Python (langage)

	
Apparu en	1990
Auteur	Guido van Rossum
Développeurs	Python Software Foundation
Dernière version	3.1 [1] (le 27 juin 2009) http://en.wikipedia.org/wiki/Modèle:dernière_version_stable/python_(langage) +/−
Version en développement	http://en.wikipedia.org/wiki/Modèle:dernière_version_avancée/python_(langage) +/−
Paradigmes	Objet, impératif
Typage	Fort, dynamique
Influencé par	ABC, C, Icon, Modula-3, Perl, Smalltalk, Tcl
A influencé	Ruby, Boo
Implémentations	CPython, Jython, IronPython, PyPy
Système d'exploitation	Multiplate-forme
Licence	Python Software Foundation License
Site Web	www.python.org [1]

Python est un langage de programmation interprété multi-paradigme. Il favorise la programmation impérative structurée, et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Le langage Python est placé sous une licence libre proche de la licence BSD[2] et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux, de Windows à Unix en passant par Linux et MacOS, avec Java ou encore .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser. Il est également apprécié par les pédagogues qui y trouvent un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation plus aisée aux concepts de base de la programmation.[3]

Utilisations

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées à chaque traitement. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses comme par exemple un script qui récupérerait la météo sur internet ou qui s'intégrerait dans un logiciel de conception assistée par ordinateur afin d'automatiser certains enchaînements d'actions répétitives. On l'utilise également comme langage de développement de prototype lorsqu'on a besoin d'une application fonctionnelle avant de l'optimiser avec un langage de plus bas niveau. Il est particulièrement répandu dans le monde scientifique, et possède de nombreuses extensions destinées aux applications numériques.

Histoire

Guido van Rossum a participé au développement du langage de programmation ABC au Centrum voor Wiskunde en Informatica (CWI) aux Pays-Bas.

Au CWI

Il travaillait alors dans l'équipe du système d'exploitation Amoeba dont les appels systèmes était difficilement interfaçables avec le bourne shell qui était utilisé comme interface utilisateur. Il estima alors qu'un langage de script inspiré d'ABC pourrait être intéressant comme interpréteur de commandes pour Amoeba[4]

En 1989, profitant d'une semaine de vacances durant les fêtes de Noël, il utilise son Macintosh personnel[5] pour écrire la première version du langage. Fan de la série télévisé des Monty Python, il décide de baptiser ce projet python. Il s'est principalement inspiré d'ABC, par exemple pour l'indentation comme syntaxe ou les types de haut niveau mais aussi de Modula-3 pour la gestion des exceptions, du langage C et des outils UNIX[6] .

Durant l'année suivante le langage commence à être adopté par l'équipe du projet Amoeba, Guido poursuivant son développement principalement pendant son temps libre. En février 1991 la première version publique, numérotée 0.9.0[7] , est postée sur le forum Usenet alt.sources. La dernière version sortie au CWI était Python 1.2.

Au CNRI

En 1995, Van Rossum continua son travail sur Python au CNRI à Reston, aux États-Unis, où il sortit plusieurs versions du logiciel.

A partir d'août 1995, l'équipe python travaille au CNRI sur *Grail*[8] un navigateur internet utilisant Tk. Il est l'équivalent pour python du navigateur HotJava, permettant d'exécuter des applets dans un environnement sécurisé. La première version publique, disponible en novembre, est la 0.2[9] . Il a entraîné le développement de modules pour la librairie standard comme *rexec*[10] , *htmllib* ou *urllib*[11] . La version 0.6 sera la dernière de *Grail*; elle est publiée en avril 1999[12] .

En 1999, le projet *Computer Programming for Everybody* (CP4E) est lancé avec collaboration entre le CNRI et la DARPA. Il s'agit d'utiliser python comme langage d'enseignement de la programmation. Cette initiative conduira à la création de l'environnement de développement IDLE. Les subventions fournies par la DARPA ne suffisant pas à pérenniser le projet, Guido doit quitter le CNRI[13] . Python 1.6 fut la

dernière version sortie au CNRI.

À BeOpen

Après la sortie de Python 1.6, et après que Van Rossum ait quitté le CNRI pour travailler avec des développeurs de logiciels commerciaux, le CNRI et la Free Software Foundation collaborèrent pour modifier la licence de Python afin de la rendre compatible avec la GPL. Python 1.6.1 est essentiellement le même que Python 1.6 avec quelques correctifs mineurs et la nouvelle licence compatible GPL.

En 2000, l'équipe principale de développement de Python déménagea à BeOpen.com pour former l'équipe PythonLabs de BeOpen. Python 2.0 fut la seule version sortie à BeOpen.com. Après cette version, Guido Van Rossum et les autres développeurs de PythonLabs rejoignirent Digital Creations.

Andrew M. Kuchling a publié en décembre 1999[14] un texte nommée *python warts*[15] qui synthétise les griefs les plus fréquents exprimés à l'encontre du langage. Ce document aura une influence certaine sur les développements futurs du langage[16] .

La Python Software Foundation

Python 2.1 fut une version dérivée de Python 1.6.1, ainsi que de Python 2.0. Sa licence fut renommée Python Software Foundation License. Tout code, documentation et spécifications ajouté, depuis la sortie de Python 2.1 alpha, est détenu par la Python Software Foundation (PSF), une association sans but lucratif fondée en 2001, modelée d'après l'Apache Software Foundation.

