1

Introduction

Python – why settle for snake oil when you can have the whole snake ?

« Python - Pourquoi se contenter d'huile de serpent
quand on peut avoir le serpent tout entier ? »

Mark Jackson

En guise d'introduction, ce premier chapitre présente quelques caractéristiques de Python et renvoie aux chapitres consacrés. S'ensuit une comparaison avec d'autres langages. Le souhait n'est pas d'être exhaustif dans cet exercice de comparaison mais plutôt de situer Python dans l'esprit des développeurs familiers avec d'autres langages.

Python ?

Pour reprendre l'énoncé de l'avant-propos, Python est un langage :

- conçu pour produire du code **de qualité**, **portable** et **facile à intégrer** ;
- de haut niveau, **orienté objet** et totalement libre ;
- **hautement productif** ;
- **dynamique**.

De qualité

Grâce à sa syntaxe claire, cohérente et concise, présentée au **chapitre 4**, Python permet aux développeurs de produire du code de qualité, lisible et maintenable. Écrire du code Python est un exercice agréable, même en respectant les conventions de codages, présentées au **chapitre 7.**

Fourni dès le départ avec des modules de tests, Python est un langage *agile*. Le terme agile est originellement issu de la *méthodologie de programmation agile* (Beck et Al.), très proche de la programmation itérative. Cette méthodologie qui permet de réduire les risques liés à la conception de logiciels, introduit entre autres des principes de tests continus du code.

> http://www.agilemanifesto.org

Le **chapitre 12** présente les techniques de programmation dirigée par les tests appliquées à Python.

Orienté objet

Même si elle n'est pas imposée, Python permet la programmation orientée objet. Tous les mécanismes objets essentiels sont implémentés et toutes les données manipulées sont des instances de classes, comme avec les langages *SmallTalk* ou *Ruby*.

Enfin, le code peut être structuré en modules (fichiers) qui sont ensuite importables dans l'interpréteur. Ce découpage, inspiré de *Modula-3*, permet d'organiser le code et son utilisation par des espaces de noms, et aussi de faciliter l'extension du langage par des bibliothèques tierces compilées dans d'autres langages.

Le **chapitre 5** explique comment écrire des classes et structurer le code en modules et paquets, et le **chapitre 14** présente quelques *design patterns* (motifs de conception) orientés Python.

Portable

Python fonctionne sous différentes variantes d'Unix, Windows, Mac OS, BeOs, NextStep, et par le biais de différentes implémentations.

Les implémentations actuelles de Python sont :

- *Cpython* : implémentation en C, qui est l'implémentation par défaut de Python et la plus répandue ;

- *Jython* : implémentation en Java, qui permet d'exécuter du code source Python dans un environnement Java, et d'utiliser des modules Java dans le code Python de manière transparente ;
- *PyPy* : implémentation en Python du langage Python ;
- *IronPython* : implémentation pour .NET et Mono ;
- *Stackless Python* : une variante de CPython, légèrement plus rapide.

Il existe bien sûr des extensions spécifiques à chaque plate-forme, mais l'ensemble des primitives du langage et la majorité des extensions de la bibliothèque standard sont disponibles sur toutes les plates-formes.

En d'autres termes, un programme conçu sur une plate-forme fonctionnera directement, sauf programmation spécifique, sur d'autres plates-formes.

CPython, implémentation de référence pour cet ouvrage, peut être installé et utilisé sous Windows, MacOs et GNU/Linux (voir **chapitre 3**)

Facile à intégrer

Un programme écrit en Python s'intègre très facilement avec d'autres composants logiciels. Il est possible par exemple d'utiliser des bibliothèques C++ depuis un programme Python par le biais de `binds`, décrits dans le **chapitre 13**, ou de fournir des points d'accès à des programmes tiers en utilisant les différents protocoles de la bibliothèque standard.

Hautement productif

La conception d'applications en Python est très rapide car certains aspects de programmation sont gérés automatiquement, comme la gestion des ressources mémoire et le typage des données, décrits dans le **chapitre 4**.

