

Python (langage)

 Pour les articles homonymes, voir [Python](#).

Python est un langage de programmation objet, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle 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^[3] et fonctionne sur la plupart des plates-formes informatiques, des supercalculateurs aux ordinateurs centraux^[4], de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS, et aussi 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 certains pédagogues qui y trouvent un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation^[5].



Guido van Rossum, créateur de Python, à la OSCON 2006.

1 Utilisation

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. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses, comme 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 (voir la section [Adoption](#)). 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 au calcul numérique.

2 Historique

2.1 Au CWI

À la fin des années 1980, le programmeur Guido van Rossum participe au développement du langage de programmation ABC au Centrum voor Wiskunde en Informatica (CWI) d'Amsterdam, aux Pays-Bas. Il travaillait alors dans l'équipe du système d'exploitation Amoeba (en) dont les appels systèmes étaient difficilement interfaçables avec le Bourne shell utilisé comme interface utilisateur. Il estime alors qu'un langage de script inspiré d'ABC pourrait être intéressant comme interpréteur de commandes pour Amoeba^[6].

En 1989, profitant d'une semaine de vacances durant les fêtes de Noël, il utilise son ordinateur personnel^[7] pour écrire la première version du langage. Fan de la série télévisée 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^[8].

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^[9], est postée sur le forum `Usenet alt.sources`. La dernière version sortie au CWI fut Python 1.2.

2.2 Au CNRI

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

À partir d'août 1995, l'équipe Python travaille au CNRI sur *Grail*^[10] un navigateur web 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^[11]. Il a entraîné le développement de modules pour la bibliothèque standard comme *rexec*^[12], *htmlib* ou *urllib*^[13]. La version 0.6 sera la dernière de *Grail* ; elle est publiée en avril 1999^[14].

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. Cependant, du fait du manque de financement du projet par la DARPA, et du départ de nombreux développeurs Python du CNRI (dont Guido van Rossum), le projet s'éteint en 2000^[15]. Python 1.6 fut la dernière version sortie au CNRI.

2.3 À BeOpen

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 (à présent connue sous le nom de Zope Corporation)^[16].

Andrew M. Kuchling a publié en décembre 1999^[17] un texte nommé *Python Warts*^[18] 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^[19].

2.4 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 spécification 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, modélisée d'après l'**Apache Software Foundation**.

Afin de réparer certains défauts du langage (par exemple l'**orienté objet** avec deux types de classes), et pour nettoyer la bibliothèque standard de ses éléments obsolètes et redondants, Python a choisi de casser la compatibilité ascendante dans la nouvelle version majeure, Python 3.0, publié en décembre 2008. Cette version a été suivie rapidement par une version 3.1 qui corrige les erreurs de jeunesse de la version 3.0.

3 Caractéristiques

3.1 Syntaxe

Python a été conçu pour être un langage lisible. Il vise à être visuellement épuré. Par exemple, il possède moins de constructions syntaxiques que de nombreux langages structurés tels que C, Perl, ou Pascal. Les commentaires sont indiqués par le caractère **croisillon** (#).

Les blocs sont identifiés par l'**indentation**, au lieu d'accolades comme en C ou C++ ; ou de `begin ... end` comme en Pascal ou Ruby. Une augmentation de l'indentation marque le début d'un bloc, et une réduction de l'indentation marque la fin du bloc courant.

NB : l'indentation pourrait être modifiée ou supprimée dans la version en C sans modifier son comportement. De même la fonction Python peut être écrite avec une expression conditionnelle. Cependant, une indentation correcte permet de détecter plus aisément des erreurs en cas d'imbrication de plusieurs blocs et facilite donc l'élimination de ces erreurs. C'est pourquoi il est préférable d'indenter convenablement les programmes en C.

3.1.1 Mots-clés du langage

Les mots-clés sont fournis dans la liste `keyword.kwlist` du module `keyword`^[20]. Les mots-clés de Python 2.7.5 sont les suivants : `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `exec`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `not`, `or`, `pass`, `print`, `raise`, `return`, `try`, `while`, `with`, `yield`.

À partir de Python 3.0, `print` et `exec` ne sont plus des mots-clés du langage, mais des fonctions du module `builtins`^[21]. Sont ajoutés aux mots-clés : `True`, `False`, `None` et `nonlocal`. Les trois premiers étaient déjà présents dans les versions précédentes, mais ils ne sont plus modifiables (auparavant, l'affectation `True = 1` était possible)^[22]. `nonlocal` a été introduit par le PEP 3104^[23], et permet, dans une fonction définie à l'intérieur d'une autre fonction, de modifier une variable d'un niveau supérieur de portée. Avant cela, seules les variables locales à la fonction, et globales (niveau module) étaient modifiables.