Un développement parallèle, entamé en 2004, est en discussion pour Python 3.0. Cette nouvelle version cassera la compatibilité ascendante afin de réparer certains défauts du langage (orientation objet avec deux types de classes, etc), et pour nettoyer la bibliothèque standard de ses éléments obsolètes et redondants. La première version alpha est sortie en septembre 2007.

Caractéristiques

Syntaxe

Python a été conçu pour être un langage lisible. Il vise à être visuellement épuré, et utilise des mots anglais fréquemment là où d'autres langages utilisent de la ponctuation, et possède également moins de constructions syntaxiques que de nombreux langages structurés tels que C, Perl, ou Pascal. Les commentaires sont indiqués par le caractère dièse.

Les blocs sont identifiés par l'indentation, au lieu d'accolades comme en C/C++, ou de Begin ... End comme en Pascal. Une augmentation de l'indentation marque le début d'un bloc, et une réduction de l'indentation marque la fin du bloc courant. Les parenthèses sont facultatives dans les structures de contrôle :

Fonction factorielle en C	Fonction factorielle en Python

```c
// Fonction factorielle en C
int factorielle(int x)
{
    if (x == 0)

        return 1;

    else
        return x *
factorielle(x-1);
}
```

```python
# Fonction factorielle en python
def factorielle(x):
    if x == 0:
        return 1
    else:
        return x * factorielle(x-1)
```

Types de base

Les types de base en Python sont relativement complets et puissants, il y a entre autres :

- Les objets numériques
  - `int` est un entier classique.
  - `long` est un entier dont la mémoire de stockage est allouée dynamiquement, la taille maximale n'est donc pas fixe et dépend de la capacité de la machine.
  - `float` est un flottant équivalent au double du C.
  - `complex` est une approximation d'un nombre complexe (typiquement deux `floats`)
- Les objets "itérables"
  - Les objets `list` sont des tableaux dynamiques (ils étendent automatiquement leur taille lorsque nécessaire) et acceptent des types de données hétérogènes.
  - Les objets `set` sont des ensembles non ordonnés d'objets.
  - Les objets `dict` sont des tableaux associatifs (ou dictionnaires) permettant d'associer un objet (une clef) à un autre.
  - Les objets `str` sont des chaînes de caractères, qui possèdent une multitude de méthodes de base pour remplacer des mots, éliminer les espaces blancs, séparer le texte en morceau, etc. Ce sont des objets itérables car ils peuvent être vus comme des listes d'autres `str` d'un seul caractère.
  - Les `tuple` (ou `n-uplet`) sont une variante non mutable des listes ; de même il existe une version immutable des dictionnaires (`frozendict`) et des ensembles (`frozenset`).

Les objets itérables sont parcourus à l'aide d'une boucle `for` de la manière suivante :

```
for element in objet_iterable:
  traiter(element)
```

Il est possible de dériver les classes des types de base pour créer ses propres types. On peut également fabriquer ses propres types d'objets itérables sans hériter des itérables de base en implémentant le protocole d'itération du langage.

Programmation fonctionnelle

Python permet de programmer dans un style fonctionnel. Les compréhensions de listes sont disponibles. Par exemple, pour construire la liste des carrés des entiers naturels plus petits que 10, on peut utiliser l'expression :

```
l = [x**2 for x in range(10)]
# l = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

la liste des nombres **pairs** :

```
l = [entier for entier in range(10)
     if entier % 2 == 0]
# l = [0, 2, 4, 6, 8]
```

Une forme limitée de fonctions lambda, ou fonctions anonymes, est disponible :

```
lambda x: x+2
```

Les fonctions lambda peuvent être définies en ligne et utilisées comme arguments dans des expressions fonctionnelles : `filter(lambda x: x < 5, une_liste)` retournera une liste constituée des éléments de une_liste inférieurs à 5. Le même résultat peut être obtenu avec `[x for x in une_liste if x <5]` .

Les lambdas de Python n'admettent que des expressions et ne peuvent être utilisées comme fonctions anonymes généralisées ; mais en Python, toutes les fonctions sont des objets, qui peuvent donc être passés en arguments d'autres fonctions, et appelés lorsque nécessaire.

Programmation objet

La programmation objet est très bien supportée par Python : tous les types de base, les fonctions, les instances de classes (les objets « classiques » des langages C++ et Java) et les classes elles-mêmes (qui sont des instances de méta-classes) sont des objets.

Une classe se définit avec le mot `class`. Les classes Python supportent l'héritage multiple ; il n'y a pas de surcharge statique comme en C++, mais le mécanisme des arguments optionnels et par mot-clef est plus général et plus flexible. En Python, l'attribut d'un objet peut référencer une variable d'instance ou de classe (le plus souvent une méthode). Il est possible de lire ou de modifier un attribut dynamiquement avec les fonctions :

- `getattr(objet, 'nom_attribut')`
- `setattr(objet, 'nom_attribut', nouvel_attribut)`

Exemple de deux classes simples :

```
class Personne:
   def __init__(self, nom, prenom):
       self.nom = nom
       self.prenom = prenom
   def presenter(self) :
       return self.nom+' '+self.prenom

class Etudiant(Personne):
   def __init__(self, classe, nom, prenom):
       Personne.__init__(self, nom, prenom)
```

