anybox

Anybox **est un centre de services Odoo (ex-OpenERP) et une société spécialisée dans le développement Python. Ce document vous aidera à découvrir ce langage et ses possibilités.**

Python : le développement logiciel productif et pérenne.

Tour d'horizon de la plateforme

Python est un langage et une plateforme de développement logiciel complète et généraliste, très facile d'accès et capable de se spécialiser de manière très pointue dans la majorité des domaines informatiques. Python est utilisé par un public très large : des développeurs web professionnels, des chercheurs en intelligence artificielle ou en bio-informatique, des administrateurs systèmes, ou même des programmeurs occasionnels. C'est le mélange de polyvalence et de facilité qui fait la force de Python. Avec un bref apprentissage et un minimum d'efforts, vous serez capable d'envisager n'importe quel type d'application de manière extrêmement efficace et de la terminer (ou de la faire terminer) en temps voulu.

Le développement de Python ayant commencé à la fin des années 1980, son déploiement dans l'industrie a commencé vers les années 2000. Aujourd'hui, Python est devenu très populaire auprès des développeurs : beaucoup de projets viennent peupler un écosystème déjà très riche, et ce dans tous les domaines. La plateforme bénéficie donc d'une visibilité croissante, qui s'accentuera encore dans les prochaines années.

À quoi ressemble la plateforme Python ?

- Un langage dynamique, interactif, interopérable et très lisible
- Un vaste ensemble de bibliothèques et *frameworks* spécialisés
- Des outils d'édition, de test et d'industrialisation
- Le support d'une communauté d'entreprises, d'individus et d'associations
- Un marché en forte croissance

Table des matières

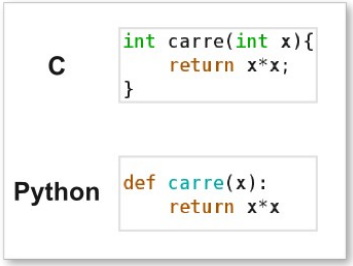
1. À quoi ressemble Python ?

1. Un langage facile et lisible

Python se caractérise par une **syntaxe claire et lisible**. Ce principe découle d'une constatation simple : un programme est lu bien plus souvent qu'il n'est écrit. Il faut donc que la compréhension du code à la lecture soit la plus rapide possible. Ceci prend même une importance considérable en milieu professionnel où un programme peut changer de mains plusieurs fois et doit être maintenu sur le long terme.

Cette lisibilité provient de plusieurs caractéristiques :

- Un très faible nombre de caractères exotiques : un programme Python est dépourvu de la plupart des caractères incompréhensibles que l'on rencontre traditionnellement dans les langages de programmation. Par exemple pour affecter le nombre 4 à la variable `a`, on n'écrit pas `$a := 4;` mais simplement `a = 4`
- Une indentation obligatoire : ce sont les espaces en début de lignes qui délimitent les blocs. Cette particularité, si elle effraye certains développeurs peu soigneux, se révèle en fin de compte être un atout énorme : un code Python est toujours indenté correctement et se lit toujours de la même manière. L'indentation a pour le programmeur la même utilité que les accolades pour le compilateur. En limitant la redondance, Python évite ainsi le risque d'incohérence entre ce qui est dit au compilateur et ce qui est dit au programmeur.



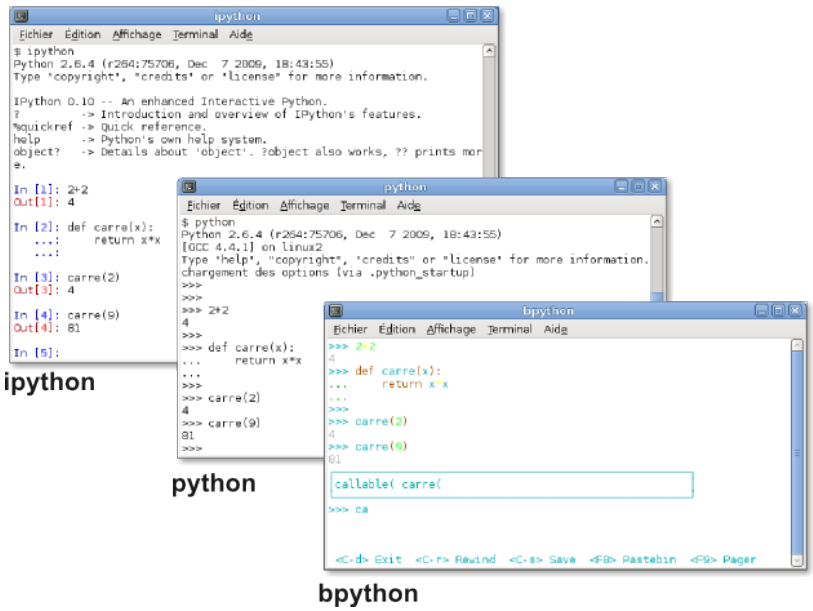
Pour ces raisons, le Python se rapproche parfois des pseudo-langages utilisés pour enseigner l'algorithmique. Il est idéal pour apprendre à programmer ou pour créer rapidement un prototype d'application. À l'autre extrémité, cette lisibilité devient un énorme avantage devant la complexité des très grands logiciels.

En pratique, un ensemble de conventions et de consignes simples a été proposé pour encadrer l'écriture de code Python. Ces règles sont définies dans un document portant le nom de « PEP 8 », qui tend à homogénéiser les programmes Python et à favoriser l'échange et la collaboration.

2. Un mode interactif et un débogueur intégré

Une autre particularité de Python est la présence d'un mode interactif : si on démarre Python sans lui donner de programme à exécuter, il donne la main à l'utilisateur, et exécute à la demande toute commande Python valide. Ceci est un atout pour la rapidité de développement, notamment lors d'un prototypage. De plus, dans ce mode il est possible de consulter l'aide des classes et des fonctions. Il est ainsi facile de faire des essais en mode interactif, puis de les recopier dans le corps du programme. Dans ce mode, Python peut aussi être utilisé comme une calculatrice programmable.

Dans l'illustration ci-dessous, l'interprète Python standard est celui du milieu. Les deux autres, *ipython* et *bpython* sont des variantes offrant des fonctionnalités additionnelles, comme la coloration, la complétion, les suggestions pendant la frappe, etc.



Nativement, le langage offre également un **débogueur**. N'importe quel programme peut ainsi être interrompu et exécuté instruction par instruction grâce à la pose de points d'arrêts ou à l'affichage de la pile d'appel. L'analyse et la correction de problèmes sont ainsi grandement facilitées : le débogueur prend l'apparence d'une console Python donnant accès à l'environnement d'exécution de l'application.

3.Multi-plateforme et interopérable

Python fonctionne sur toutes les plateformes les plus courantes, Windows, Linux et Mac Os, ainsi que sur de nombreux autres systèmes, y compris des plateformes mobiles telles que *Maemo* ou *Android*.

Python fonctionne également sur la machine virtuelle Java de Sun ou sur la plateforme .NET de Microsoft, donnant ainsi un accès direct à toutes les API Java ou .NET. Les versions correspondantes s'appellent Jython et IronPython.

Enfin, le module *ctypes* permet d'utiliser nativement n'importe quelle bibliothèque écrites en C. Python est dans ce cas idéal comme langage de glue.

4.Ouvert et libre

Les différentes versions de Python sont toutes publiées sous une licence libre permissive, autorisant l'écriture de logiciels libres ou propriétaires. Python peut être librement modifié, redistribué et même intégré à l'intérieur d'un autre logiciel pour lui offrir des capacités de *scripting*.

5.Moderne et Multi-paradigme

Au-delà d'une syntaxe très lisible, Python possède un typage fort mais dynamique, une compilation automatique en bytecode, un garbage collector, une gestion des exceptions, de l'Unicode, de la programmation multithread et multiprocess ainsi que d'autres caractéristiques qui en font un langage moderne et de plus en plus utilisé.

Python est un langage multiparadigme : il est possible de programmer en mode **impératif**, sans être un expert de la programmation objet. Si, en revanche, on programme avec des **objets**, on peut plonger sans difficulté dans tous les *motifs de conception* (*Design Patterns*), y compris en utilisant des interfaces ou des classes abstraites. La programmation **fonctionnelle**, enfin, peut être abordée grâce à l'importance accordée aux listes, aux itérateurs, aux générateurs ou grâce à la présence des fonctions `map`, `reduce`, `filter` et des fonctions anonymes `lambda`.

2. Que peut-on faire avec Python ?

Python est connu pour être « fourni avec les piles ». Cela signifie que sans aucun module additionnel, il est déjà possible de développer des applications pour de très nombreux domaines. La polyvalence est au cœur de la plateforme elle-même et constitue une de ses plus grandes forces : un développeur ayant appris le langage Python au travers d'un domaine (par exemple le développement de modules Odoo), sera capable d'aborder un autre domaine tel que les interfaces graphiques en un temps record, sans apprendre de nouveau langage. Le seul apprentissage nécessaire concerne certaines techniques supplémentaires, par exemple la programmation événementielle. Cet apprentissage sera fortement accéléré par l'absence de nouvelle difficulté syntaxique à assimiler. On pourrait nommer ceci la « réutilisation des compétences ». Ces considérations sont une des causes de ce que l'on appelle le « Paradoxe Python » : les développeurs Python ont semblé à une époque difficiles à recruter, mais former un développeur à Python est un investissement léger et rentable, car générateur de polyvalence, de motivation et d'efficacité. Un bon développeur Java ou PHP n'aura pas de mal à apprendre le langage en quelques heures et sera rapidement plus productif.

