
Table of Contents

Avant-propos

Introduction	1.1
Dédicace	1.2
Préface	1.3

A byte of Python

À propos de Python	2.1
Installation	2.2
Premiers pas	2.3
Les bases	2.4
Opérateurs et expressions	2.5
Structures de contrôle	2.6
Fonctions	2.7
Modules	2.8
Structures de données	2.9
Résolution de problème	2.10
Programmation orientée objet	2.11
Entrées et sorties	2.12
Exceptions	2.13
Bibliothèque standard	2.14
Pour aller plus loin	2.15
Et ensuite	2.16

Annexes

FLOSS	3.1
À propos du livre	3.2
Historique des révisions	3.3
Traductions	3.4
Guide de traduction	3.5
Feedback	3.6

A Byte of Python

« A Byte of Python » est un livre libre de droits sur la programmation avec le langage Python. Il peut être vu comme un tutoriel ou un guide sur le langage Python pour les débutants. Si tout ce que vous savez sur l'informatique se résume à enregistrer des fichiers, alors ce livre est fait pour vous.

Pour Python version 3

Ce livre vous apprendra à utiliser Python version 3. Il vous aidera également dans votre adaptation à la version 2 de Python, plus ancienne et plus courante.

Qui lit A Byte of Python?

Voilà ce que disent les lecteurs à propos du livre:

Ceci est le meilleur tutoriel pour débutant que j'ai jamais lu ! Merci pour votre effort. -- [Walt Michalik](#)

La meilleure chose que j'ai trouvée est « A Byte of Python », qui est tout simplement un livre brillant pour un débutant. C'est bien écrit, les concepts sont bien expliqués avec des exemples évidents. -- [Joshua Robin](#)

Excellent introduction en douceur à la programmation #Python pour les débutants -- [Shan Rajasekaran](#)

Meilleur guide pour débutant en python -- [Nickson Kaigi](#)

commencez à aimer le python à chaque page lue -- [Herbert Feutl](#)

guide parfait pour débutants en python, vous donnera la clé pour déverrouiller le monde magique du python -- [Dilip](#)

Je devrais faire mon « vrai travail » mais je viens de trouver « A Byte of Python ». Un excellent guide avec d'excellents exemples. -- [Biologist John](#)

J'ai récemment commencé à lire A Byte of Python. Super travail. Et gratuitement. Fortement recommandé pour les pythonistes en herbe. -- [Mangesh](#)

A Byte of Python, écrit par Swaroop (c'est le livre que je lis actuellement). Probablement le meilleur pour commencer, et probablement le meilleur au monde pour chaque débutant ou même pour un utilisateur plus expérimenté. -- [Apostolos](#)

Je profite de la lecture de #ByteOfPython par @swaroopch meilleur livre de tous les temps -- [Yuvraj Sharma](#)

Merci beaucoup d'avoir écrit A Byte Of Python. Je viens de commencer à apprendre à coder il y a deux jours et je suis déjà en train de construire des jeux simples. Votre guide a été un rêve et je voulais juste vous dire à quel point il a été précieux. -- [Franklin](#)

Je viens du Dayanandasagar College of Engineering (7ème semestre, CSE). Tout d'abord, je tiens à dire que votre livre « A Byte of Python » est un livre trop bon pour un débutant en python comme moi. Les concepts sont si bien expliqués à l'aide d'exemples simples qui m'ont aidé à apprendre facilement le python. Merci beaucoup. -- [Madhura](#)

Je suis un étudiant en informatique de 18 ans qui étudie à l'université irlandaise. Je tiens à vous exprimer ma gratitude pour avoir écrit votre livre « A Byte of Python ». Je connaissais déjà 3 langages de programmation - C, Java et Javascript, et Python était de loin le langage le plus simple que j'ai jamais appris et que était principalement dû au fait que votre livre était fantastique et rendait l'apprentissage du python très simple et intéressant. C'est l'un des meilleurs livres de programmation écrits et faciles à suivre que j'ai jamais lus. Félicitations et continuez votre bon travail. -- [Matt](#)

Bonjour, je viens de la République Dominicaine. Mon nom est Pavel, j'ai récemment lu votre livre A Byte of Python et je le considère comme excellent !! :) J'ai beaucoup appris de tous les exemples. Votre livre est d'une grande aide pour les débutants comme moi ... -- [Pavel Simo](#)