```
        self.classe = classe
    def presenter(self):
        return self.classe + Personne.presenter(self)

e = Etudiant('Licence INFO', 'Dupontel', 'Albert')
assert e.nom == 'Dupontel'
```

Méthodes spéciales et surcharge des opérateurs

Python fournit un mécanisme élégant et orienté objet pour la surcharge des opérateurs: tout objet python peut se voir doter de méthodes dites spéciales.

Ces méthodes, commençant et finissant par deux underscores, sont appelées lors de l'utilisation d'un opérateur sur l'objet : + (méthode __add__), += (méthode __iadd__), [] (méthode __getitem__), () (méthode __call__), ... Des méthodes comme __repr__ et __str__, permettent de définir la représentation d'un objet dans l'interpréteur interactif et son rendu avec le mot clé *print*.

Les possibilités sont nombreuses et sont décrites dans la documentation du langage : http://docs.python.org/ref/specialnames.html.

Par exemple on peut définir l'addition de deux vecteurs à 2 dimensions avec la classe suivante :

```
class Vector2D:
    def __init__(self, x, y):
        # On utilise un tuple pour stocker les coordonnées
        self.coords = (x, y)

    def __add__(self, other):
        # L'instruction a+b sera résolue comme a.__add__(b)
        # On construit un objet Vector2D à partir des coordonnées
propres à l'objet, et à l'autre opérande
        return Vector2D(self.coords[0]+other.coords[0],
self.coords[1]+other.coords[1])

    def __repr__(self):
        # L'affichage de l'objet dans l'interpréteur
        return "Vector2D(%s, %s)" %self.coords

>>> a = Vector2D(1, 2)
>>> b = Vector2D(3, 4)
>>> print a+b
Vector2D(4, 6)
```

Générateurs

Le mot-clef *yield* utilisé dans une fonction permet de faire de cette fonction un générateur. L'appel de cette fonction renvoie un objet de type *generator*, qui peut être utilisé dans une boucle *for*, par exemple.

À chaque appel, le générateur effectue son traitement jusqu'à rencontrer le mot-clé *yield*, renvoie la valeur de l'expression devant *yield*, et à l'appel suivant, reprend son déroulement juste après le *yield*. Par exemple pour calculer la suite de Fibonacci, on peut faire :

```
import sys
def gen_fibonacci(max=sys.maxint):
    a=0
    b=1
    while a<max:
        yield a         # Renvoi du "a" en cours
        a,b = b,a+b
for n in gen_fibonacci(1000):
    print n
```

Réflexivité

Grâce à un usage intensif des dictionnaires (conteneur associatif implémenté avec des tables de hachage), Python permet d'explorer les divers objets du langage (introspection) et dans certains cas de les modifier (intercession).

Typage

Le typage n'est pas vérifié à la compilation. De ce fait, des opérations sur un objet peuvent échouer, signifiant que l'objet en question n'est pas du bon type. Malgré l'absence de typage statique, Python est fortement typé, interdisant des opérations ayant peu de sens (comme, par exemple, additionner un nombre à une chaîne de caractères) au lieu de tenter silencieusement de la convertir en une forme qui a du sens. Python propose des fonctions permettant de transformer les variables dans un autre type :