1. Services fournis en standard

Un langage de haut niveau comme Python se caractérise par le fait qu'il évite de réinventer ce qui existe déjà. Tout est déjà prévu pour que vous puissiez profiter de manière simple de protocoles courant, comme HTTP, FTP, IMAP ou POP, de techniques de programmation avancées comme le *multithreading* ou le *multiprocessing*, ou bien de chiffrement, compression ou stockage des données.

Voici un aperçu des possibilités fournies en standard avec Python.

Tous les types et services de base sont directement intégrés, comme les chaînes de caractères *unicode*, les types numériques, la gestion des erreurs (exceptions), la gestion des fichiers et des entrées/sorties, le formatage des chaînes et des dates. Des structures de haut niveau sont également présentes, comme les listes, les ensembles, les tableaux ou les dictionnaires. Ces structures sont fortement optimisées et leur implémentation est écrite en C. Elles apportent un haut niveau d'homogénéité et de souplesse dans la gestion des données.

De nombreux modules spécialisés sont présents par défaut, en voici quelques exemples :

re	expressions rationnelles
difflib	calculs de delta de différences
datetime	opérations sur les dates et heures
calendar	opérations sur des calendriers
math	opérations mathématiques
random	opérations aléatoires
zlib	opérations de compression
csv	lecture/écriture de fichiers .CSV
ConfigParser	lecture/écriture de fichiers .INI
sqlite3	accès à des bases de données SQLite
md5, sha	opérations de hachage
shutil	opérations de haut niveau sur les fichiers
io	opérations de bas niveau sur des flux de données
threading	programmation multithread
subprocess	création de sous-processus
multiprocessing	programmation multiprocessus

Quelques exemples de modules standards pour la programmation **réseau et internet** :

ssl	connexion réseau sécurisée
email	opérations sur des courriels
json	encodeur et décodeur JSON
webbrowser	gestion des navigateurs internet
cgi	outils pour la programmation web en CGI
httplib	programmation HTTP
ftplib	programmation FTP
poplib	réception d'e-mail par POP
imaplib	gestion d'e-mail par IMAP
smtplib	envoi d'e-mail (SMTP)
BaseHTTPServer	un serveur web basique
Cookie	gestion des cookies
xmlrpclib	connexion à des services web XML-RPC

Exemples de modules pour le **multimedia**, fournis aussi en standard :

wave	manipulation de fichiers audio WAV
colorsys	conversion de couleurs RGB, HSV, YIQ
imghdr	détection des types d'images
sndhdr	détection des types de fichiers audio

Exemples de modules utiles au **développement, débogage, profilage**, également fournis en standard :

pydoc	générateur automatique de documentation
doctest	écriture de la documentation testable
unittest	écriture des tests unitaires
pdb	le débogueur Python
profile	le profileur Python
gc	accès au Garbage Collector
inspect	outils d'introspection des objets Python

D'autres modules sont spécifiques aux plateformes UNIX, Windows ou Mac Os. Les modules ci-dessus ne sont que quelques exemples, vous pouvez vous référez à la documentation officielle pour avoir une vision plus complète des possibilités de la bibliothèque standard.

2. Accès aux bases de données

La plateforme Python permet d'accéder de manière très simple à la majorité des bases de données actuelles, y compris à des bases de données émergentes suivant la tendance actuelle du « NoSQL ». En dehors des pilotes bas niveau, on trouve également un ensemble de surcouches facilitant la programmation et améliorant fortement la sécurité (par exemple des protections contre les injections SQL).

Bases de données relationnelles

Il s'agit des bases de données classiques que l'on peut interroger avec le langage **SQL**. La majorité de ces bases de données, aussi bien libres que propriétaires, possèdent un pilote Python et ces pilotes suivent une spécification commune : la *DB-API*. La façon d'ouvrir un accès à une base et de lancer une requête est donc la même quelle que soit la base de données. Nul besoin d'apprendre à utiliser chaque pilote indépendamment. Ceci facilite également la migration d'une base vers une autre.

Parmi les bases de données ou les API prises en charge on trouve (liste non exhaustive) :

- PostgreSQL
- MySQL
- Oracle
- SQLite
- Ingres
- Informix
- Sybase
- Microsoft ADO
- IBM DB2
- ODBC
- Berkeley DB

Bases de données objet

Des bases de données **objets** sont également en production depuis plus de dix ans et simplifient la persistance des données dans le cadre d'un langage orienté objet comme Python. L'une d'entre elles, la ZODB, provient du *framework* web Zope, mais peut être utilisée de manière complètement indépendante pour permettre la persistance d'objets Python dans n'importe quelle application. L'utilisation de la ZODB rend la persistance complètement transparente dans une application : les objets sont stockés tels quels sous forme d'un graphe hiérarchique et chacun peut être utilisé directement comme un objet déjà en mémoire, sans avoir à effectuer la moindre requête. Une ZODB est capable de gérer des millions d'objets de manière fiable, transactionnelle, et historisée. Elle répond aux quatre propriétés *ACID* : Atomicité, Cohérence, Isolation et Durabilité.

Bases de données « NoSQL »

Il s'agit souvent de bases de données à très grands volumes de stockage, souvent **distribuées**. Un bon exemple de base de ce type est **CouchDB**, une base de données supportée par la fondation Apache, écrite en Erlang avec un système de vues en Javascript. Le principe de CouchDB est de stocker uniquement des documents non structurés, possédant chacun un ensemble non figé de propriétés.

Les documents sont stockés à plat, de manière non hiérarchique. L'équivalent des requêtes SQL se fait grâce à l'utilisation de « vues » pré-calculées pour l'ensemble des documents. Le dialogue avec CouchDB peut se faire avec n'importe quel langage, via HTTP et un échange de données en JSON. Plusieurs bibliothèques en Python, comme *CouchDBKit*, facilitent le dialogue, la création de bases, l'insertion ou la modification des vues.

MongoDB est un autre exemple de base orientée « document », et possédant un pilote Python.

La tendance NoSQL suit divers chemins : en dehors des bases orientées *document* comme *CouchDB* et *MongoDB*, on trouve des bases orientées « clé/valeur » comme *Redis* ou *MemcacheDb*, ou orientées « colonnes », comme **Cassandra** qui a été libérée par Facebook en 2008. Toutes les bases de ce type possèdent un pilote ou une bibliothèque Python : celle de Cassandra se nomme *Lazyboy* et simplifie l'accès aux données en s'intégrant aux structures de données haut niveau de Python. Celle de *Redis* se nomme simplement « redis », etc.

Surcouches pour l'accès aux bases

Python étant un langage objet, il est souvent plus agréable d'accéder aux bases de données en utilisant des objets. Au delà de la spécification *DB-API* offrant une interface unifiée, on trouve différents niveaux de surcouches aux bases de données : des bibliothèques comme **SQLAlchemy** proposent la construction de requêtes SQL par de simples appels de méthode de type *insert*, *filter* ou *join* sur des objets, sans avoir à écrire une seule ligne de SQL. Cela présente de nombreux avantages en terme de fiabilité, de sécurité et de possibilité de changement de base de données : le langage d'accès est unifié entre toutes les bases.

À un niveau encore supérieur, on trouve les « ORM » (Object Relational Mappers). SQLAlchemy contient aussi un excellent *ORM*, probablement le plus utilisé en Python. L'ORM permet de travailler directement sur les objets métiers sans se soucier de la base de données : lors d'une lecture dans la base, les objets Python sont automatiquement reconstruits par agrégation ou jointure de tables. Inversement, lorsqu'on modifie les attributs d'un objet, l'ORM sait dans quelles tables écrire. Cette connaissance de l'ORM provient de l'écriture des schémas de données métiers effectuée lors de la phase de conception : la gestion de la base de données n'a plus besoin d'être prise en charge par le développeur, hormis pour des questions d'optimisation. Heureusement pour ce dernier point, il est possible de connaître toutes les requêtes générées et d'agir même au plus bas niveau.

D'autres ORM existent, comme *Storm*, créé par Canonical, la société qui développe le système d'exploitation Ubuntu Linux.

3. Sites et applications web

Python est utilisé depuis plus d'une décennie pour des développements web professionnels. Certains sites web actuels à très fort traffic comme **YouTube** ou **Reddit** ont été initialement écrits en Python. Les offres d'hébergement mutualisé classiques à très faible coût proposent assez rarement ce langage car il nécessite l'utilisation d'un serveur dédié ou privé virtuel. Cependant, l'avènement des serveurs dédiés low-cost et du *Cloud Computing* s'accompagne de possibilités d'héberger des applications web en Python pour un coût modique. L'offre de *Cloud Computing* de Google, nommé *App Engine* a d'abord été disponible exclusivement pour Python (Google est un grand utilisateur de ce langage).

Des applications web professionnelles de type *CMS* (Gestion de contenu, intranets, extranets), *ERP* (Gestion d'entreprise) profitent également de la souplesse du langage et de la facilité avec laquelle on peut écrire des programmes génériques et extensibles.

Pour la gestion de contenu, les portails ou sites collaboratifs, les intranets ou extranets, l'outil phare est **Plone**. Ce *CMS* très mature est utilisé depuis près de quinze ans, il est complètement industrialisé, supporté par une importante communauté internationale et possède un très grand nombre de modules d'extension.

