TV 4K 165CM

SAMSUNG

POUR 1EURO DE +



La TV peut être utilisée comme moniteur PC



Atelier de Génie Logiciel Professionnel cross-plateformes. N°1 en France

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 24 (ou WINDEV 24, ou WEBDEV 24) chez PC SOFT au tarif catalogue avant le 21 Décembre 2018. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. **Voir tous les détails sur : WWW.PCSOFT.FR ou appelez-nous au 04.67.032.032**
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.980,00 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels. Tarifs modifiables sans préavis.

WWW.PCSOFT.FR