Je suis un étudiant originaire de Chine. J'ai lu votre livre « A Byte of Python ». Oh, c'est beau. Le livre est très simple mais peut aider tous les premiers apprenants. Vous savez que je suis intéressé par Java et par le cloud computing, je dois programmer un programme pour le serveur. Je pense donc que python est un bon choix. J'ai terminé votre livre. Je pense que ce n'est pas seulement un bon choix, il doit également utiliser le Python. Mon anglais n'est pas très bien, l'email à vous, je veux juste vous remercier! Meilleurs voeux pour vous et votre famille. -- Roy Lau

J'ai récemment fini de lire Byte of Python et je pensais devoir vous remercier. J'étais très triste d'atteindre les dernières pages, car je dois maintenant revenir à des manuels ennuyeux et fastidieux pour apprendre le python. Quoi qu'il en soit, j'apprécie vraiment votre livre. [Samuel Young](#)

Cher Swaroop, je prends une classe d'un enseignant qui n'a pas d'intérêt dans l'enseignement. Nous utilisons Learning Python, deuxième édition, par O'Reilly. Ce n'est pas un texte pour débutant sans aucune connaissance en programmation, et l'instructeur devrait travailler dans un autre domaine. Merci beaucoup pour votre livre, sans cela, je n'aurais rien compris à Python et à la programmation. Merci mille fois, vous êtes capable de décomposer le message à un niveau que des débutants peuvent comprendre et ce n'est pas donné à tout le monde. -- [Joseph Duarte](#)

J'adore votre livre! C'est le meilleur tutoriel sur Python de tous les temps et une référence très utile. Brillant, un vrai chef-d'œuvre! Continuez votre bon travail! -- [Chris-André Sommerseth](#)

Tout d'abord, je tiens à vous remercier pour ce superbe livre. Je pense que c'est un bon livre pour ceux qui recherchent un tutoriel pour débutant en Python. J'ai entendu parler de ce livre, je pense, il y a deux ou trois ans environ. À cette époque, je n'étais pas encore capable de lire un livre en anglais. J'ai donc obtenu une traduction en chinois, qui m'a amené à la programmation en Python. Récemment, j'ai relu ce livre. Cette fois, bien sûr, la version anglaise. Je ne pouvais pas croire que je puisse lire le livre en entier sans mon dictionnaire sous la main. Bien sûr, c'est dû à tous vos efforts pour rendre ce livre facile à comprendre. -- [myd7349](#)

Je vous envoie simplement un e-mail pour vous remercier d'avoir écrit Byte of Python en ligne. J'avais essayé Python pendant quelques mois avant de tomber sur votre livre, et bien j'ai eu un succès limité avec pyGame, je n'ai jamais terminé un programme. Grâce à votre simplification par catégories, Python semble être un objectif atteignable. Il semble que j'ai enfin appris les bases et que je puisse continuer dans mon objectif réel, le développement de jeux. ... Encore une fois, merci beaucoup d'avoir placé un guide aussi structuré et utile sur la programmation de base sur le Web. Cela m'a poussé à entrer dans la POO avec une compréhension que deux livres avaient échoué à m'apporter. -- [Matt Gallivan](#)

Je tiens à vous remercier pour votre livre *A Byte of Python* que je trouve moi-même le meilleur moyen d'apprendre le python. J'ai 15 ans, je vis en Egypte, mon nom est Ahmed. Python était mon deuxième langage de programmation. J'ai appris Visual Basic 6 à l'école mais je n'ai pas apprécié, mais j'ai vraiment aimé apprendre le python. J'ai créé un programme de carnet d'adresses et j'ai réussi. Je vais essayer de commencer à créer plus de programmes et à lire des programmes Python (si vous pouviez m'indiquer des codes sources qui pourraient être utiles). Je vais aussi commencer par apprendre java et si vous pouvez me dire où trouver un tutoriel aussi bon que le vôtre pour java, cela me serait très utile. Merci -- [Ahmed Mohammed](#)

Le didacticiel PDF de 110 pages A Byte of Python de Swaroop C H. est une excellente ressource pour les débutants qui souhaitent en savoir plus sur Python. Il est bien écrit, facile à suivre et constitue peut-être la meilleure introduction disponible à la programmation Python. -- [Drew Ames](#)

Hier, j'ai lu presque tout Byte of Python sur mon Nokia N800 et c'est l'introduction de Python la plus simple et la plus concise que j'ai jamais rencontrée. Fortement recommandé comme point de départ pour apprendre Python. -- [Jason Delport](#)