```
points = 3.2 # points est du type float
print "Tu as " + points + " points !" # Génère une erreur de typage
points = int(points) # points est maintenant du type int (et vaut 3)
print "Tu as " + points + " points !" # Génère une erreur de typage
points = str(points) # points est maintenant du type str (chaîne de
caractères)
print "Tu as " + points + " points !" # Plus d'erreur de typage,
affiche 'Tu as 3 points !'
```

De même, chaque variable devra être déclarée avant d'être utilisée.

Python propose aussi un mécanisme de typage fort grâce à l'API *trait* ou au design pattern *decorators*.

Compilation

Il est cependant possible d'effectuer une analyse statique des modules Python avec des outils comme Pylint ou PyChecker. Ceux-ci éliminent la plupart des fautes, comme une classe qui hérite d'une classe abstraite et qui ne surcharge pas les méthodes abstraites, ou bien des variables utilisées avant d'être déclarées, ou encore des attributs d'instance déclarés en dehors de la méthode __init__, etc. En fonction de la nature du projet, ces outils peuvent être réglés de façon fine afin d'obtenir une très grande qualité de code.

Il est aussi possible de générer un Bytecode Python.

Des outils comme PyInstaller[17] ou d'autres plus spécifiques comme Freeze sous Unix et py2exe sous Windows permettent de « compiler » un programme Python sous forme d'un exécutable comprenant le programme et un interpréteur Python. Le programme ne tourne pas plus rapidement (il n'est pas compilé sous forme de code machine) mais cela simplifie largement sa distribution, notamment sur des machines où l'interpréteur python n'est pas installé.

Modèle objet

En python *tout est objet*, dans le sens qu'une variable peut contenir une référence vers tous les éléments manipulés par le langage : nombres, méthodes, modules, etc.[18] Néanmoins, avant la version 2.2, les classes et les instances de classes étaient un type d'objet particulier, ce qui signifiait qu'il était par exemple impossible de dériver sa propre sous-classe de l'objet *list*.

Méthodes

Le modèle objet de python est inspiré de celui de Modula-3[19] . Parmi ces emprunts se trouve l'obligation de déclarer l'instance de l'objet courant, conventionnellement nommée *self*, comme premier argument des méthodes, et à chaque fois que l'on souhaite accéder à une donnée de cette instance dans le corps de cette méthode. Cette pratique n'est pas naturelle pour des programmeurs venant par exemple de C++ ou Java, la profusion des *self* étant souvent critiquée comme étant une pollution visuelle qui gène la lecture du code. Les promoteurs du *self* explicite estiment au contraire qu'il évite le recours à des conventions de nommage pour les données membres et qu'il simplifie des tâches comme l'appel à une méthode de la superclasse ou la résolution d'homonymie entre données membres[20] . Il permet par ailleurs un traitement orthogonal des méthodes et fonctions.

Python reconnaît trois types de méthodes :

- les méthodes d'instances, qui sont celles définies par défaut. Elle reçoivent comme premier argument une instance de la classe où elles ont été définies.
- les méthodes de classes, qui reçoivent comme premier argument la classe où elles ont été définie. Elles peuvent être appelées depuis une instance ou directement depuis la classe. Elles permettent de définir des constructeurs alternatifs comme la méthode *fromkeys()* de l'objet *dict*.
- les méthodes statiques, qui ne reçoivent pas de premier argument implicite. Elles sont similaires aux méthodes statiques que l'on trouve en Java ou C++.

Visibilité

Le langage a un support très limité de l'encapsulation. Il n'y a pas, comme en java par exemple, de contrôle de l'accessibilité par des mots clefs comme *protected* ou *private*.

La philosophie de python est de différencier conceptuellement l'encapsulation du masquage d'information. Le masquage d'information vise à prévenir les utilisations frauduleuses, c'est une préoccupation de sécurité informatique . Le module *bastion* de la librairie standard, qui n'est plus maintenu dans les dernières versions du langage, permettait ainsi de contrôler l'accès aux attributs d'un objet dans le cadre d'un environnement d'exécution restreint.

L'encapsulation est une problématique de développement logiciel. Le slogan des développeurs python est *we're all consenting adults here*[21] (nous sommes entre adultes consentants). Ils estiment en effet qu'il suffit d'indiquer, par des conventions d'écriture, les parties publiques des interfaces et que c'est aux utilisateurs des objets de se conformer à ces conventions ou de prendre leurs responsabilités. L'usage est de préfixer par un underscore les membres privés. Le langage permet par ailleurs d'utiliser un double underscore pour éviter les collisions de noms, en préfixant automatiquement le nom de la donnée par celui de la classe où elle est définie.

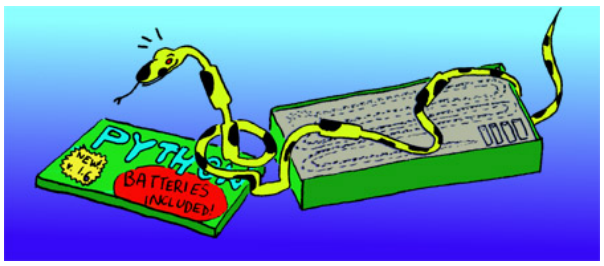
L'utilisation de la fonction property() permet de définir des propriétés qui ont pour but d'intercepter, à l'aide de méthodes, les accès à une donnée membre. Cela rend inutile la définition systématiques d'accesseurs et le masquage des données comme il est courant de le faire en C++ par exemple.

Héritage