3.1.2 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 illimité. Avant la version 3.0, ce type était dénommé `long`, et le type `int` correspondait à un entier 32 ou 64 bits. Néanmoins, une conversion automatique évitait tout débordement.
 - `float` est un flottant équivalent au type double du C, soit un nombre entre $-1,7 \times 10^{308}$ et $1,7 \times 10^{308}$ sur les plateformes en conformité avec l'IEEE 754.
 - `complex` est une approximation d'un **nombre complexe** (typiquement deux `float`).
- Les objets « itérables »
 - Les objets `tuple` (**n-uplet**) sont des listes **immuables** d'objets hétérogènes.
 - 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 `frozenset` forment une variante immuable des `set`.
 - 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. À partir de la version 3.0, les caractères sont en **Unicode** sur 16 ou 32 bits ; les chaînes d'octets sont des objets `bytes`^[24]. Dans les versions précédentes, ces objets étaient respectivement de type `unicode` et `str`. Les objets `str` et `bytes` sont **immuables**.
 - Les objets `bytearray` sont des chaînes d'octets modifiables. La version d'Unicode employée par Python peut être déterminée à l'aide de la variable `unicdata_version` du module `unicodedata`.

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

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

Pour une chaîne de caractères, l'itération procède caractère par caractère.

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 utilisant le protocole d'itération du langage.

3.1.3 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 :

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

la liste des nombres **pairs** :

```
liste = [entier for entier in range(10) if entier % 2 == 0]
# liste = [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, elles peuvent donc être passées en arguments à d'autres fonctions, et appelées lorsque nécessaire. En effet, les fonctions définies avec **def** sont équivalentes à celles définies avec **lambda**, il est d'ailleurs possible de **définir** une fonction à l'intérieur d'une autre fonction et ainsi obtenir une définition de fonction dans une variable locale, exemple :

```
def filtre_inferieur_a_5(une_liste) :
    def mon_filtre(x) :
        # variable locale mon_filtre
        return x < 5
    return filter(mon_filtre, une_liste)
```

3.1.4 Programmation objet

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-clé `class`. Les classes Python supportent l'héritage multiple ; il n'y a pas de surcharge statique comme en C++, ou de restrictions sur l'héritage comme c'est le cas en Java (une classe implémente plusieurs interfaces et hérite d'une seule classe) mais le mécanisme des arguments optionnels et par mot-

clé 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, niveau, nom, prenom) :
        Personne.__init__(self, nom, prenom)
        self.niveau = niveau
    def presenter(self) :
        return self.niveau + " " + Personne.presenter(self)
e = Etudiant("Licence INFO", "Dupontel", "Albert")
assert e.nom == "Dupontel"
```

3.1.5 Méthodes spéciales et définition des opérateurs

Python fournit un mécanisme élégant et orienté objet pour définir un ensemble pré-défini d'opérateurs : tout objet Python peut se voir doté de méthodes dites spéciales.

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

Les possibilités sont nombreuses et sont décrites dans la documentation du langage^[25].

Par exemple on peut définir l'addition de deux vecteurs à deux 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)
```

3.1.6 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 `yield`, et à l'appel suivant, reprend son déroulement juste après le `yield`. Par exemple pour calculer la suite de Fibonacci, on peut écrire :

```
def gen_fibonacci() :
    """Générateur de la suite de Fibonacci"""
    a, b = 0, 1
    while True :
        yield a # Renvoie la valeur de "a", résultat de l'itération en cours
        a, b = b, a + b
fi = gen_fibonacci()
for i in range(20) :
    print(next(fi))
```

Le module `itertools` permet de manipuler les générateurs. Par exemple, pour extraire les 10 premiers éléments du générateur précédent :

```
import itertools
list(itertools.islice(gen_fibonacci(), 10))
# renvoie [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Depuis Python 3.3, il est possible de produire un générateur à partir d'une fonction récursive, grâce à la syntaxe `yield from`, apparue dans le PEP 380^[26] et qui « délègue » le calcul à un sous-générateur. L'exemple suivant calcule les permutations des dames correspondant aux solutions du problème des huit dames étendu à un échiquier de taille $n \times n$.

```
def queens(n) :
    a = list(range(n))
    up = [True] * (2 * n - 1)
    down = [True] * (2 * n - 1)
    def sub(i) :
        nonlocal a, up, down
        for k in range(i, n) :
            j = a[k]
            p = i + j
            q = i - j + n - 1
            if up[p] and down[q] :
                if i == n - 1 :
                    yield tuple(a)
                else :
                    up[p] = down[q] = False
                    a[i], a[k] = a[k], a[i]
                    yield from sub(i + 1)
            up[p] = down[q] = True
            a[i], a[k] = a[k], a[i]
        yield from sub(0)
    sum(1 for a in queens(8))
# Nombre de solutions, renvoie 92
```

Un générateur peut sembler identique à une fonction qui retourne une liste, mais contrairement à une liste qui contient *tous* ses éléments, un générateur calcule ses éléments *un par un*.