Byte of Vim et Byte of Python de @swaroopch est de loin le meilleur travail d'écriture technique auquel j'ai eu affaire. Excellente lecture #FeelGoodFactor -- [Surendran](#)

« Byte of python » est le meilleur de loin mec (en réponse à la question « Quelqu'un peut-il suggérer une bonne ressource bon marché pour apprendre les bases de Python? ») -- [Justin LoveTrue](#)

Le livre Byte de python était très utile .. Merci beaucoup :) [Chinmay](#)

J'ai toujours été fan de A Byte of Python, conçu pour les programmeurs débutants et expérimentés. -- [Patrick Harrington](#)

J'ai commencé à apprendre Python il y a quelques jours à partir de votre livre. Merci pour un livre aussi agréable. Il est si bien écrit, que vous m'avez rendu la vie facile... donc vous voici avec un nouveau fan... c'est moi :). Des tonnes de remerciements. -- [Gadadhari Bheem](#)

Avant de commencer à apprendre le python, j'avais acquis des compétences de base en programmation en assembleur, C, C++, C# et Java. La raison même pour laquelle j'ai voulu apprendre le python est qu'il est populaire (les gens en parlent) et puissant (la réalité). Ce livre écrit par M. Swaroop est un très bon guide pour les nouveaux programmeurs et les nouveaux programmeurs python. Il a fallu 10 demi-journées pour le parcourir. Grande aide! -- [Fang Biyi \(PhD Candidate ECE, Michigan State University\)](#)

Merci beaucoup pour ce livre !! Ce livre a éclairci de nombreuses questions sur certains aspects de Python, telles que la programmation orientée objet. Je ne me sens pas comme un expert en OO mais je sais que ce livre m'a aidé pour les premières étapes. J'ai maintenant écrit plusieurs programmes python qui font de vraies choses pour moi en tant qu'administrateur système. Ils sont tous procéduraux mais ils sont petits par rapport aux standards de la plupart des gens. Encore merci pour ce livre. Merci de l'avoir mis à disposition sur le web. -- Bob

Je tiens simplement à vous remercier d'avoir écrit le premier livre sur la programmation que j'ai jamais vraiment lu. Python est maintenant ma langue maternelle et je peux imaginer toutes les possibilités. Je vous remercie donc de m'avoir donné les outils pour créer des choses que je n'aurais jamais imaginé pouvoir faire auparavant. -- « The Walrus »

Je voulais vous remercier d'avoir écrit *A Byte Of Python* (versions 2 et 3). Cela a été inestimable pour mon expérience d'apprentissage en Python et en Programmation en général. Inutile de dire que je suis un débutant dans le monde de la programmation, quelques mois d'auto-apprentissage jusqu'à maintenant. J'avais utilisé des tutoriels youtube et d'autres tutoriels en ligne, y compris d'autres livres gratuits. J'ai décidé de fouiller dans votre livre hier et j'ai appris davantage au cours des premières pages que tout autre livre ou tutoriel. Quelques choses qui m'avaient dérouté, ont été éclaircies avec un excellent exemple et une explication. J'ai hâte de lire (et d'apprendre) plus !! Merci beaucoup d'avoir non seulement écrit le livre, mais aussi de l'avoir placé sous la licence Creative Commons (gratuite). Dieu merci, il y a des gens altruistes comme vous pour aider et enseigner au reste d'entre nous. -- Chris