Python supporte l'héritage multiple. Depuis la version 2.3, il utilise l'algorithme C3, issu du langage Dylan[22] , pour résoudre l'ordre de résolution de méthode (*MRO]*). Les versions précédentes utilisaient un algorithme de parcours en profondeur qui posait des problèmes dans le cas d'un héritage en diamant[23] .

Bibliothèque standard

Python possède une grande bibliothèque standard, fournissant des outils convenant à de nombreuses tâches diverses. Le nombre modules de la bibliothèque standard peut être augmenté avec des modules spécifiques écrits en C ou en Python.



Python est fourni « Piles Incluses ».

La bibliothèque standard est particulièrement bien conçue pour écrire des applications utilisant Internet, avec un grand nombre de formats et de protocoles standards gérés (tels que MIME et HTTP). Des modules pour créer des interfaces graphiques et manipuler des expressions rationnelles sont également fournis. Python inclut également un framework de tests unitaire pour créer des suites de test exhaustives.

Interfaces graphiques

Python possède plusieurs modules disponibles pour la création de logiciels avec une interface graphique. Le plus répandu est Tkinter. Ce module convient à beaucoup d'applications et peut être considéré comme suffisant dans la plupart des cas. Néanmoins, d'autres modules ont été créés pour pouvoir lier Python à d'autres bibliothèques logicielles ('toolkit'), pour davantage de fonctionnalités, pour une meilleure intégration avec le système d'exploitation utilisé, ou simplement pour pouvoir utiliser Python avec sa bibliothèque préférée. En effet, certains programmeurs trouvent l'utilisation de Tkinter plus pénible que d'autres bibliothèques. Ces autres modules ne font pas partie de la bibliothèque standard et doivent donc être obtenus séparément.

Les principaux modules donnant accès aux bibliothèques d'interface graphique sont Tkinter pour Tk, wxPython pour wxWidgets, PyGTK pour GTK+, PyQt pour Qt, FxPy pour le FOX Toolkit, et Pmw pour Mega-Widgets, il existe aussi une implantation de SDL : Pygame, et un binding de la SFML : PySFML.

La communauté Python

Van Rossum est le principal auteur de Python, et son rôle de décideur central permanent de Python est reconnu avec humour par le titre de « Dictateur bienveillant à vie » (*Benevolent Dictator for Life*, BDFL).

Il est assisté d'une équipe de *core developers* qui ont un accès en écriture au dépôt de CPython et qui se coordonnent sur la liste de diffusion python-dev. Ils travaillent principalement sur le langage et la bibliothèque de base. Ils reçoivent ponctuellement les contributions d'autres développeurs Python via la plateforme de gestion de bug Roundup, qui a remplacé SourceForge.

Les utilisateurs ou développeurs de bibliothèques tierces utilisent diverses autres ressources. Le principal média généraliste autour de python est le forum Usenet anglais comp.lang.python.

Les allusions aux Monty Python sont assez fréquentes. Les didacticiels consacrés à python utilisent souvent les mots *spam* et *eggs* comme variable métasyntaxique. Il s'agit d'une référence à l'épisode 25 de la deuxième saison du Monty Python's Flying Circus, ou deux clients tentent de commander un repas à l'aide d'une carte qui contient du jambon en conserve (le spam) dans pratiquement tous les plats. Ce sketch a été aussi pris pour référence pour désigner un email non sollicité.

Adoption de Python

Plusieurs entreprises mentionnent sur leur site officiel[24] qu'elles utilisent Python :

- Google (Guido van Rossum travaille depuis 2005 au sein de cette entreprise[25]) ;
- Industrial Light & Magic ;
- la NASA ;
- et CCP Games, les créateurs du jeu vidéo EVE Online.

Implémentations

Outre l'implémentation de référence, nommée CPython (car écrite en langage C), il existe d'autres systèmes implémentant le langage Python :

- Stackless Python, une version de CPython n'utilisant pas la pile d'appel du langage C ;
- Jython, un interprète Python pour machine virtuelle Java. Il a accès aux bibliothèques fournies avec l'environnement de développement Java ;
- IronPython, un interprète / compilateur (expérimental) pour plateforme .Net / Mono ;
- PyPy un interprète Python écrit dans un sous-ensemble de Python compilable vers le C ou LLVM ;
- un compilateur (expérimental) pour Parrot, la machine virtuelle de Perl 6 ;
- Movable Python [26] (payant) : Python sur une clef USB pour lancer des scripts sans installer Python.

Ces autres implémentations ne bénéficient pas forcément de la totalité de la bibliothèque de fonctions écrites en C pour l'implémentation de référence.

Les distributions

Differentes distributions sont disponibles, qui incluent parfois beaucoup de packages dédiés à un domaine donnée:

- Python(x,y) : une distribution Python à l'usage des scientifiques basée sur Qt et Eclipsehttp://www.pythonxy.com/foreword_fr.php
- Enthought Python Distribution (EPD) : Distribution scientifiquehttp://www.enthought.com/products/epd.php

Historique des versions

Version	Date de sortie	Nouveautés
1.5(.2)	13 avril 1999	• Ajout du mot clé **assert** • Possibilité d'importer une hierarchie de modules (import spam.ham.eggs) • Nouveau module re qui remplace regex • Les exceptions sont maintenant des classes • Ajout de l'option -O qui supprime les assertions et informations de ligne de fichier