Ainsi, le test `36 in [n * n for n in range(10)]` va s'effectuer sur la liste calculée en entier, alors que dans `36 in (n * n for n in range(10))`, qui utilise un générateur, le calcul des carrés s'arrête dès que 36 est trouvé.

3.2 Réflexivité

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

3.3 Typage

Le **typage** n'est pas vérifié à la compilation. Python utilise le **duck typing** : lors de l'exécution, si une méthode invoquée sur un objet a la même **signature** qu'une méthode déclarée sur cet objet, alors c'est cette dernière méthode qui est exécutée. De ce fait, invoquer une méthode qui n'existe pas sur un objet va 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 (entier), sa valeur est arrondie à l'unité inférieure (ici 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*^[27] ou au **patron de conception** *decorators*.

3.3.1 Compilation

Il est possible d'effectuer une analyse statique des modules Python avec des outils comme Pylint^[28] ou PyChecker. Sans nécessiter une exécution, ces outils repèrent des fautes ou des constructions déconseillées. Par exemple, une classe qui hérite d'une classe abstraite et qui ne redéfinit 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__`.

Il est aussi possible de générer un code intermédiaire (**bytecode**) Python.

Des outils comme PyInstaller^[29] ou d'autres plus spécifiques comme `cx_Freeze` sous **Unix**, **Windows** et **macOS**, `py2app`^[30] sous **macOS** 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.

3.4 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.^[31]. 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*.

3.4.1 Méthodes

Le modèle objet de Python est inspiré de celui de Modula-3^[32]. 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 gêne 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^[33].

Python reconnaît trois types de méthodes :

- les méthodes d'instances, qui sont celles définies par défaut. Elles 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 définies. 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++.

3.4.2 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 bibliothèque 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.

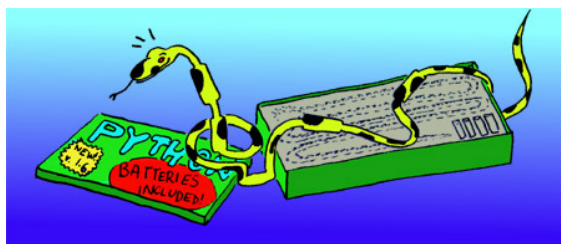
L'encapsulation est une problématique de développement logiciel. Le slogan des développeurs Python est *we're all consenting adults here*^[34] (nous sommes entre adultes consentants). Ils estiment en effet qu'il suffit d'indiquer, par des conventions d'écriture, les parties publiques des interfaces et 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 systématique d'accesseurs et le masquage des données comme il est courant de le faire en C++ par exemple.

3.4.3 Héritage

Python supporte l'héritage multiple. Depuis la version 2.3, il utilise l'algorithme C3 (en), issu du langage Dylan^[35], 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^[36].

4 Bibliothèque standard



Python est fourni « piles incluses ».

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

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 unitaires (unit-test, anciennement PyUnit avant version 2.1) pour créer des suites de tests exhaustives.

5 Conventions de style

Bien que chaque programmeur puisse adopter ses propres conventions pour l'écriture de code Python, Guido van Rossum a mis un guide à disposition, référencé comme « PEP 8 »^[37]. Publié en 2001, il est toujours maintenu pour l'adapter aux évolutions du langage. Google propose également un guide^[38].

6 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 et Pmw (Python megawidgets)^[39] pour Tk, wxPython pour wxWidgets, PyGTK pour GTK+, PyQt et PySide pour Qt, et enfin Fx-Py pour le FOX Toolkit. Il existe aussi une adaptation de la bibliothèque SDL : Pygame, un binding de la SFML : PySFML, ainsi qu'une bibliothèque écrite spécialement pour Python : Pyglet (en).

Il est aussi possible de créer des applications Silverlight en Python sur la plateforme IronPython.