Je vous ai écrit en 2011 et je commençais tout juste à utiliser Python. Je souhaitais vous remercier pour votre tutoriel « A Byte of Python ». Sans cela, j'aurais abandonné en chemin. Depuis lors, j'ai programmé un certain nombre de fonctions dans ma société avec ce langage, et ce n'est pas fini. Je n'oserais pas m'appeler « programmeur avancé », mais je remarque les demandes d'aide occasionnelles de la part d'autres personnes depuis que je l'utilise. En lisant « Byte », j'ai découvert pourquoi j'avais cessé d'étudier le C et le C++ et c'est parce que le livre qui m'a été donné a commencé avec un exemple contenant une affectation augmentée (+=). Bien sûr, il n'y avait aucune explication à cet arrangement d'opérateurs et je me suis arraché les cheveux en essayant de comprendre ce qui était sur la page écrite. Si je me souviens bien, c'était un exercice des plus frustrants et j'ai fini par abandonner. Cela ne veut pas dire que C ou C++ soit impossible à apprendre, ou même que je sois stupide, mais cela signifie que la documentation à partir de laquelle j'ai travaillé n'a pas défini les symboles et les mots, ce qui est une partie essentielle de toute instruction. Tout comme les ordinateurs ne pourront pas comprendre un mot ou un symbole d'ordinateur qui ne soit pas conforme à la syntaxe du langage utilisé, un étudiant débutant dans un domaine donné ne comprendra pas son sujet s'il rencontre des mots ou des symboles pour lesquels il n'y a pas de définition. Vous obtenez un « écran bleu » pour ainsi dire dans les deux cas. La solution est simple: rechercher le mot ou le symbole et obtenir la définition ou le symbole approprié et voilà, l'ordinateur ou l'étudiant peut procéder. Votre livre est si bien rédigé que je n'y ai trouvé que très peu de choses que je ne comprenais pas. Alors merci. Je vous encourage à continuer à inclure des définitions complètes des termes. La documentation de Python est bonne, une fois que vous avez déjà compris (les exemples font sa force de ce que je vois), mais dans de nombreux cas, il semble que vous ayez à savoir en avance pour comprendre la documentation, ce qui, à mon avis, ne devrait pas être le cas. Les didacticiels tiers indiquent la nécessité de clarifier la documentation et leur succès dépend en grande partie des mots utilisés pour décrire le jargon. J'ai recommandé votre livre à beaucoup d'autres. Certains en Australie, certains dans les Caraïbes et d'autres encore aux États-Unis. Il remplit un créneau qu'aucun autre livre ne couvre. J'espère que vous allez bien et je vous souhaite tout le succès possible à l'avenir. -- Nick

hé, c'est ankush(19). Je faisais face à une grande difficulté pour commencer avec python. J'ai essayé beaucoup de livres, mais ils étaient tous plus volumineux et n'étaient pas orienté objectif. et puis j'ai trouvé cette adorable livre, qui m'a fait aimer le python en un rien de temps. Merci beaucoup pour ce « beau morceau de livre ». -- Ankush

Je tiens à vous remercier pour votre excellent guide sur Python. Je suis biologiste moléculaire (avec peu de formation en programmation) et pour mon travail, j'ai besoin de manipuler de grands ensembles de données de séquences d'ADN et d'analyser des images au microscope. Pour les deux choses, la programmation en python a été utile, voire indispensable pour terminer et publier un projet de 6 ans. La disponibilité d'un tel guide est un signe clair que les forces du mal ne dominent encore pas le monde! :) -- Luca

Étant donné que ce sera le premier language que vous apprendrez, vous pourriez utiliser A Byte of Python. Il donne vraiment une bonne introduction à la programmation en Python et est bien rythmé pour le débutant moyen. Dès lors, le plus important sera de commencer à vous entraîner à créer vos propres petits programmes. -- « [{Unregistered}](#) »

Juste pour dire un fort et joyeux merci beaucoup pour la publication de « A Byte of Python » et « A Byte of Vim ». Ces livres m'ont été très utiles il y a quatre ou cinq ans lorsque j'ai commencé à apprendre la programmation. En ce moment, je développe un projet qui a été un rêve pendant très longtemps et je veux juste dire merci. Continuez dans cette direction. Vous êtes une source de motivation. Bonne chance. -- Jocimar

J'ai fini de lire Un octet de Python en 3 jours. C'est vraiment intéressant. Pas une seule page n'était ennuyeuse. Je veux comprendre le code du lecteur d'écran Orca. Votre livre m'a, espérons, équipé pour cela. -- Dattatray

Salut, « A byte of python » est vraiment une bonne lecture pour les débutants en python. Donc, encore une fois, BEAU TRAVAIL! Je suis un programmeur chinois expérimenté en Java & C depuis 4 ans. Récemment, je souhaitais travailler sur le projet de note zim-wiki qui utilise pygtk. J'ai lu votre livre en 6 jours et je peux maintenant lire et écrire des exemples de code Python. Merci pour votre contribution. S'il vous plaît, gardez votre enthousiasme pour rendre ce monde meilleur, ceci est juste un petit mot d'encouragement de Chine. -- Lee

Je suis Isen, de Taiwan, diplômé en doctorat du département de génie électrique de l'Université nationale de Taiwan. Je voudrais vous remercier pour votre excellent livre. Je pense que ce n'est pas seulement facile à lire, mais aussi complet pour un nouveau venu de Python. La raison pour laquelle j'ai lu votre livre est que je commence à travailler sur le framework GNU Radio. Votre livre m'a permis de comprendre la plupart des idées de base importantes et les compétences de Python en un minimum de temps. J'ai aussi vu que cela ne vous dérangeait pas que les lecteurs vous envoient une note de remerciement dans votre livre. Donc, j'aime beaucoup votre livre et je l'apprécie. Merci. -- [Isen I-Chun Chao](#)

Le livre est même utilisé par la NASA! Il est utilisé au [Jet Propulsion Laboratory](#) avec leur projet Deep Space Network.