1.6	5 septembre 2000	• La méthode append() des listes n'accepte plus qu'un seul argument • Le résultat des fonctions str() et repr() est maintenant beaucoup plus souvent différent, exemple : str(1L)=='1' et repr(1L)=='1L' • Les chaînes de caractères ont maintenant des méthodes (" abc ".strip()) • Le module re est compatible avec l'ancien moteur, est plus rapide, et accepte les chaînes Unicode • Ajout du module distutils • Nouveau prototype "def f(*args, **kw)" pour les fonctions, avant il fallait utiliser apply() • int() et long() acceptent maintenant une base en second argument • L'opérateur in peut être surchagé avec une méthode __contains__()

2.0	16 octobre 2000	• Changement majeur : support d'Unicode • Ajout des compréhensions de liste (*List Comprehensions*) • Ajouts des opérateurs avec assignement (a+=b, a*=b, etc.) • Les chaînes str ont maintenant des méthodes • Nouveau ramasse-miettes à cycles • Nouveau prototype "def f(*args, **kw)" pour les fonctions, avant il fallait utiliser apply() • Ajout des modules distutils, xml.dom.minidom module et xml.sax
2.1	17 avril 2001	• Création du module __future__ pour rendre les transitions plus douces • Comparaisons riches (méthodes __lt__, __le__, ..., __ne__) • Framework d'avertissement • Ajout des modules inspect, pydoc, doctest, pyunit • Ajout des références faibles (*weak references*) • Les fonctions peuvent avoir des attributs • "*Nested Scopes*" • La version 2.0.1 (juin 2001 en informatique) sera la première version compatible GPL
2.2	21 décembre 2001	• Unification de Type et de Class: on peut maintenant hériter des types de base • Ajout des itérateurs et générateurs • Nouvel opérateur a // b pour la division entière
2.3	29 juillet 2003	• Ajout des fonctions enumerate() et sum() • Le type bool est maintenant vraiment distinct d'un entier • Beaucoup d'améliorations du support Unicode
2.4	30 novembre 2004	• Ajout des décorateurs de fonction/méthode (@decorateur) • Conversion automatique d'un entier court en entier long si le résultat d'une opération est trop grand • Expressions de générateur retournant les résultats l'un après l'autre et non pas sous forme d'une liste, exemple : sum(x for x in xrange(10000)) • Ajout des fonctions reversed() et sorted() • La fonction sort() accepte les mots clés cmp, key et reverse • Création du module decimal et du routeur
2.5	19 septembre 2006	• Ajout de l'opérateur **with** • Ajout des méthodes send(), throw() et close() aux générateurs • Expression conditionnelle (a if test else b) • Les imports de module peuvent être relatifs • Ajout des méthodes partition() et rpartition() aux chaînes str et unicode • Ajout des fonctions any() et all() • Intégration des bibliothèques ctypes, ElementTree, hashlib, sqlite3 et wsgiref
2.6	1er octobre 2008	• Nouvelle syntaxe de formatage de chaines de caractères • Classes de bases abstraites • Décorateurs de classes • Modules json, multiprocessing, contextmanager et fractions • Amélioration de la compatibilité avec Python 3

3.0	3 décembre 2008[27]	• Fusion des types 'int' et 'long' • Les chaînes sont en Unicode par défaut, 'bytes' remplace l'ancien type 'str' • Utilise des itérateurs plutôt que des listes là où c'est approprié (ex : dict.keys()) • a/b est la vraie division par défaut • exec et print deviennent des fonctions • None et as deviennent des mots clé • Le fichier __init__.py n'est plus nécessaire pour les modules Python • `x`, l'opérateur <>, et la méthode find() des chaînes disparaissent • De nombreuses fonctions disparaissent : apply(), buffer(), callable(), ... • map() et filter() disparaissent au profit des compréhensions de liste • reduce() disparaît au profit des boucles explicites *Voir la PEP 3100* [28] *pour les détails*

Développement

Les PEP

Les Proposition d'Amélioration de Python (ou "PEP", *Python Enhancement Proposal*) sont des documents textuels qui ont pour objet d'être la voie d'amélioration de Python et de précéder à toutes ses modifications ultérieures. Un pep est une proposition d'orientation pour le développement (*process PEP*), une proposition technique (Standard Track PEP) ou une simple recommandation (*Informational PEP*, la plus célèbre étant certainement celle de Tim Peters : http://www.python.org/dev/peps/pep-0020/) . À leur sortie, les PEPs sont relus et commentés par le BDFL[29] .

Python 3000

Une nouvelle version de Python, appelée Python 3.0 (le projet était appelé "Python 3000" ou "Py3K") abolit la compatibilité descendante avec la série des versions 2.x, dans le but d'éliminer les faiblesses actuelles du langage. La ligne de conduite du projet était de "réduire la redondance dans le fonctionnement de python par la suppression des méthodes obsolètes". Python 3.0a1, la première version alpha, a été publiée le 31 août 2007[30] , et il existe un PEP[31] qui détaille les changements prévus.

Philosophie

Python 3.0 a été développé avec la même philosophie que dans ses versions antérieures, donc toute référence à la philosophie de Python s'appliquera aussi bien à la version 3.0. Comme toujours, Python a accumulé beaucoup de nouvelles méthodes qui font en fait acte de redondance avec d'autres préexistantes. Python 3.0 , en recherchant la suppression du code redondant et des modules semblables, suit la grande directive philosophique de Python "Il ne devrait subsister qu'une seule méthode, qui soit à la fois optimale **et** naturelle pour chaque chose".