7 La communauté Python

Guido 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 au sketch *Spam* des Monty Python, où deux clients tentent de commander un repas à l'aide d'une carte qui contient du jambon en conserve de marque SPAM dans pratiquement tous les plats. Ce sketch a été aussi pris pour référence pour désigner un courriel non sollicité.

7.1 Adoption de Python

Article détaillé : Liste de logiciels Python.

Plusieurs entreprises ou organismes mentionnent sur leur site officiel^[40] qu'ils utilisent Python :

- Google (Guido van Rossum a travaillé au sein de cette entreprise entre 2005 et 2012^[41]);
- Industrial Light & Magic ;
- la NASA ;
- et CCP Games, les créateurs du jeu vidéo EVE Online.

Python est aussi le langage de commande d'un grand nombre de logiciels libres :

- FreeCAD, logiciel de CAO 3D
- Blender, logiciel de modélisation 3D et d'édition vidéo
- Inkscape, logiciel de dessin vectoriel
- LibreOffice et Apache OpenOffice, les deux branches de développement d'une suite bureautique issue de StarOffice
- Portage, le gestionnaire de paquets du système d'exploitation Gentoo
- ParaView, logiciel de visualisation de données numériques
- Kodi, un lecteur multimédia
- QGIS, un logiciel de cartographie
- Weblate, un outil de traduction

Et commerciaux :

- Wing IDE, environnement de développement intégré spécialisé sur Python, et écrit en Python
- Corel Paint Shop Pro, logiciel de traitement d'image et d'édition graphique
- capella, logiciel de notation musicale
- ArcGIS, un logiciel de cartographie^[42]
- Visual Studio^[43]

Python est utilisé comme langage de programmation dans l'enseignement élémentaire et supérieur, notamment en France^[44]. Depuis 2013, il y est enseigné, en même temps que Scilab, à tous les étudiants de classes préparatoires scientifiques dans le cadre du tronc commun (informatique pour tous). Auparavant, l'enseignement d'informatique était limité à une option en MP, et se faisait en langage Caml ou Pascal. Les premières épreuves de concours portant sur le langage Python sont celles de la session 2015^{[45],[46]}.

8 Implémentations du langage

Outre la version de référence, nommée CPython (car écrite en langage C), il existe d'autres systèmes mettant en œuvre 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 ;
- Brython, une implémentation de Python 3 pour les navigateurs web
- 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 ;
- Shed Skin (en)^[47], un compilateur d'un sous-ensemble de Python produisant du code en C++ ;
- Unladen Swallow (en)^[48], une version de CPython optimisée et basée sur LLVM, maintenant abandonnée (la dernière version remonte à octobre 2009).

Ces autres versions ne bénéficient pas forcément de la totalité de la bibliothèque de fonctions écrites en C pour la version de référence, ni des dernières évolutions du langage.

9 Les distributions

Différentes distributions sont disponibles, qui incluent parfois beaucoup de packages dédiés à un domaine donné :

- ActivePython^[49]
- Python(x,y)^[50] : une distribution Python à l'usage des scientifiques basée sur Qt et Eclipse
- Enthought Canopy^[51] : une autre distribution à usage scientifique
- Anaconda^[52] : une troisième distribution à usage scientifique
- Pyzo^[53] : « *Python to the people* », destinée à être facile d'utilisation

Ce ne sont pas des *implémentations* différentes du langage Python : elles sont basées sur CPython, mais sont livrées avec un certain nombre de bibliothèques préinstallées.

9.1 Historique des versions

10 Développement

10.1 Les PEP

Les propositions 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*).

À leur sortie, les PEP sont relus et commentés par le BDFL^[68].

10.2 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 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^[69], et il existe un PEP^[70] qui détaille les changements prévus, ainsi qu'une page pour orienter les programmeurs dans leur choix de Python 2 ou 3^[71].

10.3 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 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.

10.4 Planning et compatibilité

Python 3.0a1, la première version alpha de Python 3.0, a été publiée le 31 août 2007. Les versions 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 3.x. Le PEP 3000^[72] 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 garantie 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 versions 2.x vers les versions 3.x, en indiquant les zones de codes sujettes à caution par des commentaires spéciaux et des mises en garde. De plus, dans sa pré-version, 2to3 semble réussir franchement à réaliser une traduction correcte^[73]. Dans le cadre d'une migration de Python 2.x vers Python 3.x, le PEP 3000 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 « mise en garde » qui devrait faire prendre conscience des problèmes potentiels de transition pour le passage à Python 3.0^[74].