Pour la gestion d'entreprise, on trouve deux PGI Open Source (Progiciels de Gestion Intégrés) :

Odoo connaît une croissance fulgurante. Il fonctionne entièrement en mode web et possède un nombre impressionnant de modules métiers. C'est à la fois un ERP open-source très puissant, mais aussi une plateforme web permettant de construire très rapidement un site web, une boutique e-commerce en ligne, un blog ou d'autres applications web directement intégrées avec l'ERP. En outre, il se base sur un framework de développement rapide *OpenObject* avec lequel il est possible de développer n'importe quelle application d'entreprise en un temps record.

Un autre PGI majeur est **ERP5**. Basé sur les technologies Zope, celui-ci s'offre le luxe de concurrencer les meilleures solutions d'ERP propriétaires, et son caractère générique lui permet de s'adapter à n'importe quelle entreprise ou organisation de n'importe quel pays. Il constitue également la base du service *Tiolive* d'externalisation totale de l'infrastructure informatique d'une entreprise.

À titre d'exemple nous citerons quelques outils courants dans le milieu du web qui sont écrits en Python : **Mailman** est un gestionnaire de listes de diffusion très robuste et extrêmement utilisé dans la communauté du logiciel libre. **Moinmoin** est un moteur de wiki également très courant. Enfin, plus orienté réseau que web, l'implémentation historique du serveur et du client **Bittorent** sont en Python. Du fait de l'ouverture du protocole, de nombreuses autres versions en différents langages existent, mais il faut noter que ces premières versions sont celles qui l'ont mené à son succès originel.

Frameworks de développement web

L'intérêt d'un *framework* (ou « cadriciel » en français) est d'offrir de manière cohérente toutes les briques de base pour créer une application. Ainsi, les frameworks se chargent de vous offrir toutes les facilités indispensables : accès aux bases de données, génération de page HTML, génération de formulaires, sessions, cache, authentification, etc.

Le monde du développement web en Python est extrêmement riche. Il est possible d'attaquer tous les niveaux de complexité, depuis la simple page personnelle jusqu'à un gros ERP ou un intranet. Cette richesse est heureusement contrôlée par une spécification qui rend les composants web et les frameworks interopérables. (La spécification *WSGI*, *Web Server Gateway Interface*).

La partie émergée de cet énorme iceberg laisse apparaître trois frameworks majeurs : Zope, Django et Odoo. La partie immergée, tout autant digne d'intérêt, est abordée plus bas.

Zope est un écosystème très vaste et très riche, certainement le plus ancien. Le principe fondateur de Zope est la publication d'objets sur le web : dès la fin des années 90, ce *framework* était entièrement orienté objet, y compris sa base de données (ZODB). Zope est toujours très utilisé et a donné naissance à tout un écosystème, composé de *frameworks*, micro-*frameworks* modernes, bibliothèques, outils et applications comme Plone. Zope est le framework idéal pour créer des application de gestion de contenu ou gestion documentaire.

Django est un framework web généraliste de type *MVC*, très similaire à *Ruby on Rails* ou *Symfony*. Sa documentation est complète et sa prise en main rapide. La communauté française est très active et anime le site web *django-fr.org*. Django est idéal pour créer rapidement des sites ou applications web avec de bonnes performances. Il peut s'interfacer avec différents types de bases de données et prend nativement en charge *MySQL*, *PostgreSQL*, *SQLite* et *Oracle*. De très nombreux modules pour Django existent (qu'on appelle des « applications Django ») et peuvent se greffer pour offrir des fonctionnalités additionnelles. Outre son excellente documentation, un des intérêts historiques de Django est d'offrir une interface d'administration, créée automatiquement, pour votre application.

Odoo, si l'on fait abstraction de tous ses modules applicatifs, est aussi un framework de développement rapide. Il se base exclusivement sur la base de données PostgreSQL, gage de performance et de fiabilité. Outre sa modularité, son plus grand atout est d'être un framework beaucoup plus cadré que les deux précédents, faisant de lui le framework idéal pour le développement d'outils de gestion d'entreprise : l'interface graphique, la base de données, les workflows métiers étant définis de manière quasi-descriptive, on arrive très rapidement à obtenir un résultat attrayant et ergonomique, et on peut passer ainsi plus de temps sur les besoins métiers.

Les succès de Zope, Django et Odoo proviennent du fait qu'ils offrent, à l'instar de Python, un maximum de possibilités de manière native. Ces deux produits ne doivent pas occulter l'extraordinaire richesse d'autres *frameworks* comme **Pyramid**, **CubicWeb**, **web2py, CherryPy, Werkzeug, Flask**, **Anyblok**, etc. Vous pouvez obtenir une liste plus complète sur la page http://wiki.python.org/moin/WebFrameworks

Micro-frameworks et assemblage de composants web

Plutôt que d'utiliser des *frameworks* tout-en-un comme Zope, Django, ou Odoo, une alternative intéressante est de créer des applications par assemblage de composants génériques. Le principe est de partir d'une base minimale (comme *Pyramid* ou *Flask*), puis d'ajouter des briques. Cet assemblage peut se faire de différentes façons :

- par ajout et utilisation de bibliothèques additionnelles,
- grâce à WSGI, un standard Python permettant de brancher des middlewares
- grâce à la *Component Architecture* de Zope, utilisable avec n'importe quel projet Python
- avec d'autres techniques de composants existantes (voir *CubicWeb*)

Voici quelques exemples de fonctionnalités pouvant être prises en charge par des composants séparés et réutilisables :

Accès aux bases de données

SQLAlchemy est une bibliothèque d'abstraction permettant d'accéder à n'importe quelle base de données relationnelle de manière identique et en programmation objet. SQLAlchemy peut dialoguer avec SQLite, PostgreSQL, MySQL, Oracle, MS-SQL, Firebird, MaxDB, MS Access, Sybase, Informix, et DB2. D'autres outils du même genre existent : **Storm**, **SQLObject**. Nous vous invitons à consulter la section Bases de données en page 9.

Génération de formulaires

Le principe est de décrire les données grâce à un schéma puis de laisser les formulaires se générer automatiquement en fonction du schéma. Il existe souvent un lien entre formulaires, schémas et bases de données. Pour cette raison, les bibliothèques de génération de formulaires sont souvent liées à une autre technologie, soit le framework lui-même, soit la base de données ou une abstraction sur la base de données. Voici quelques exemples de bibliothèques matures utilisables en production : **z3c.form** (Zope), **FormAlchemy** (SQLAlchemy), **tw.forms** (TurboGears), **gnue-forms** (GNUe).

Authentification / autorisation

Il est possible de prendre en charge la notion d'authentification et d'autorisation de manière générique, hors de tout framework, juste par ajout d'un middleware WSGI sur une application existante. Les composants **repoze.who** et **repoze.what** sont conçus dans cette optique et couvrent les deux types de besoin. Ils fonctionnent grâce à des plugins et sont capables de gérer n'importe quelle source d'utilisateurs, de groupes, de permissions, et n'importe quelle méthode d'authentification et d'autorisation.

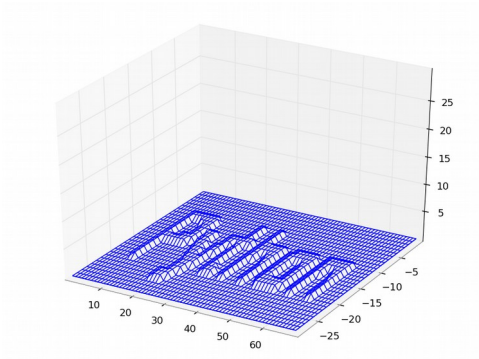
Templating

La création dynamique des pages HTML est prise en charge par ce qu'on appelle un outil de *templating*. Le principe est d'insérer du contenu en plaçant dans les pages HTML soit des balises spécifiques, soit des attributs XML. Tous les frameworks de développement web proposent un ou plusieurs outils de ce genre (dont les noms sont ZPT, Mako, Genshi, Chameleon, et d'autres).

Gestion du cache et des sessions

Beaker est une bibliothèque permettant de gérer du cache ou des sessions avec abstraction du stockage. De nombreuses méthodes de stockage sont disponibles, persistantes ou non : memcached, dbm, sql, mémoire vive, etc.

4. Calcul et visualisation scientifique



Le domaine scientifique est un des plus gros points forts de Python. Sa facilité d'apprentissage permet à des scientifiques, chercheurs, mathématiciens d'être efficaces rapidement lors de prototypages, de calculs lourds ou distribués, de visualisation, d'apprentissage automatique.

Python remplace progressivement les outils ou *frameworks* scientifiques propriétaires et son modèle Open Source est un avantage dans le milieu scientifique où le partage de connaissance et d'innovation est une pratique naturelle.