En dépit de cela, Python 3.0 restera un langage multi-paradigme. Les programmeurs auront encore le choix entre l'orientation objet, la programmation structurée, la programmation fonctionnelle, et l'aspect orienté-objet, et d'autres paradigmes ; en dépit du choix existant, Python 3.0 a cependant pour but d'être utilisé de manière plus naturelle que dans les versions 2.x.

Planning et compatibilité

Python 3.0a1, la première version alpha de Python 3.0, a été publiée le 31 août 2007. Les version 2.x et 3.x de Python seront publiées en parallèle pendant plusieurs cycles de développement, pendant lesquels la série des 2.x subsistera principalement pour la compatibilité, en incluant quelques caractéristiques importées depuis Python 2.3. Le PEP 3000 [32] contient plus d'informations à propos du processus de publication d'une version.

Comme Perl 6, Python 3.0 rompt la compatibilité descendante (rétro-compatibilité). L'utilisation de code écrit pour les séries 2.x n'est pas garanti avec Python 3.0. Ce dernier apporte des changements fondamentaux, comme le passage généralisé à l'Unicode pour les chaînes de caractères et une distinction forte entre les chaînes de caractère et les objets « bytes ». Le typage dynamique associé à certaines méthodes sur les objets de type dictionnaire font qu'une transition parfaite de Python 2.x vers Python 3.0 est très difficile. Comme toujours, un outil nommé "2to3" réalise la plus grande part du travail de traduction des version 2.x vers les versions 3.x, en indiquant les zones de codes sujettes à caution par des commentaires spéciaux et des Warnings. De plus, dans sa pré-version, 2to3 semble réussir franchement à réaliser une traduction correcte. [33] Dans le cadre d'une migration de python 2.x vers Python 3.x, le PEP 3000 [32] recommande de conserver le code original comme base des modifications et de le *traduire* pour la plateforme 3.x en utilisant 2to3.

Python 2.6 devra fournir des caractéristiques de compatibilité ascendante, aussi bien qu'un mode "Warnings" qui devrait faire prendre conscience des problèmes potentiels de transition pour le passage à Python 3.0. (voir PEP 361 [34] pour plus d'informations.)

Voir aussi

Liste de frameworks principaux

- Bibliothèques scientifiques:
  - Calcul : numpy [35],SciPy, PyIMSL [36], Sympy [37],SAGE [38]
  - Système expert :pyCLIPS [39], pyswip [40]
  - Visualisation : pydot [41], maplotlib [42], pyngl [43], MayaVi [44]
  - Datamining : Orange [45]
  - Simulation : simPy [46]
  - Chimie : pymol [47], Molecular Modelling Toolkit [48], Chimera [49], PyQuante [50]
  - Biologie : Biopython [51]
- Utilitaire : eGenix [52], ctype
  - Parser : PyParsing [53]
- Graphisme: Pygame, pil [54], Soya 3D, Vpython [55], (http://www.pymedia.org/pymedia]
- Framework web : Django, Karrigell, webware, Grok, Turbogears
- serveur web, CMS : Plone, zope, Google App Engine,CherryPy
- cartographie : TileCache, FeatureServer, Cartoweb 4, Shapely, GeoDjango, PCL
- Protocoles :
  - Serialisation : simplejason
  - TCP, UDP, SSL/TLS, multicast, Unix sockets, HTTP, NNTP, IMAP, SSH, IRC, FTP :twisted [56]
- vecteur : Geojson, OWSLib, Quadtree, Rtree, Shapely, WorldMill, ZCO
- ORM : SQLObject, SQLAlchemy, Django ORM
- driver sgbd : PyGresQL, Psycopg, MySQL

Articles connexes

- Association Francophone Python
- Python Software Foundation License
- **(en)** IPython
- Implémentation : Jython, CPython, IronPython, PyPy
- application : Mailman, lilypond, tini ERP, MoinMoin, Skencil, PySol, PyGeo, IpyKit
- IDE : IDLE, Eric, komodo, Democracy, Chandler, TreeLine, Netbeans [57], Pyscripter [58], Boaconstructor [59]

Liens externes

- **(en)** Site officiel [60]
- **(fr)** çais/Informatique/Programmation/Langages/Python/ Catégorie Python [61] de l'annuaire dmoz
- **(en)** Tutoriel de Guido van Rossum [62]
- **(fr)** Tutoriel Plongez au coeur de Python [63]
- **(fr)** Tutoriel Apprendre à programmer avec Python [64]

Références

[1] http://www.python.org/
[2] Python License (http://www.python.org/psf/license/)
[3] **(en)** « il faut treize paragraphes pour expliquer un "hello world!" en C++, en python il n'en faut que 2 » (http://www.openbookproject.net//thinkCSpy/preface.xhtml#auto2)
[4] *FAQ Python 1.2 Why was Python created in the first place?* (http://www.python.org/doc/faq/general/#why-was-python-created-in-the-first-place)
[5] *Introduction to MacPython* (http://homepages.cwi.nl/~jack/macpython/intro.html)
[6] Introduction de la 1er édition du livre "Programming Python" de Mark Lutz (http://www.python.org/doc/essays/foreword/), Guido van Rossum 1996
[7] selon le fichier HISTORY (http://svn.python.org/projects/python/trunk/Misc/HISTORY) mais la plus ancienne versions accessibles dans les archives du forum est la 0.9.1 (http://groups.google.com/group/alt.sources/browse_thread/thread/3b6652abb0e23b03/53ee3620bc53c281)
[8] The Grail Development Team (http://grail.sourceforge.net/info/team.html)
[9] *Grail -- The Browser For The Rest Of Us* (http://grail.sourceforge.net/info/papers/restofus.html)
[10] 28.1 rexec - Restricted execution framework (http://docs.python.org/lib/module-rexec.html)
[11] http://grail.sourceforge.net/info/diagram.gif
[12] Grail Home Page (http://grail.sourceforge.net/)
[13] Computer Programming for Everybody (http://www.python.org/doc/essays/cp4e)
[14] Python warts (http://mail.python.org/pipermail/python-list/1999-December/018678.html)