10.5 Python pour smartphones

Il existe des versions de Python adaptées pour **Android** et **iPhone** en version 2.5 ou 2.6. Disponible en **Jailbreak** d'iOS sur iOS grâce à “setup tools”, et sur Android grâce à **SL4A** qui donne même une possibilité de faire des petites interfaces graphiques grâce au module “android” et qui permet d'envoyer des SMS, d'allumer la caméra^[75], ou encore de faire vibrer le téléphone. Les quelques lignes suivantes montrent comment faire ça :

```
droid = android.Android() # client lié au serveur
local lancé par l'application SL4A # pour contrôler
un téléphone distant à l'adresse 192.168.0.5, avec
SL4A lancé sur le port 9887 # il suffit de faire : an-
droid.Android('192.168.0.5', 9887) droid.vibrate(2.5)
# fait vibrer le téléphone (local ou distant) pendant 2.5
secondes
```

Un portage de Python sur les terminaux **Blackberry** est sorti en juin 2012, pour le système **BlackBerry OS 10**^[76]. Une version allégée est sortie en septembre 2012, appelée « **BlackBerry-Tart** »^{[77],[78]}, en raison d'un jeu de mots en anglais : « *a “tart” is lighter-weight than a “pie”* », en référence à la traditionnelle « *apple pie* ». Elle est basée sur Python 3.2.2.

11 Notes et références

- [1] <https://www.python.org/downloads/release/python-360/>
- [2] <https://www.python.org/downloads/release/python-2713/>
- [3] (en) « Python License »
- [4] Download Python for Other Platforms
- [5] (en) « il faut treize paragraphes pour expliquer un *Hello, World!* en C++, seulement deux en Python »
- [6] *FAQ Python 1.2 Why was Python created in the first place ?*
- [7] *Introduction to MacPython*
- [8] Introduction de la première édition du livre *Programming Python* de Mark Lutz, Guido van Rossum 1996
- [9] Selon le fichier HISTORY mais la plus ancienne version accessible dans les archives du forum est la 0.9.1
- [10] The Grail Development Team
- [11] *Grail -- The Browser For The Rest Of Us*
- [12] 28.1 rexec - Restricted execution framework
- [13] <http://grail.sourceforge.net/info/diagram.gif>
- [14] Grail Home Page
- [15] Computer Programming for Everybody
- [16] (en) History of the software
- [17] Andrew M. Kuchling, *Python warts*, Python Mailing List, 22 décembre 1999
- [18] Python Warts
- [19] Unifying types and classes in Python 2.2
- [20] The Python Standard Library - 31.6. keyword — Testing for Python keywords
- [21] The Python Standard Library - 2. Built-in Functions
- [22] (en) « Python 3.0 - Core Language Changes »
- [23] (en) « PEP 3104 - Access to Names in Outer Scopes »
- [24] Les objets bytes ont été introduits par le PEP 358, voir aussi le PEP 3137
- [25] The Python Language Reference » 3. Data model » 3.3. Special method names
- [26] PEP 380 - Syntax for Delegating to a Subgenerator
- [27] traits - explicitly typed attributes for Python
- [28] <http://www.pylint.org>
- [29] (en) Site officiel de PyInstaller
- [30] <http://svn.pythonmac.org/py2app/py2app/trunk/doc/index.html#abstract>
- [31] Dive into Python 3 - *1.5 Everything Is An Object*
- [32] *Python Tutorial* chapitre 9
- [33] Why must 'self' be used explicitly in method definitions and calls ?
- [34] Python For AppleScripters
- [35] Kim Barrett, Bob Cassels, Paul Haahr, David A. Moon, Keith Playford et P. Tucker Withington, « A monotonic superclass linearization for Dylan », *ACM SIGPLAN Notices*, vol. 31, n° 10, octobre 1996, p. 69-82 (DOI 10.1145/236337.236343, lire en ligne)
- [36] The Python 2.3 Method Resolution Order
- [37] PEP 8 - Style Guide for Python Code
- [38] Google Python Style Guide
- [39] Pmw - Python megawidgets
- [40] Quotes about Python
- [41] *Python Creator Guido van Rossum now working at Google*, article sur *ONLamp.com*
- [42] Python for ArcGIS
- [43] Visual Studio 2015 - Getting Started with Python sur le MSDN
- [44] *Bulletin officiel spécial n° 3 du 30 mai 2013*, article sur <http://www.education.gouv.fr/>
- [45] Sujets du concours Mines-Ponts
- [46] Sujets du concours Centrale-Supélec