NumPy

NumPy est l'outil de base pour faire du calcul scientifique en Python. Il offre notamment des capacités de calcul sur des matrices à N dimensions, des capacités d'intégration avec C/C++ et Fortran, de l'algèbre linéaire, des transformées de Fourier et des outils de calcul aléatoire. NumPy permet de travailler sur des très gros jeux de données en minimisant la consommation mémoire (calcul sur des sous-matrices, sans duplication). Les algorithmes de calcul sont implémentés en C et fortement optimisés.

$$\sqrt[3]{x} = 5$$

$$\frac{X}{\frac{X}{Y}}$$

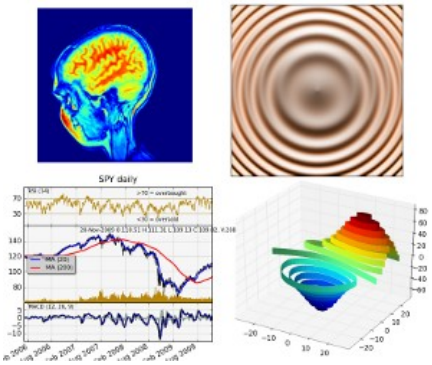
$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \frac{1}{8\pi 2}\int_{\alpha_2}^{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{U^{0\beta}_{\rho_1\sigma_2}}\right]$$

$$\mathcal{H} = \int d\tau (\epsilon E^2 + \mu H^2)$$

SciPy

SciPy est construit au dessus de NumPy et offre un vaste ensemble d'outils et d'algorithmes pour les mathématiques, le calcul scientifique et l'ingénierie : calculs matriciels, polynomiaux, algèbre linéaire, traitement du signal, statistiques, algorithmes génétiques, machine learning, etc.

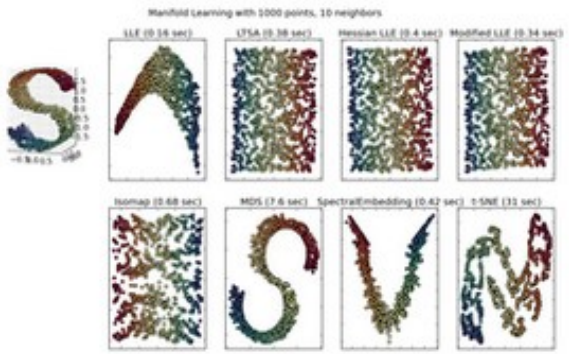
Matplotlib

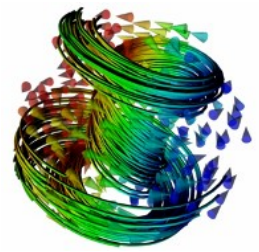


Matplotlib est une bibliothèque de tracé et visualisation produisant des graphiques de qualité professionnelle. Les graphiques peuvent être produits dans différents formats de sortie, y compris des formats interactifs permettant une interaction à la souris, intégrables dans une interface graphique. Matplotlib peut être utilisé soit depuis un programme en Python, soit depuis un terminal de commande interactif. De très nombreux types de graphiques peuvent être générés. Par défaut il s'agit de graphiques en 2D et des extensions permettent de produire des cartographies, graphiques en 3D ou des grilles de données.

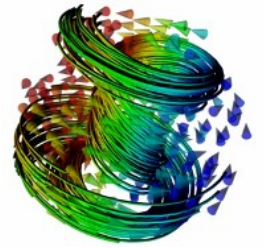
Scikit Learn

Construit à partir des trois outils précédents, Scikit Learn est un ensemble de bibliothèques et d'algorithmes dédiés à l'apprentissage automatique, au data mining et à l'analyse de données, c'est à dire aux problèmes de classification, régression, clustering, réduction dimensionnelle, sélection de modèle, extraction de caractéristiques et normalisation.





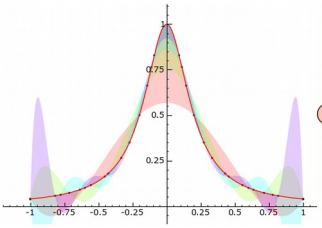
MAYAVI

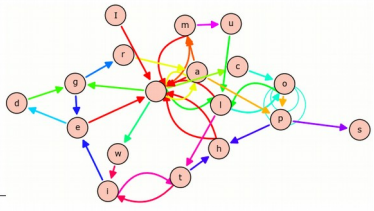


Outil de visualisation interactive de données scientifiques, prévu pour être intégré avec les différentes bibliothèques scientifiques Python (notamment Scipy).

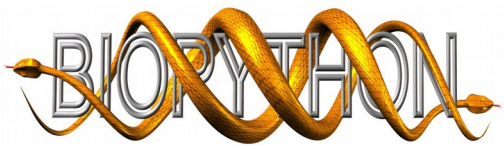
Sage

L'objectif de Sage est l'étude des mathématiques, élémentaires ou avancées, fondamentales ou appliquées. Cela comprend notamment l'algèbre basique, le calcul infinitésimal, la théorie des nombres, la cryptographie, le calcul numérique, l'algèbre commutative, la théorie des groupes, la combinatoire, la théorie des graphes, l'algèbre linéaire exacte et beaucoup d'autres domaines. Sage est dédié à l'enseignement et la recherche. Son principe est de rassembler plus d'une centaine de programmes Open Source dans une interface unifiée, soit une ligne de commande Python, soit un « notebook » accessible depuis un simple navigateur web.





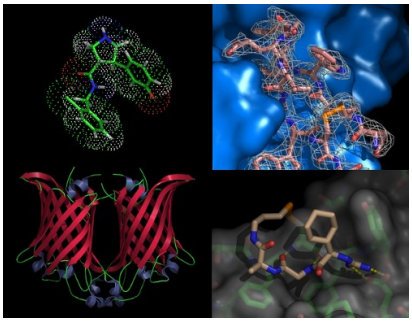
Biopython



C'est un framework orienté biologie. Il contient plusieurs modules spécialisés, pour travailler avec les séquences ADN, les protéines qu'elles codent et interagir avec les principales bases de données biologiques en ligne.

PyMol

PyMol est un système de visualisation moléculaire, un outil de rendu et un éditeur moléculaire en 3D, dédié à la visualisation de structures chimiques, y compris les structures cristallines à résolution atomique. PyMol peut être utilisé pour visualiser des protéines, des acides nucléiques (ADN, ARN) des glucides, ou n'importe quelle structure moléculaire ou atomique pour la recherche pharmacologique, biotechnologique, dans l'industrie, la recherche académique ou l'enseignement. PyMol permet de générer des images statiques ou des animations et peut exporter les données vers d'autres formats 3D.



Autres projets

De très nombreux autres projets existent, en voici quelques uns :

MMTK : Le *Molecular Modelling Toolkit*, permet la modélisation et la manipulation de molécules, grâce à des algorithmes de simulations permettant d'implémenter facilement des simulations complexes (trajectoires, codage de protéines étape par étape...).

SymPy : un outil pour les mathématiques symboliques.

Une liste plus complète en anglais se trouve à l'adresse suivante : http://wiki.python.org/moin/NumericAndScientific

5. Scripting d'applications

La légèreté de Python et ses capacités d'interopérabilité lui permettent d'être embarqué directement dans de nombreux logiciels afin d'être utilisé comme langage de script ou de macro. Quand il n'est pas embarqué, Python est souvent proposé comme méthode de scripting et de pilotage pour des applications de bureautique ou de graphisme.

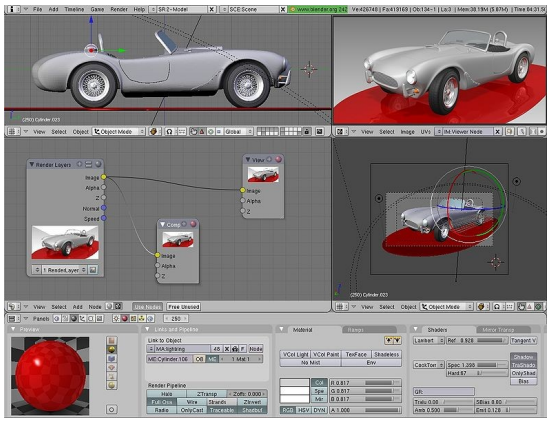
En voici quelques exemples.

LibreOffice

La célèbre suite bureautique libre peut être pilotée entièrement en Python. Il est donc possible d'écrire des macros en Python pour réaliser toutes sortes de tâches automatisée, de générer ou remplir des documents depuis une base de données, générer des factures, etc.

Autodesk Maya

Maya est un modeleur 3d propriétaire réputé, développé par Autodesk. Depuis la version 8.5, Maya offre nativement une interface Python aux commandes Maya et à l'API interne.



Blender

Blender est un logiciel de modelage et d'animation 3D libre et très puissant, entièrement scriptable en Python, ce qui permet d'automatiser des traitements, des créations d'objets ou d'animations complexes.

Inkscape

Inkscape est un illustrateur vectoriel libre, il est aussi interfaçable avec Python, de l'intérieur ou de l'extérieur (appel d'un script depuis Inkscape ou communication avec Inkscape depuis un script).

Gimp

Gimp est un logiciel libre de retouche d'images possédant de nombreuses fonctions avancées. Il inclut de base un interprète Python, permettant de le scripter et de réaliser des filtres très puissants.

Autres logiciels pilotables

De très nombreux autres logiciels sont pilotables nativement en Python. Pour les autres, sous Windows, sous Mac et sous Linux, des API ou des bus d'événements de type COM, D-Bus, AppleScript, sont également programmables en Python pour piloter des applications, des parties du système ou réagir à des événements.

6. Interfaces Graphiques

Python est fourni par défaut avec un module « TkInter » permettant de créer des interfaces graphiques simples. Pour les besoins plus avancés, il peut s'interfacer avec plusieurs bibliothèques graphiques présentées plus bas. Le code créé peut en outre fonctionner de la même manière sur Windows, Linux, MacOsX ou n'importe quel autre système sur lequel la bibliothèque utilisée est disponible ! Visuellement il est impossible de différencier une application écrite en Python et la même application écrite par exemple en C ou C++ : si la bibliothèque graphique est la même, le rendu est identique. Le code source, par contre, sera beaucoup plus facile à comprendre et à maintenir. L'utilisation de Python avec ces bibliothèques réduit les temps de développement de manière spectaculaire, sans pénaliser la réactivité des logiciels.