Cours en Universités

Ce livre est ou à été utilisé comme support dans différentes institutions:

- « Principes des langages de programmation » [Vrije Universiteit, Amsterdam](#)
- « Concepts de base de l'informatique » [Université de California, Davis](#)
- « Programmer avec Python » [Université de Harvard](#)
- « Introduction à la programmation » [Université de Leeds](#)
- « Introduction à la programmation d'applications » [Université de Boston](#)
- « Compétences informatiques pour la météorologie » [Université d'Oklahoma](#)
- « Géotraitement » [Université d'Etat du Michigan](#)
- « Systèmes Web sémantiques multi-agents » [Université d'Édimbourg](#)
- « Introduction à l'informatique et à la programmation » [MIT OpenCourseWare](#)
- « Programmation de base à la Faculté des sciences sociales, Université de Ljubljana, Slovénie » -- Aleš Žiberna dit « *J'ai (ainsi que mon prédecesseur) utilisé votre livre comme littérature principale pour ce cours* »
- « Introduction à la programmation », Département des sciences de l'information, Université de Zadar, Croatie -- Krešimir Zauder dit « *J'aimerais vous informer que la lecture de A Byte of Python est une lecture obligatoire de mon cours* »

License

Ce livre est sous la licence [Creative Commons Attribution-ShareAlike 4.0 International License](#).

Cela signifie que:

- Vous êtes libre de partager et donc copier, distribuer et transmettre ce livre.
- Vous êtes libre de retoucher et adapter ce livre (en particulier translations)
- Vous êtes libre de l'utiliser à des fins commerciales

Notez cependant:

- S'il vous plaît, ne vendez *pas* de copies électroniques ou imprimées du livre à moins que vous n'ayez clairement indiqué dans la description que ces copies ne sont *pas* de l'auteur original du livre.
- L'attribution *doit* être indiquée dans la description d'introduction et la page de couverture du document en renvoyant à <https://python.swaroopch.com/> et en indiquant clairement que le texte original peut être récupéré à cet emplacement.
- Sauf avis contraire, tout le code compris dans ce livre est licencié sous la licence [BSD à 3-clause](#).

Lire maintenant

Vous pouvez lire le livre en ligne à l'adresse suivante <https://python.swaroopch.com/>

Acheter le livre

Une copie papier du livre peut être achetée à l'adresse <https://www.swaroopch.com/buybook/>, pour le plaisir de lire hors-ligne, et pour soutenir le développement permanent et l'amélioration continue de ce livre.

Téléchargement

Visitez <https://www.gitbook.com/book/swaroopch/byte-of-python/details> pour accéder aux téléchargements suivants:

- [PDF \(lecture hors ligne, etc.\)](#)
- [EPUB \(pour iPhone/iPad, liseuse, etc.\)](#)
- [Mobi \(pour liseuse Kindle\)](#)

Rendez-vous sur <https://github.com/swaroopch/byte-of-python> pour accéder au contenu brut (pour suggérer des corrections, des modifications, traduire, etc.)

Lire le livre dans votre langue maternelle

Si vous souhaitez lire ou contribuer à la traduction de ce livre dans d'autres langues, consultez la section [Traductions](#).

Dédicace

À [Kalyan Varma](#) et à beaucoup d'autres seniors de [PESIT](#) qui nous ont présenté GNU/Linux et le monde de l'open source .

À la mémoire de [Atul Chitnis](#), un ami et guide qui nous manquera énormément.

Aux [pionniers qui ont créé Internet](#). Ce livre a été écrit pour la première fois en 2003. Il reste toujours populaire grâce à la nature du partage des connaissances sur Internet telle qu'envisagée par les pionniers.

Préface

Python est sans doute l'un des rares langages de programmation à être à la fois simple et puissant. C'est une bonne chose pour les débutants comme pour les experts, et plus important, c'est amusant de s'en servir. Ce livre a pour but de vous aider à apprendre ce formidable langage et de vous montrer comment faire des choses rapidement et facilement. En fait « Le Parfait Antidote à vos problèmes de programmation ».