- [47] Shed Skin
- [48] Unladen Swallow
- [49] ActivePython
- [50] Python(x,y)
- [51] Enthought
- [52] Anaconda
- [53] Pyzo
- [54] (en) What's New in Python sur python.org
- [55] Python 3.0 <http://www.python.org/download/releases/3.0/>
- [56] Python 3.1 <http://www.python.org/download/releases/3.1/>
- [57] Python 3.2 <http://www.python.org/download/releases/3.2/>
- [58] Python 3.3 <http://www.python.org/download/releases/3.3.0/>
- [59] <http://www.python.org/dev/peps/pep-0380/>
- [60] « Python 3.4 »
- [61] « What's New In Python 3.4 — Python 3.4.5 documentation », sur docs.python.org (consulté le 2 février 2017)
- [62] « Python 3.5 »
- [63] « What's New In Python 3.5 — Python 3.5.3 documentation », sur docs.python.org (consulté le 2 février 2017)
- [64] Python 3.6.0
- [65] PEP 515 - Underscores in Numeric Literals
- [66] PEP 506 - Adding A Secrets Module To The Standard Library
- [67] PEP 498 - Literal String Interpolation
- [68] Parade of the PEPs
- [69] Python 3.0a3 Release
- [70] Python 3000
- [71] Should I use Python 2 or Python 3 for my development activity ?
- [72] Plan de développement PEP 3000
- [73] Sam Ruby, 2to3, 1^{er} septembre 2007
- [74] PEP 361
- [75] <http://code.google.com/p/android-scripting/>
- [76] BlackBerry-Py Project
- [77] blackberry-py sur Bitbucket
- [78] BlackBerry-Tart Preview

12 Voir aussi




12.1 Liste de frameworks principaux

- Bibliothèques scientifiques :
 - Calcul : NumPy, SciPy, Pandas, PyIMSL Studio, SymPy, SAGE
 - Système expert : pyCLIPS, pyswip, pyData-log, pyKE (« Python Knowledge Engine »)
 - Visualisation : pydot, Matplotlib, pyngl, MayaVi
 - Exploration de données : Orange
 - Simulation : simPy
 - Chimie : PyMOL, MMTK, Chimera, PyQuante
 - Biologie : Biopython
- Utilitaire : eGenix, ctype
- Analyseur syntaxique : PyParsing, PLY (« Python Lex & Yacc »), NLTK (« Natural Language Toolkit »)
- Générateur de documentation : Sphinx
- Graphisme : PIL / « Pillow » Soya 3D, Vpython, pymedia, NodeBox
- Jeux : Pygame, Panda3D
- Framework web : Django, Karrigell, webware, Grok (en), TurboGears, Pylons (en), Flask, Bottle
- Serveur web, CMS : Plone, Zope, Google App Engine, CherryPy, MoinMoin, django CMS
- Cartographie : TileCache, FeatureServer, Cartoweb 4, Shapely, GeoDjango, PCL
- Protocoles :
 - pyTango
 - Sérialisation : SimpleJSON
 - TCP, UDP, SSL/TLS, multicast, Unix sockets, HTTP, NNTP, IMAP, SSH, IRC, FTP : Twisted
- Vecteur : GeoJSON, OWSLib, Quadtree, Rtree, Shapely, WorldMill, ZCO
- ORM : SQLAlchemy, Storm, Django ORM
- Driver SGBD : PyGresQL, Psycopg, MySQL-python

12.2 Articles connexes

- [Python Software Foundation License](#)
- [Pip](#) : gestionnaire de paquets des paquets Python.
- [Cython](#) : langage permettant d'écrire des modules compilables pour Python.
- [IPython](#) : terminal interactif.
- [RUR-PLE](#) : outil éducatif pour apprendre le Python de façon ludique en pilotant un robot virtuel.
- [PyPI](#) : dépôt tiers officiel.

12.3 Liens externes

- [\(en\) Site officiel](#)
- [\(fr\) Catégorie Python de l'annuaire DMOZ](#)
-  [Portail de l'informatique](#)
-  [Portail des logiciels libres](#)
-  [Portail de la programmation informatique](#)