Bibliothèques disponibles en Python

Une bibliothèque graphique est le composant qui offre l'ensemble des widgets composant une interface graphique (boutons, checkbox, menus, ascenseurs, etc.). Plusieurs bibliothèques existent pour créer des interfaces graphiques multiplateformes, et toutes peuvent être utilisées avec Python. Les plus connues sont **GTK+**, **Qt** et **WxWidgets**.

GTK+ utilise automatiquement le thème natif de Windows et passe donc inaperçu sur ce système. Il est utilisé par défaut dans beaucoup de systèmes d'exploitations comme Ubuntu, Redhat ou Solaris, car l'environnement *Gnome* se base entièrement sur lui. De très nombreux logiciels l'utilisent, aussi bien libres, comme *GIMP* (retouche photo), *Inkscape* (illustrateur vectoriel) ou *Maemo* (plateforme embarquée), que propriétaires, comme *VMware* (virtualisation). L'accès en Python se fait grâce à **PyGtk**, disponible sous Windows, Linux et MacOsX.

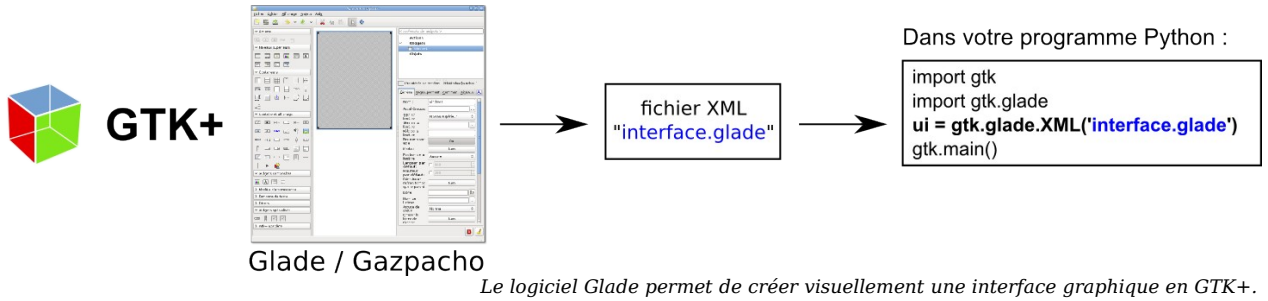
Qt est une bibliothèque développée aujourd'hui par Nokia et aussi très employée, aussi bien par des logiciels libres que des logiciels propriétaires. Elle est utilisée également par de nombreux systèmes d'exploitation comme Mandriva ou Pardus, car l'environnement KDE se base dessus. L'accès en Python se fait grâce à **PyQt**, disponible sous Windows, Linux et MacOsX.

Le principal intérêt de *WxWidgets* est qu'il utilise la bibliothèque graphique du système cible, rendant l'intégration 100% transparente. L'accès en Python se fait grâce à **WxPython**, disponible sous Windows, Linux et MacOsX.

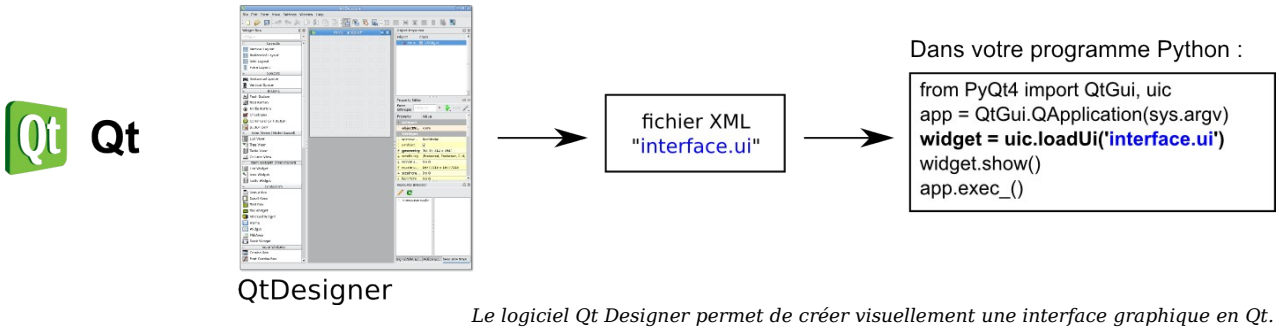
D'autres bibliothèques similaires existent, comme *Tk*, dont l'interface est même fournie par défaut avec Python.

Création rapide d'interfaces graphiques

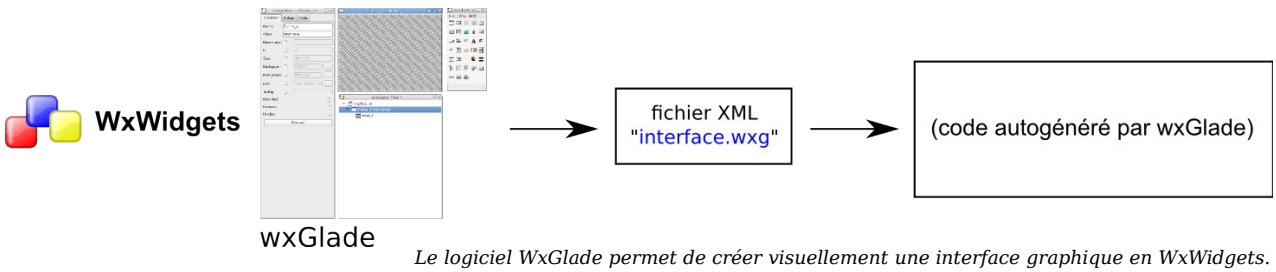
Pour créer rapidement des interfaces graphiques complexes, il existe des outils interactifs et très simples à utiliser. En voici quelques uns, dont certains sont écrits en Python. Leur fonctionnement est similaire : vous créez visuellement l'interface graphique, puis l'outil enregistre le description de cette interface dans un fichier XML. Ce fichier XML peut ensuite être utilisé grâce à une unique instruction dans le programme final en Python. Avec WxWidgets, étant donné que le code fait plus d'une ligne, il est généré automatiquement par l'outil *wxGlade*.



Le logiciel Glade permet de créer visuellement une interface graphique en GTK+.



Le logiciel Qt Designer permet de créer visuellement une interface graphique en Qt.



Le logiciel WxGlade permet de créer visuellement une interface graphique en WxWidgets.

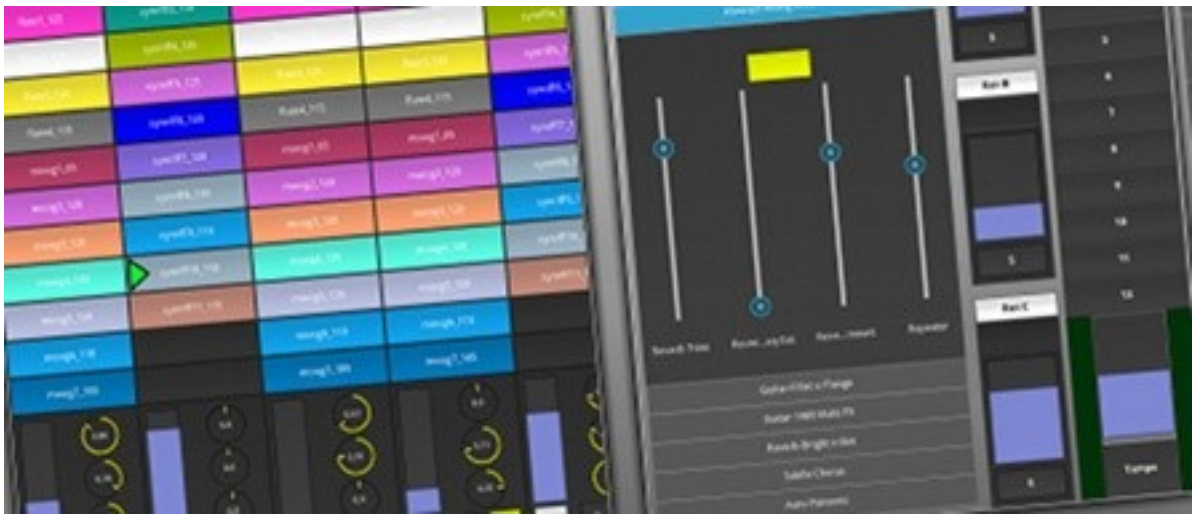
Une alternative à Glade

Il existe un programme équivalent à Glade, appelé Gazpacho et entièrement écrit en Python. Son interface est légèrement différente, mais il offre globalement les mêmes fonctionnalités, avec en plus l'accès à un ensemble supplémentaire de widgets (Kiwi).

Autres types d'interfaces graphiques

D'autres technologies matures ou innovantes existent. En voici quelques exemples :

Kivy est un framework de développement rapide d'applications faisant usage d'interfaces innovantes comme les interfaces tactiles multipoints, et fonctionne sous Linux, MacOsX, Windows, Android et iOS.