[15] Python Warts (http://web.archive.org/web/20031002184114/www.amk.ca/python/writing/warts.html)
[16] Unifying types and classes in Python 2.2 (http://www.python.org/download/releases/2.2.3/descrintro/)
[17] **(en)** Site officiel de PyInstaller (http://pyinstaller.python-hosting.com/)
[18] 2.4. Tout est objet (http://diveintopython.adrahon.org/getting_to_know_python/everything_is_an_object.html)
[19] *Python Tutorial* (http://docs.python.org/tut/node11.html) chapitre 9
[20] Why must 'self' be used explicitly in method definitions and calls? (http://effbot.org/pyfaq/why-must-self-be-used-explicitly-in-method-definitions-and-calls.htm)
[21] Python For AppleScripters (http://www.mactech.com/articles/mactech/Vol.20/20.11/PythonForAppleScripters/index.html)
[22] A Monotonic Superclass Linearization for Dylan (http://192.220.96.201/dylan/linearization-oopsla96.html)
[23] The Python 2.3 Method Resolution Order (http://www.python.org/download/releases/2.3/mro/)
[24] Quotes about Python (http://www.python.org/about/quotes/)
[25] *Python Creator Guido van Rossum now working at Google* (http://www.oreillynet.com/onlamp/blog/2005/12/python_creator_guido_van_rossu.html), article sur *ONLamp.com*

[26] http://www.voidspace.org.uk/python/movpy/
[27] Python 3.0 http://www.python.org/download/releases/3.0/
[28] http://www.python.org/dev/peps/pep-3100/
[29] Parade of the PEPs (http://www.python.org/doc/essays/pepparade.html)
[30] Python 3.0a3 Release (http://python.org/download/releases/3.0/)
[31] PEP 3000 - Python 3000 (http://www.python.org/peps/pep-3000.html)
[32] http://www.python.org/dev/peps/pep-3000/
[33] Sam Ruby, 2to3 (http://intertwingly.net/blog/2007/09/01/2to3), 1er septembre 2007
[34] http://www.python.org/dev/peps/pep-0361/
[35] http://numpy.scipy.org/
[36] http://www.vni.com/products/imsl/pyimsl/overview.php
[37] http://docs.sympy.org/
[38] http://www.sagemath.org/
[39] http://pyclips.sourceforge.net/
[40] http://code.google.com/p/pyswip/
[41] http://dkbza.org/pydot.html
[42] http://matplotlib.sourceforge.net/
[43] http://www.pyngl.ucar.edu/index.shtml
[44] http://mayavi.sourceforge.net/
[45] http://www.ailab.si/
[46] http://simpy.sourceforge.net/
[47] http://www.pymol.org/
[48] http://dirac.cnrs-orleans.fr/MMTK/
[49] http://www.cgl.ucsf.edu/chimera/
[50] http://pyquante.sourceforge.net/
[51] http://biopython.org/wiki/Documentation#Documentation
[52] http://www.egenix.com/products/python/
[53] http://pyparsing.wikispaces.com/
[54] http://www.pythonware.com/products/pil/
[55] http://www.vpython.org/
[56] http://twistedmatrix.com/trac/
[57] http://www.netbeans.org/features/python/
[58] http://www.mmm-experts.com/Products.aspx?ProductId=4
[59] http://boa-constructor.sourceforge.net/
[60] http://www.python.org
[61] http://www.dmoz.org/World/Fran
[62] http://docs.python.org/tut/
[63] http://diveintopython.adrahon.org/toc/index.html
[64] http://inforef.be/swi/python.htm

Sources et contributeurs de l'article

Python (langage) *Source*: http://fr.wikipedia.org/w/index.php?oldid=42801812 *Contributeurs*: A3 nm, Akeron, Al Maghi, Alcalazar, Alno, Alvaro, Antoine, Apultier, Archibald, Argos42, Arm@nd, Arno., Arronax50, Asabengurtza, Ash Crow, Athymik, Aurelienc, Badmood, Bap, Bapti, Bearnaise, Beltegeuse, Bestter, Blidu, Bluestorm, Bobochan, Boly38, Buf, CRicky, Calo, Camico, Canarix, Cantons-de-l'Est, Ch dav, Charlotte m, Chikyu, Citare, Coyazuu, Creasy, Cyril Guilloud, Cyril guilloud, Cédric, Daddo, Darkoneko, Dewi78, Dpa787, Dr gonzo, Duloup, Edhral, Elapied, Elg, Eric.pommereau, Erkethan, Eternel curieux, Eusebius, FR, Fabien.morin, FabienSchwob, Fabrice747, Fero14041, Fpeters, François-Dominique, FvdP, Garulfounix, Gh, Gilles.L, Goa103, Grecha, Grondin, Gui, GôTô, Haypo, Hcanon, Hemmer, Hervée, IAlex, Ianare, Ifernyen, Inike, Interwiki884, Isaac Sanolnacov, J-Luc, JB, JackPotte, JazzBass, Jd, Jerome misc, Jerome66, JihemD, Karl1263, Kerflyn, Koko90, Koyuki, Kuri2006, Kyle the hacker, Kyro, Le gorille, Leag, Legrocha, Levochik, Lgd, Litlok, Lmaltier, Looxix, Lstep, Mamelouk, Manu1400, Marchash, Matrixise, Med, Melkor73, Michel BUZE, Michelbailly, Mikayé, Mishkoba, MisterMatt, Moipaulochon, Morshwan, Mota, Nataraja, NeMeSiS, Negon, Neitsa, Neustradamus, Nikai, Nodulation, Nojhan, OPi, Obi Yann, Ofol, Okiokiyuki, Orthogaffe, Oz, Pabix, Pamplelune, Pamputt, Panoramix, Pascal Boulerie, Pem, Phe, Phisto, PierreLalet, Piglop, Pio, Pleifrest, Plinn, Ploum's, Plyd, Pog, PolyPrograms, Poulos, Poulpy, Poweroid, Pulsar, Rapha222, Rifleman, Rohanec, Romain Ballais, Romanc19s, Ryo, Sanao, Sdouche, SebGR, Shawn, Sherbrooke, Shlublu, Ske, Skippy le Grand Gourou, Spack, Stanlekub, Surt Fafnir, Tavernier, Th. Thomas, Thralia, Tjunier, Toutoune25, Traroth, VIGNERON, Volapuk, Vonvon, WikiDreamer, Xerus, Xfigpower, Xtrochu, Yannick Laurent, Youssefsan, Yvobrie, Zelda, ZeroJanvier, Zetron, script de conversion, 424 modifications anonymes

Source des images, licences et contributeurs

Image:Python.svg *Source*: http://fr.wikipedia.org/w/index.php?title=Fichier:Python.svg *Licence*: Creative Commons Attribution-Sharealike 3.0 *Contributeurs*: The people from the Tango! project.

Image:Python batteries included.jpg *Source*: http://fr.wikipedia.org/w/index.php?title=Fichier:Python_batteries_included.jpg *Licence*: Attribution *Contributeurs*: Frank Stajano

Licence