13 Sources, contributeurs et licences du texte et de l'image

13.1 Texte

- **Python (langage)** *Source* : [https://fr.wikipedia.org/wiki/Python_\(langage\)?oldid=134688671](https://fr.wikipedia.org/wiki/Python_(langage)?oldid=134688671) *Contributeurs* : Youssefsan, FvdP, Athymik, Calo, Gui, Med, Buf, Ryo, Alvaro, Panoramix, Nataraja, Looxix, Fpeters, Jerome misc, Hemmer, Lstep, Popolon, Orthogaffe, Traroth, Aurelienc, Oz, Xtrochu, Ske, Ploum's, Rohanec, (:Julien :), Alno, Cdag, Maxlelubre, HasharBot, Tjunier~frwiki, Nojhan, Pulsar, Koyuki, Nikai, Jd, Robbot, Michel BUZE, ZeroJanvier, Haypo, Plinn, Yvobrie, Pem, Vncnt83, JackAttack, Archibald, Sanao, Phe, Goa103, Marc Mongenet, MedBot, Ifernyen, VIGNERON, AntoinePotrou, Phe-bot, FabienSchwob, Xerus, François-Dominique, JB, Domsau2, Cédric, L.Willms, Arno., Bap, The RedBurn, NeMeSiS, Duloup, Darkoneko, MisterMatt, Boly38, Ofol, Kyle the hacker, Poulpy, Gh~frwiki, Gasche, Hervée, Bestter, Leag, Bob08, Clemux, Xavier Combelle, Koko90, Volapuk, Pabix, Poulos, Sherbrooke, Arm@nd, Kerflynn, Xfig-power, Yannick Laurent, DocteurCosmos, PolyPrograms, Elg, Chobot, Peter17, Eusebius, GôTô, JihemD, Stanlekub, Romanc19s, StarObs, Lgd, Dereckson, Lmaltier, JazzBass, Inike, A3nm, Gzen92, Mota, Scs19fr, Matrixise, Plyd, Neustradamus, RobotQuistnix, Gpvosbot, FlaBot, Necrid Master, Mamelouk, Spack, Zorg724, Tavernier, Ash Crow, YurikBot, El pitareio, Gene.arboit, Negon, Zelda, Lt-wiki-bot~frwiki, Jerome66, Askywhale, Litlok, Toutoune25, PierreLalet, Bobochan, Pio~frwiki, Poweroid~frwiki, Le gorille, Canarix, KoS, Michelbailly, Fapae~frwiki, Shawn, Grecha, Amine Briki N, Interwiki884, Rapha222, Silex6, Omezo, Noar, Phisto, Blidu, Citare, Kuri2006, Elapied, SashatoBot, Edhral, Karl1263, Vonvon, Manu1400, Eric.LEWIN, Ch dav, Liquid-aim-bot, Asabengurtza, Zetron, Cyril Guilloud, Genium, Cyril guilloud, Grondin, Pleifrest, Neitsa, PieRROBoT, AzertyFab, Erkethan, Sdouche, Rifleman~frwiki, Fabrice747, Escalabot, Hcanon, Alcalazar~frwiki, Gibus, Morshwan, Thijs !bot, Legrocha, Romain Ballais, JnRouvignac, A2, Piglop, Creasy, Kyle the bot, Coyazuu, Gilles.L, Daddo, Thralia, RémiH, Dpa787, JAnDbot, J-Luc, Chatelot16, Ianare, Arkanosis, Mishkoba, IAlex, Xzander, Sebleouf, Bearnaise, OPi, CRicky, BetBot~frwiki, Zaver, Arronax50, Alexis Chazard, PouX, Erabot, FR, Shlublu, Pamplélune, McSly, Fabien.morin, Rei-bot, Aputier, Salebot, Bot-Schafter, Pamputt, Akeron, Surt Fafnir, Levochik, Tépabot, Beltegeuse, DodekBot~frwiki, Isaac Sanolnacov, Yf, AlnoktaBOT, Idioma-bot, Jonathan1, TXiKiBoT, Bapti, VolkovBot, Nodulation, Mikayé, Jeremiyah76, Chicobot, Melkor73, Marin M., Synthebot, Okiokiyuki, Pbotgourou, Ssx'z, Fero14041, Pascal Boulerie, SieBot, Llann, TcheBTchev, Kyro, Love Sun and Dreams, Jyr37, Ange Gabriel, Alecs.bot, Chikyuu, Vlaam, Hercule, PetitDej, Charlotte m, Conaclos, Th. Thomas, Bloody-libu, DumZiBoT, SniperMaské, Charlie Pinard, Raude, Eric.pommereau, Bastien Sens-Méyé, Garulfounix, Loudumo, Skippy le Grand Gourou, Dewi78, Obi Yann, PicMirandole, Mro, HerculeBot, WikiCleanerBot, WikiDreamer, SilvononBot, ZetudBot, Linedwell, François Dongier, Guillaume70, JackPotte, Herr Satz, Vis libre luron, Moipaulochon, Eternel curieux, TaBOT~zerem, Penjo, SebGR, ArthurBot, Cantons-de-l'Est, Gorkrane, Camilleroux, SassoBot, Xqbot, Eliovir, Marchash, Al Maghi, Rubinbot, Loreleil, JackBot, LucienBOT, Camico, Skull33, Spidermario, MastiBot, Coyote du 57, Lomita, TobeBot, RedBot, Ansero, TSeeker, Casotte, Absinthologue, Alexgtk19, Bobodu63, Nerville, Pylade, Oliparcol, EmausBot, HRoestBot, Renommé 20150211, Mastergreg82, Mchiazzaro, Pjacquot, Mentibot, WikitanvirBot, ChuispastonBot, Jules78120, Firwen, Chris16384, Driquet, Developpeur567, Hunsu, PierreQuentel, MerllwBot, Sodatus, Oblomov2, Bertol, Noelmace, Slippingspy, OrlodrimBot, Sergelucas, Elopash, Lydie Noria, Automatik, BonifaceFR, Nochnix, Metamorforme42, Coconox, Calinou1, Rome2, Dinopis, GhislainHivon, Naereen, Mo5ul, Mllx-lise, Arbautjc, Frolemageblanc, Addbot, Sahrayana, BerAnth, Snowflake Fairy, Jean Gre, Macadam1, Gradungula, Fustis, Dorfsmay, Do not follow, Sami1404, Nairwolf, HeyCat, Kiwipidae, Indetronable, Mywiz, Tartatindou03, Leo ponchio et Anonyme : 344