Jython est une version de Python fonctionnant sur la plateforme Java. Il permet d'accéder en Python à toutes les bibliothèques Java et donc de créer des interfaces graphiques avec *AWT* ou *Swing*.

IronPython est une version de Python fonctionnant sur la plateforme **.NET**. Il permet d'accéder à toutes les bibliothèques .NET et donc de créer des interfaces graphiques avec *WPF*, *MFC* ou directement *win32*.

Pyjamas provient des technologies **web**. Il s'agit d'un compilateur Python vers Javascript, d'un framework AJAX et d'un ensemble de widgets pour créer une application fonctionnant indifféremment sur le web ou en mode desktop.

7. Cloud, devops, infrastructure, systèmes

De par sa nature dynamique et multi-plateforme, et aidé par sa syntaxe simple et claire, Python se révèle être un très bon langage pour des besoins d'infrastructure allant d'un petit script système à une infrastructure Cloud géante. Ainsi on le retrouve de façon de plus en plus fréquente dans de nombreux outils d'installation, de configuration, de maintenance, de surveillance, de déploiement ou d'hébergement.

La gestion des exceptions, avec les erreurs possibles clairement identifiées dans les modules systèmes, permet ainsi de gérer finement et clairement les différents états dans lesquels une opération peut se terminer. L'intégration à l'environnement, la gestion simple des variables systèmes, des arborescences, des journaux d'exécutions, des expressions rationnelles puissantes permettent entre autres d'écrire des scripts systèmes efficaces, lisibles et maintenables.

OpenStack

Openstack est un monumental ensemble de logiciels consacrés à la création de clouds privés et publics. Il est devenu en quelques années la référence open-source internationale en matière de Cloud à tel point que des entreprises comme HP ont investi en 2014 un milliard de dollars dans cette solution, et qu'il est utilisé par exemple en France comme infrastructure pour offrir des services de Cloud souverain. OpenStack est écrit majoritairement en Python.

Salt, Ansible

Salt et Ansible représentent la nouvelle tendance en matière d'automatisation d'infrastructure et de cloud. Ils facilitent l'automatisation, le provisionning, l'orchestration et la gestion de configuration de parcs d'applications, de machines physiques, de serveurs virtuels ou de Clouds en tous genres.

ANSIBLE

8. Jeux vidéos

Le domaine des jeux vidéos n'est pas non plus ignoré :

Dans le jeu en ligne massivement multijoueurs **EVE online**, Stackless Python a été utilisé pour implémenter le serveur de jeu, un système demandant une approche hautement concurrentielle de la programmation et de grandes performances. La souplesse de Python permet ici de créer un design applicatif extensible, adapté à un besoin en évolution constante, tout en restant facile à modifier et améliorer dans des délais brefs. Dans le jeu **Civilisation IV**, un jeu de stratégie réinventant l'histoire des civilisations humaines sur plusieurs milliers d'années, Python est utilisé pour accéder à de nombreux composants du jeux, et scripter un certain nombre de fonctionnements.

Il existe également des moteurs 3D comme les moteurs **Panda3D**, ou **Soya3d**, dont les cœurs sont respectivement en C++ et Python/Pyrex et avec lesquels on peut écrire des jeux entièrement en Python et avec de bonnes performances.

9. Performances et algorithmique

Apprentissage

La simplicité d'utilisation de Python en fait un langage intéressant pour apprendre l'algorithmique sans être perturbé par des considérations telles que la compilation, la gestion des pointeurs et de la mémoire ou la maîtrise d'outils liés au langage. Juste après installation, on peut immédiatement lancer un interprète et effectuer quelques tests en ligne de commande. À ce stade, même un éditeur de code est inutile.

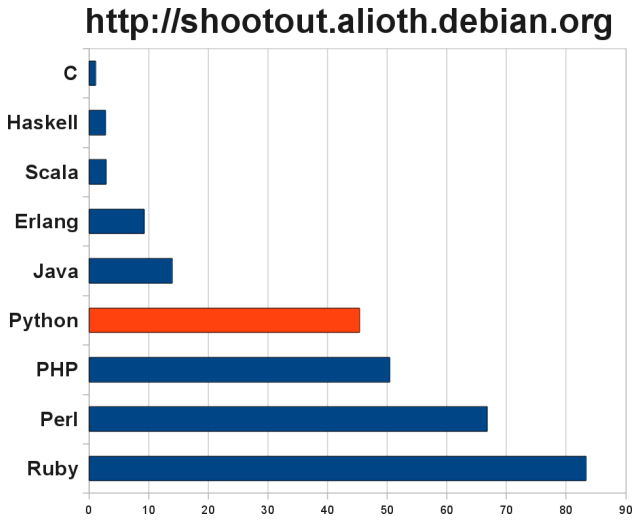
Maquettage

Réaliser un algorithme intensif final en pur Python n'est pas une bonne idée, mais faire une maquette de cet algorithme en Python en est une excellente. Cela permet de confirmer ou infirmer une hypothèse très rapidement et d'éviter d'allouer trop de temps ou de ressource à quelque chose qui pourra éventuellement être abandonné.

Une fois la maquette réalisée, il suffit de réécrire uniquement les parties qui doivent être accelérées dans un langage compilé puis de les placer dans un module Python, ou bien d'utiliser une des techniques ci-dessous qui ont l'avantage de rester en Python.

Optimisation

Python étant un langage interprété, il profite pleinement du fait que le temps d'exécution d'un programme coûte beaucoup moins cher que son temps de développement : le but est de créer des programmes le plus rapidement possible, tout en restant extrêmement lisible et maintenable.



Néanmoins des besoins d'optimisation peuvent survenir. Pour ce qui est des performances algorithmiques, Python permet d'exprimer simplement des algorithmes complexes et de comparer les performances de différentes approches, mais ne sera pas nécessairement la plateforme de choix pour faire tourner, in fine, lesdits algorithmes.

Diverses approches ont été explorées pour offrir des accélérations, sous la forme d'implémentations alternatives, de compilation *just-in-time*, de traduction de code ou simplement d'optimisations.

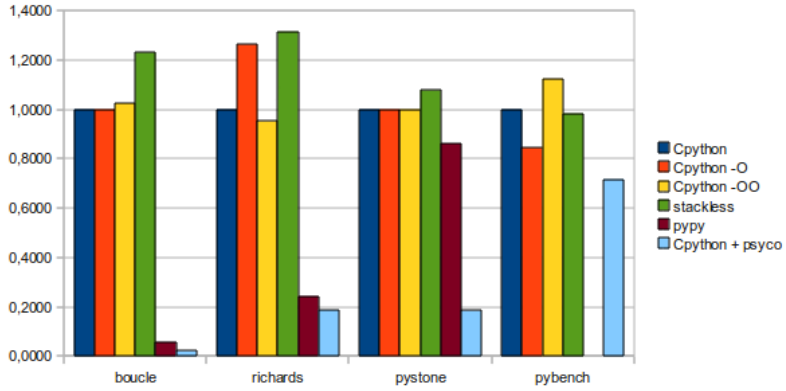
Voici différentes techniques pouvant être utilisées, dont certaines sont capables d'atteindre les performances du langage C.

Psyco : c'est un mécanisme de compilation *just-in-time*. Psyco annonce un gain moyen de 4x, mais pouvant aller jusqu'a 100x, et ne demande aucune modification du code source. Il ne marche cependant que sur architecture i386.

Stackless Python : Stackless Python vise à améliorer le support du multiprocesseur en ajoutant des outils de parallélisme au langage : les *tasklets*, les *channels*, un *ordonanceur*...

PyPy : très ambitieux et dopé un temps par un financement européen, PyPy est un projet de recherche visant à effectuer de la traduction de code et de la compilation. L'idée directrice est de traduire une description de Python effectuée en Python lui-même vers des langages de plus bas-niveau. La rumeur veut que le but recherché est d'aller plus vite que le C. PyPy commence à connaître des retombées intéressantes et affiche des performances souvent meilleures que le Python de référence.

Cython : produit des modules C à partir du code python, permet de rendre les types de variables statiques, offrant dans ce cas de grandes optimisations. C'est l'outil parfait pour accélérer les zones d'un programme qui doivent absolument être rapides.



Voici un bref graphique comparatif de quelques *benchmarks*, afin de visualiser les différences potentielles entre les solutions mentionnées ci-dessus. L'unité est le temps mis par CPython sur chaque algorithme.

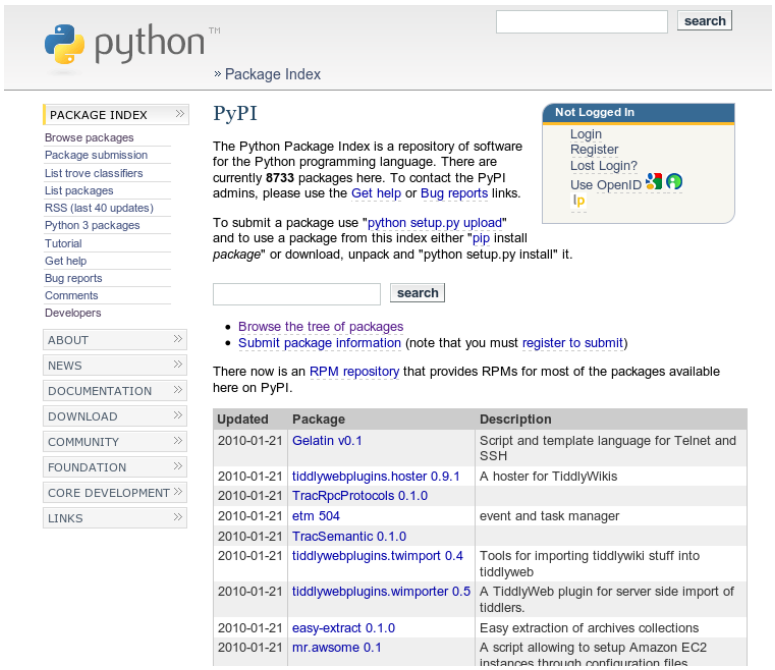
On constate que les possibilités d'accélération d'un code sont énormes avec Psyco qui est probablement l'une des solutions les plus efficaces actuellement. Stackless Python trouve tout son intérêt lors d'écriture d'applications multithread.