Grâce à des types de base très puissants et des primitives de haut niveau, présentées dans le **chapitre 6**, un programme Python est simple à concevoir et concis. Un programme Python est en général 3 à 5 fois plus court qu'un programme C++ équivalent.

Ces qualités font de Python un langage idéal dans beaucoup de domaines, comme le **chapitre 2** le décrit.

Enfin, la bibliothèque standard de Python est très complète, et permet de répondre aux besoins communs de programmation. Les **chapitres 8, 9 et 10** présentent les modules les plus fréquemment utilisés.

Grâce au modèle Open Source, la communauté des développeurs Python est en outre très productive et de nombreuses extensions (voir **annexe B**) gravitent autour du langage

Dynamique

Python est un langage dynamique : le code source n'est pas compilé contrairement à des langages comme C ou Pascal, mais exécuté à la volée. On parle alors de *langage interprété*.

> **CULTURE Langage interprété et langage compilé**
>
> Un langage est dit interprété lorsque le système traduit et exécute chaque ligne d'un programme à la volée. Le résultat d'une modification peut être constatée en relançant tout simplement l'exécution du programme.
>
> À l'inverse, un langage compilé voit son code transformé de manière globale en instructions exécutables au moment de la compilation. Ce n'est qu'ensuite que vous pourrez exécuter le fichier produit lors de cette étape préalable. La modification d'une instruction du fichier source nécessite de repasser par l'étape compilation avant de pouvoir tester la nouvelle version.

Ce mode de fonctionnement rend la programmation beaucoup plus souple puisqu'il est possible de changer un programme en cours d'exécution, ou de tester du code en mode interactif sans disposition particulière.

Ce dynamisme fait partie également de la philosophie de programmation objet Python, basée sur le *duck typing*, décrit dans le **chapitre 14**.

L'interprétation rend aussi l'exécution plus lente mais ce défaut est surmontable grâce à de bonnes pratiques de programmation et des techniques d'optimisation décrites dans le **chapitre 13**.

Des applications où les performances sont un facteur critique ne seront pas écrites à 100 % en Python, mais pourront avantageusement être nivelées : un noyau codé en C, C++ ou tout autre langage compilé, et une couche supérieure en Python, pour toutes les parties non critiques.

Python et les autres langages

Si vous être habitué à un autre langage, cette section, sans vouloir faire un comparatif exhaustif, présente les différences majeures entre Python et certains langages.

Python et Perl

Le **chapitre 2** fournit des éléments de comparaison avec le langage Perl, relatifs à la programmation système. En attendant, voici un message humoristique publié sur la mailing-list Python il y a quelques années, qui décrit bien une des différences majeures entre Python et Perl : *la lisibilité*.

Comparaison de Perl et Python par Yoda

Sur la planète Dagobah.

Avec Yoda accroché dans son dos, Luke grimpe sur une des vignes qui poussent dans le marais pour atteindre le laboratoire de statistiques de Dagobah.

Il y continue ses exercices, greppe, installe des nouveaux paquets, se connecte en root, écrit des nouvelles versions de scripts en Python pour remplacer des scripts Perl vieux de deux ans.

Yoda :
Écris du code ! Oui. La force d'un programmeur découle de la maintenabilité de son code. Mais méfies-toi de Perl ! Syntaxe laconique, plus d'une manière de faire quelque chose ! Le côté obscur de la maintenabilité Perl est.

Si une seule fois par le chemin obscur tu t'engages, pour toujours ta destinée sera marquée.

***Luke** : est-ce que Perl est mieux que Python ?*

***Yoda** : non... non... non. Plus rapide, plus facile, plus séduisant.*

***Luke** : mais comment saurais-je pourquoi Python est mieux que Perl ?*

***Yoda** : tu sauras. Quand le code écrit il y a 6 mois de relire tu tenteras.*

Python et Java

Java est un langage à typage statique, doté d'une syntaxe proche du C++, beaucoup plus bavarde et moins concise que Python.

Cette différence se ressent beaucoup sur la productivité qui est aussi affectée par des types de base moins puissants et plus complexes à mettre en œuvre. Par exemple, les conteneurs de base de Java ne supportent pas le mélange d'objets et de types élémentaires comme les entiers.