13.2 Images

- **Fichier:Blue_pencil.svg** *Source* : https://upload.wikimedia.org/wikipedia/commons/7/73/Blue_pencil.svg *Licence* : Public domain *Contributeurs* : File:Arbcom ru editing.svg by User:VasilievVV with color change by user:Jarekt *Artiste d'origine* : User:VasilievVV and user:Jarekt
- **Fichier:Crystal_mycomputer.png** *Source* : https://upload.wikimedia.org/wikipedia/commons/e/e3/Crystal_mycomputer.png *Licence* : LGPL *Contributeurs* : All Crystal icons were posted by the author as LGPL on kde-look *Artiste d'origine* : Everaldo Coelho (YellowIcon);
- **Fichier:Disambig_colour.svg** *Source* : https://upload.wikimedia.org/wikipedia/commons/3/3e/Disambig_colour.svg *Licence* : Public domain *Contributeurs* : Travail personnel *Artiste d'origine* : Bub's
- **Fichier:Guido_van_Rossum_OSCON_2006.jpg** *Source* : https://upload.wikimedia.org/wikipedia/commons/6/66/Guido_van_Rossum_OSCON_2006.jpg *Licence* : CC BY-SA 2.0 *Contributeurs* : 2006oscon_203.JPG *Artiste d'origine* : Doc Searls
- **Fichier:Info_Simple.svg** *Source* : https://upload.wikimedia.org/wikipedia/commons/3/38/Info_Simple.svg *Licence* : Public domain *Contributeurs* : Travail personnel *Artiste d'origine* : Amada44
- **Fichier:Nuvola_apps_emacs.png** *Source* : https://upload.wikimedia.org/wikipedia/commons/6/6a/Nuvola_apps_emacs.png *Licence* : LGPL *Contributeurs* : <http://icon-king.com> *Artiste d'origine* : David Vignoni
- **Fichier:Nuvola_apps_kcmsystem.svg** *Source* : https://upload.wikimedia.org/wikipedia/commons/7/7a/Nuvola_apps_kcmsystem.svg *Licence* : LGPL *Contributeurs* : Own work based on Image:Nuvola apps kcmsystem.png by Alphax originally from [1] *Artiste d'origine* : MesserWoland
- **Fichier:Python.png** *Source* : <https://upload.wikimedia.org/wikipedia/commons/e/e4/Python.png> *Licence* : CC0 *Contributeurs* : Travail personnel *Artiste d'origine* : Terzo
- **Fichier:Python_batteries_included.jpg** *Source* : https://upload.wikimedia.org/wikipedia/commons/6/68/Python_batteries_included.jpg *Licence* : Attribution *Contributeurs* : <https://www.python.org/images/batteries-included.jpg> *Artiste d'origine* : Frank Stajano
- **Fichier:Python_logo_and_wordmark.svg** *Source* : https://upload.wikimedia.org/wikipedia/commons/f/f8/Python_logo_and_wordmark.svg *Licence* : GPL *Contributeurs* : <https://www.python.org/community/logos/> *Artiste d'origine* : www.python.org

13.3 Licence du contenu

- Creative Commons Attribution-Share Alike 3.0