3. Outils de qualité et d'industrialisation

Par défaut et dans l'esprit, Python n'oblige pas les développeurs à utiliser des outils lourds et contraignants, un simple éditeur de texte et un terminal sont suffisants pour développer des applications de toutes tailles, y compris les plus importantes. La courbe d'apprentissage est ainsi beaucoup plus douce que dans d'autres environnements.

Néanmoins tous les outils sont présents pour s'adapter aux goûts de chacun, aux pratiques modernes ainsi qu'aux contraintes industrielles. Cette section présente des outils qui doivent être mis en place pour améliorer la qualité des projets, l'intégration continue ou la réactivité des équipes.

1. Index général des paquets Python



Les composants applicatifs en Python peuvent être distribués sous forme de paquets individuels appelés des *eggs*. Un projet peut donc être décomposé en plusieurs *eggs* et ceux-ci peuvent être facilement réutilisés dans d'autres projets.

Dans le cadre d'une gestion de projet, le découpage en *eggs* facilite la répartition des tâches entre plusieurs équipes qui peuvent publier individuellement les versions de leur module pour les intégrer au produit final. La gestion de projet elle-même peut donc être séparée et le module peut suivre son propre cycle de développement, test, publication, intégration, mise en production, et maintenance corrective.

Les composants ayant un intérêt général sont publiés sur un site web regroupant l'ensemble de ce qui est produit par la communauté : l'index PyPI (*Python Package Index*). Cette pratique, courante dans le milieu du logiciel libre, améliore la mutualisation du code, évite de réinventer ce qui existe déjà, permet de trouver facilement un module pour un besoin particulier, et encourage tout le monde à écrire du code générique réutilisable.

Grâce à des outils comme **PIP**, tout composant présent sur l'index PyPI peut être téléchargé et installé d'une simple commande ou ajouté à un projet par une simple ligne dans un fichier de configuration. À l'inverse, un composant peut être publié sur l'index PyPI avec la même facilité, ou éventuellement sur un index privé, interne à l'entreprise.

2. Construction d'applications

Une application se compose généralement d'un assemblage de produits, de paquets, de bases de données, de bibliothèques, de code source spécifique, ou de configuration. Pour assembler tous ces éléments de manière automatique et répétable, il est possible d'utiliser un outil de construction. **Buildout** en fait partie et est très utilisé dans le milieu du web. Il se base sur une succession de « recettes », chaque recette étant responsable d'une partie de l'application, par exemple la mise en place de la base de données ou d'un distributeur de charge. Les recettes sont elles-mêmes distribuées sous forme de paquets Python et sont disponibles pour tous. Un unique fichier de configuration de quelques dizaines de lignes est suffisant pour décrire et construire une application complexe en puisant dans diverses sources.

3. Déploiement d'applications

Le déploiement s'automatise facilement, soit en utilisant l'outil *Buildout* mentionné dans le paragraphe précédent, éventuellement en générant une archive ou un paquet système contenant l'application. Des outils additionnels comme *Fabric* peuvent être utilisés pour automatiser des tâches simultanément sur plusieurs serveurs. Pour des cas plus complexes, Salt peut avantageusement être utilisé pour effectuer des déploiements et configurations en parallèle.

4. Tests automatisés et documentés

Si les tests unitaires peuvent se pratiquer dans tous les environnements, la plateforme Python offre une notion complémentaire extrêmement bénéfique : les « *doctests* ». On peut présenter les *doctests* au choix comme de la **documentation testée** ou des **tests documentés**. Ils permettent de documenter et tester un projet en même temps pendant la phase de conception, avant même l'écriture du code. Il s'agit d'une succession de paragraphes explicatifs et de morceaux de code de haut niveau donnant un aperçu de l'utilisation des composants applicatifs dans une console Python.

En voici un exemple :

```
Voici comment utiliser la classe `Ham` avec ses attributs
et ses méthodes. On peut créer une instance de la classe et modifier ses
attributs :

>>> from monprojet import Ham
>>> ham = Ham()
>>> ham.nom = 'nom du ham'
>>> ham.valeurs = [2, 3, 4]

Ensuite on peut calculer la moyenne des valeurs grâce à une méthode `mean` :

>>> ham.mean()
3.0
```

Cette documentation peut être exécutée comme un programme test et on doit retrouver à l'exécution les mêmes valeurs de sortie que dans le texte.

Les *doctests* sont complémentaires aux tests unitaires ou fonctionnels et garantissent que la documentation existe et est à jour. Cette technique permet de travailler naturellement et simultanément en **développement dirigé par les tests** et **développement dirigé par la documentation**. L'ensemble de ces pratiques apportent des avantages indéniables :

- garantie de non-régression
- documentation existante et à jour
- détection précoce des erreurs de conception
- possibilités d'intégration continue

En Python il est également possible et conseillé d'intégrer une (petite) partie de la documentation directement dans le code source sous forme de *docstring*, qui peuvent elles-même être des *doctests* ! On peut donc placer des petits tests documentés directement dans le corps du programme, à l'intention des développeurs. Cette documentation intégrée dans le code peut ensuite être testée et extraite automatiquement afin de générer automatiquement des livres de documentation et des rapports de tests. Dans ce cas d'utilisation, la proximité entre le code et les tests garantissent que les tests sont adéquats et modifiés en même temps.

Parmi l'éventail des outils de tests, on trouve également des outils de mesure du **taux de couverture** des tests, qui sont souvent intégrés par défaut dans les outils de lancement de test et doivent faire partie de l'indicateur final de qualité.

Les outils de *Mock* sont également utiles et aident à simuler un composant applicatif manquant pour les tests.

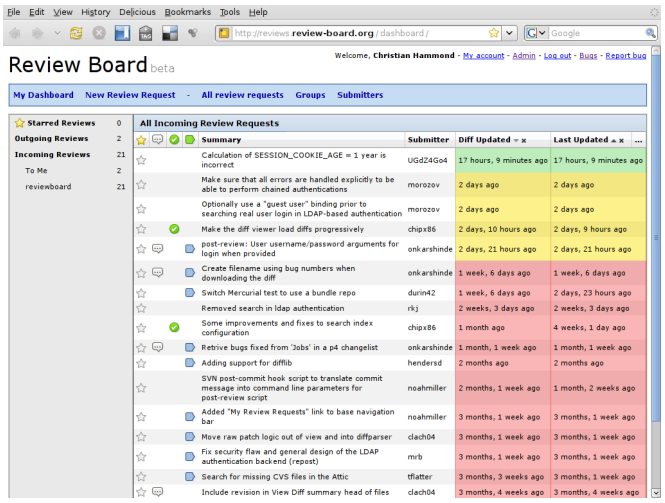
5. Qualité du code

Analyse automatique : comme déjà mentionné dans le premier chapitre, un code Python doit préférablement suivre un ensemble de conventions de codage (définies dans le document *PEP-8*). En plus de ces conventions, il est possible de définir un ensemble de règles spécifiques à un projet, ou de contraintes supplémentaires sur le code. Des outils d'analyse du code sont dans ce cas utiles pour vérifier ces règles, et définir un ensemble de métriques de qualité à respecter. Ces métriques peuvent alors être vérifiées en même temps que les tests unitaires.

Parmi les outils d'analyse automatique du code, on peut citer *Pylint*, *PyFlakes* *Flake8* ou *PyChecker*. Ces outils sont capables de détecter des problèmes potentiels dans le code, un manque de documentation, un non-respect des conventions, des imports inutiles, des morceaux de code jamais exécutés, etc.

Des indicateurs supplémentaires existent pour détecter par exemple des problèmes de conception, mesurer la complexité cyclomatique du code, ou trouver des redondances potentielles.

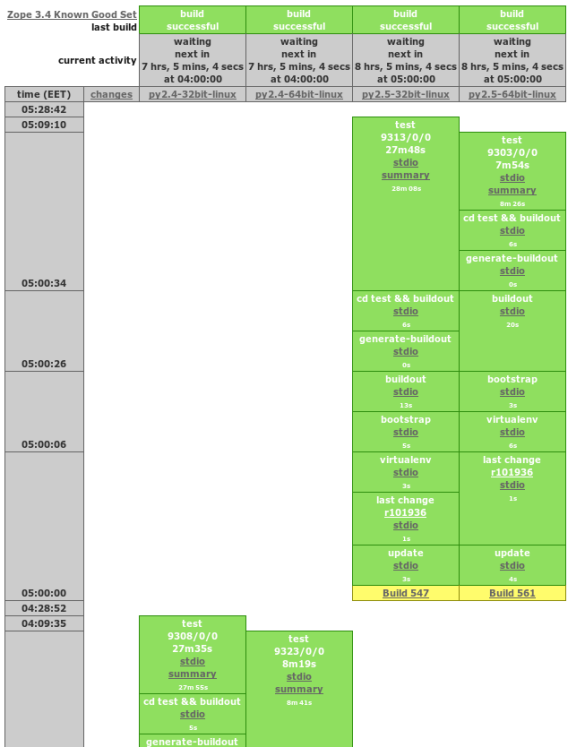
Analyse humaine : l'analyse humaine se pratique notamment grâce à des outils de revue de code. L'application *Review Board*, écrite elle-même en Python (Django), prend en charge cette fonctionnalité : au travers d'une interface web conviviale, le code source peut être commenté, toute modification peut être soumise à approbation avant ou après propagation dans le dépôt.