En terme d'organisation de code, Java impose un découpage précis : une classe publique est dans un fichier, et il n'est pas possible d'avoir plusieurs classes dans le même fichier. Cette contrainte est relativement pesante dans certains types de programmes.

Contrairement au système d'exception de Python, les exceptions en Java ne peuvent pas remonter plusieurs niveaux de dérivation de classes si cette exception n'est pas ajoutée à la signature des classes de chaque niveau.

Enfin, Java ne permet pas d'implémenter de fonctions à nombre de paramètres variables.

Le manque de souplesse de Java face à Python est relativement conséquent.

Java reste cependant un poids lourd des langages de programmation, et possède des outils riches et puissants. Portable, car également interprété, Java reste parfois incontournable dans certains domaines. Python peut cependant être utilisé dans Java grâce à l'implémentation Jython.

Python et C++/C#

Les langages C++ et C# sont équivalents à Java en terme de syntaxe. C# a bénéficié d'améliorations importantes par rapport au C++, comme la suppression des pointeurs et l'ajout d'un ramasse-miettes, ce qui le rapproche beaucoup plus de Java sur ce terrain. Globalement, les mêmes constatations en terme de productivité et de lisibilité peuvent être faites entre Python et la famille des langages C.

C++ et C# sont cependant des langages compilés, ce qui les rend nettement plus rapides à l'exécution. Dans des domaines où les performances sont critiques, ils sont incontournables.

Il est possible de marier Python avec ces langages, comme décrit précédemment dans la section *dynamique*, par le biais d'extensions, pour profiter des qualités de Python pour les couches supérieures d'une application.

Python et PHP 5

PHP est très utilisé pour la programmation d'applications web, et a beaucoup de points en commun avec Python, à savoir :

- langage interprété ;
- Open Source ;
- facilement extensible ;
- programmation orientée objet possible.

Les différences majeures sont :

- la syntaxe : beaucoup moins lisible et concise en PHP, qui est très similaire à C++ et à Java ;
- le multi-threading : inexistant en PHP ;
- l'impossibilité de surcharger les opérateurs en PHP ;
- des types de base (listes, dictionnaires, tuiles) plus puissants en Python ;
- l'organisation modulaire de grosses applications est beaucoup plus facile en Python.

Lors de la conception d'applications web, un autre aspect de comparaison important est la qualité des frameworks d'applications disponibles.

Python possède des frameworks majeurs qui permettent la conception d'applications web, à savoir Zope et Twisted. De nombreux progiciels très puissants se sont greffés sur ces frameworks, comme le gestionnaire de contenu CPS.

PHP propose des frameworks comme WASP ou Zend framework qui sont moins évolués que leurs pendants Python, mais possèdent une multitude d'outils de gestion de contenu légers et très faciles à mettre en œuvre, comme SPIP, PHP-Nuke, DotClear, WordPress, etc.

Python et Ruby

Le langage Ruby a beaucoup de points communs avec Python, car c'est également un langage dynamique, doté d'un prompt interactif et où tout élément est objet. La syntaxe de Ruby est également très claire et concise.

Les différences majeures, au moment de l'écriture de ce livre, sont les suivantes :

- Ruby permet l'exécution de code dans un environnement restreint, fonctionnalité retirée à Python il y a quelques temps et sans équivalent aujourd'hui.
- Python reconnaît nativement Unicode, contrairement à Ruby.
- Seul Python permet l'utilisation de *threads préemptifs*. Ce mode géré par le système hôte, permet d'éviter des situations de blocage particulières.
- Python offre l'héritage multiple.
- Les bibliothèques disponibles sous Python sont plus complètes et plus matures à l'heure actuelle, même si Ruby tente de combler ce fossé (`scipy > sciruby`, `py2exe > rubyscript2exe`, etc.).
- Python propose des systèmes d'interfaces (zope.interfaces, PyProtocols).

Les avantages récurrents de Python face aux autres langages sont la puissance de ses types de bases et la concision de sa syntaxe. Un autre aspect important dans le choix d'un langage est le contexte dans lequel un programme est conçu. Le prochain chapitre présente les domaines d'applications les plus fréquents de Python.