Analyse aléatoire : les outils de *fuzzing* peuvent détecter des bugs ou des problèmes de sécurité en soumettant au programme des données invalides générées aléatoirement, ou générées à partir de données valides dont quelques bits ont été modifiés aléatoirement. Plusieurs outils de ce genre sont écrits en Python, l'un d'entre eux (*Fusil*) est capable de tester n'importe quel programme et a même servi à détecter des problèmes dans de nombreux logiciels tels que Gimp, PHP ou Python lui-même.

6. Intégration continue

Pour améliorer la qualité et la réactivité d'un développement, on peut mettre en place un robot qui lance les tests unitaires et fonctionnels automatiquement, de manière périodique ou de manière synchrone dès qu'une modification est effectuée sur le code. L'équipe peut être ainsi avertie immédiatement et sans intervention humaine.



L'un des outils les plus utilisés est *BuildBot*, un robot d'intégration continue écrit en Python. Cet outil est très souple. Un ensemble de clients *BuildBots* peut être lancé sur différentes machines, par exemple un Windows 32bits, un Linux 64bits, et renvoyer les informations à un serveur Buildbot qui affiche un tableau récapitulatif de tous les tests et de leurs résultats. En même temps que les tests unitaires, le *BuildBot* peut exécuter et publier les analyses de qualité de code. En fonction de ce résultat, n'importe quelle action automatique peut être entreprise, comme l'envoi d'un e-mail ou le hurlement d'un Tuxdroïd.

Les métriques de qualité peuvent aussi être recueillies par des outils comme *Bitten* pour faciliter l'intégration continue. L'intérêt de *Bitten* est qu'il s'intègre dans la forge *Trac*, pour rapprocher les métriques qualité de la gestion des tickets, la documentation ou le code source.

7. Génération de documentation

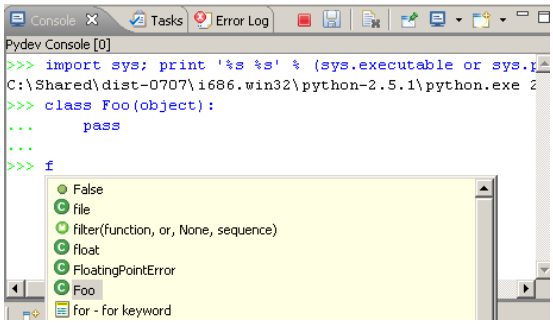
L'univers Python utilise une syntaxe commune pour la documentation, les commentaires de code ou les *doctests* : le *reStructuredText* (RST). Le RST a la particularité d'être conçu pour être lu directement. Il ne comporte pas d'éléments gênant la lecture tels que les balises ou les délimiteurs de blocs (crochets, accolades...), mais est inspiré des pratiques intuitives d'écriture de contenu en mode texte pur. Voici un exemple :

```
Outils d'industrialisation
==========================

Par exemple, les titres de chapitre sont soulignés de signes 'egal'

Génération de documentation
---------------------------

Les titres de paragraphe sont soulignés de tirets,
l'emphase est indiquée avec des **étoiles** (gras) ou des `quotes`
(italique)
```

Python fournit tous les outils nécessaires pour valoriser de la documentation écrite en RST. L'outil phare de génération de documentation dans l'écosystème Python est **Sphinx**. Cet outil génère de la documentation au format HTML, PDF ou Latex à partir de documents en RST. Le html produit offre par défaut une interface web conviviale avec une table des matières générée automatiquement et un moteur de recherche intégré écrit en Javascript. Il gère la coloration syntaxique des blocs de code pour de nombreux langages. Il peut également inspecter un projet pour extraire la documentation du code et la liste des composants.

Sphinx peut être utilisé pour n'importe quel projet de documentation. Le phase initiale d'écriture de cette publication a elle-même été gérée avec cet outil.

8. Environnements de développement intégrés



Les développeurs habitués aux environnement de développement intégrés pourront retrouver leurs marques car il est possible de travailler notamment avec *Eclipse* en utilisant un module nommé *PyDev*, qui offre tous les outils classiques d'un EDI avec une spécialisation pour Python : complétion de code, colorisation de syntaxe, analyse et évaluation de code, etc.

D'autres EDI non mentionnés ici sont disponibles, aussi bien libres que propriétaires et souvent écrits eux-mêmes en Python.

9. Dépôts de code source

Les dépôts de code source contiennent tout l'historique, ligne par ligne, de l'évolution du code source d'un projet. En dehors de *Subversion*, encore très utilisé, la tendance actuelle est à l'utilisation de gestionnaires de version décentralisés comme *Mercurial*, *Bazaar* ou *Git*. Sur ces trois systèmes, les deux premiers sont écrits en Python ! Leur intérêt est multiple, ils ne nécessitent pas de serveur central, sont faciles à utiliser, autorisent un travail distant hors-ligne, s'adaptent à n'importe quelle organisation d'équipe, et offrent de par leur fonctionnement une réplication intrinsèque pouvant servir de sauvegarde. De très nombreux projets migrent actuellement vers ces systèmes, grâce à la facilité avec laquelle on peut convertir un dépôt Subversion en dépôt décentralisé.

10.Forges de développement

Les forges sont des environnements conviviaux regroupant dans une interface unifiée tous les outils utiles pendant les phases de développement d'un logiciel : gestion des tickets, wiki de documentation, navigation dans le code, gestion des traductions, affichage des métriques de qualité, affichage du résultat des tests automatisés, etc. Ces forges peuvent être installées en interne dans les entreprises et peuvent gérer souvent plusieurs projets en même temps. ***Trac*** en fait partie : cette forge s'installe en quelques minutes et bénéficie de nombreux modules offrant des fonctionnalités additionnelles. Certaines de ces forges possèdent un pendant public comme le *Launchpad* de *Canonical* qui gère de très nombreux projets de logiciels libres, notamment le système d'exploitation Ubuntu Linux. Le *Launchpad* a été libéré depuis quelques mois, il offre un niveau fonctionnel très large pour les entreprises souhaitant s'en équiper.

Kallithea est une autre forge open-source écrite en Python, issue d'un fork de RhodeCode et qui a évolué de manière très positive pour arriver à un niveau proche des solutions propriétaires en ligne comme Github ou Bitbucket. Elle offre aujourd'hui tout ce dont une équipe de développement a besoin : navigation dans des dépôts *Git* ou *Mercurial*, revue et commentaire de code, notifications, Gists, indexation des dépôts, Pull requests, etc.

4. Communauté

Le langage Python est développé sur le mode communautaire et n'est pas dépendant d'une seule entreprise. Il est le résultat de travaux et de votes issus de propositions d'améliorations nommées « PEP » (*Python Enhancement Proposal*). Au delà du langage en lui-même, la création et le support de l'ensemble des outils et bibliothèques de l'écosystème est pris en charge par une communauté internationale constituée d'entreprises, particuliers, organisations ou associations.

En France, le côté associatif est animé par l'**afpy**, au travers de nombreuses activités :

- conférence annuelle PyCon FR
- rencontres mensuelles sur des sujets techniques et non techniques
- présence aux salons, rencontres et conférences :
  - Salon Solutions Linux
  - RMLL
  - OSDC fr
  - JDLL
  - JM2L
- Rédaction de dossiers et articles de presse(Hors-Série Linux Magazine n°40 sur Python)
- Animation du site web http://afpy.org (forum, liste de diffusion, offres d'emploi, etc.)
- Présence sur des canaux IRC

5. Conclusion

Les 20 ans d'âge de la plateforme Python lui ont apporté une forte maturité et ont engendré un écosystème très varié. Cet écosystème est composé de nombreux acteurs, institutions, indépendants, particuliers, associations, mais surtout de très nombreuses entreprises des plus petites aux plus grandes comme Google ou Microsoft qui ont compris les avantages - agilité, polyvalence, lisibilité et efficacité - et l'engouement que pouvait apporter ce langage. Dans de nombreux pays et secteurs, on constate même une forte accélération du taux d'utilisation de Python. Tournée vers l'avenir, la plateforme a également réussi sa modernisation avec la sortie d'une nouvelle version majeure (3.0 à 3.4) qui se débarrasse de certains défauts de jeunesse. Or, en dépit d'une bonne industrialisation de ses procédés et la présence de nombreux outils, Python reste une plateforme extrêmement facile à aborder et très homogène. Cette homogénéité tire sa source d'une sensibilité propre à la communauté, qui se retrouve dans l'écriture de ce qu'on appelle un code « pythonique ». Qu'est-ce qu'un code pythonique ? Une première réponse existe dans l'interprète Python lui-même :

```
>>> import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one -- and preferably only one —obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

6. Licence et diffusion