Le public visé par ce livre

Ce livre est un guide ou un tutoriel pour le langage de programmation Python. Le public visé est les débutants. Il est également utile aux programmeurs expérimentés.

Le but est le suivant: si vos seules connaissances en informatique se limitent à enregistrer un fichier, vous pouvez apprendre Python à partir de ce livre. Si vous avez déjà une expérience en programmation, alors vous pouvez aussi apprendre Python à partir de ce livre.

Si vous avez déjà programmé, vous serez intéressé par les différences entre Python et votre langage de programmation préféré, et j'ai mis en évidence ces différences. Un avertissement cependant, Python va devenir sous peu votre langage de programmation favori!

Site officiel

Le site officiel du livre est <https://python.swaroopch.com/> où vous pouvez lire l'intégralité du livre en ligne, télécharger les dernières versions du livre, acheter un exemplaire imprimé et également m'envoyer des commentaires.

Matière à réflexion

Il existe deux manières de concevoir un logiciel: l'une consiste à le rendre si simple qu'il n'y a manifestement aucune lacune; l'autre est de le rendre si compliqué qu'il n'y a pas de lacunes évidentes. -- C. A. R. Hoare

Le succès dans la vie ne dépend pas tant du talent et des opportunités que de la concentration et de la persévérance. -- C. W. Wendte

À propos de Python

Python est un des rares langages à être à la fois *simple* et *puissant*. Vous serez agréablement surpris de voir comme il est facile de se concentrer sur la solution d'un problème plutôt que sur la syntaxe et la structure du langage utilisé .

L'introduction officielle à Python est :

Python est un langage de programmation puissant et facile à apprendre. Il possède des structures de données de haut niveau et une approche simple mais efficace à la programmation orientée objet. La syntaxe élégante de Python et le typage dynamique des données, avec la nature interprétée du langage, en font un langage idéal pour le scriptage et le développement rapide d'applications dans de nombreux domaines sur la plupart des plate-formes.

Je parlerai plus en détail de ces fonctionnalités dans le paragraphe suivant.

Histoire derrière le nom

Guido van Rossum, le créateur du langage, l'a appelé en référence à la série télévisée de la BBC « Monty Python's Flying Circus ». Il n'aime pas particulièrement les serpents qui tuent des animaux pour manger en les étouffant et les écrasant.

Fonctionnalités de Python

Simple

Python est un langage simple et minimal. Lire un bon programme écrit en Python ressemble à lire de l'anglais, mais de l'anglais très strict ! La nature de pseudo-code de Python est une de ses plus grandes forces. Cela vous permet de vous concentrer sur la solution du problème plutôt que sur le langage lui-même.

Facile à apprendre

Comme vous le constaterez, il est très facile de démarrer avec Python. Python a une syntaxe extraordinairement simple, comme indiqué précédemment.

Libre et Open Source

Python est un exemple de *FLOSS* (Free/Libre et Open Source Software). En d'autres termes, vous pouvez librement distribuer des copies de ce logiciel, lire son code source, le modifier, et en utiliser des morceaux dans un nouveau programme. FLOSS est basé sur le concept de communauté qui partage le savoir. C'est l'une des raisons pour lesquelles Python est tellement bien, il a été créé et est constamment amélioré par une communauté qui veut juste voir un meilleur Python.

Langage de haut niveau

Quand vous écrivez des programmes en Python, vous n'avez pas besoin de vous préoccuper de détails comme gérer la mémoire utilisée par votre programme, etc...

Portable

En raison de son caractère open-source, Python a été porté sur de nombreuses plate-formes. Tout programme Python peut fonctionner sur n'importe laquelle de ces plate-formes sans avoir besoin du moindre changement si vous évitez des fonctionnalités dépendant d'un système.

Vous pouvez utiliser Python sous GNU/Linux, Windows, FreeBSD, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, BeOS, OS/390, z/OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE et même PocketPC !

Vous pouvez même utiliser une plate-forme comme [Kivy](#) pour créer des jeux pour votre ordinateur et pour iPhone, iPad et Android.

Interpréte

Cela appelle quelques explications.

Un programme écrit dans un langage compilé comme C ou C++ est converti à partir du langage source (C ou C++) dans un langage parlé par l'ordinateur (code binaire avec des 0 et des 1) en utilisant un compilateur avec différents flags et options. Quand vous lancez le programme, l'éditeur de liens/le chargeur copie le programme du disque dur vers la mémoire et commence à l'exécuter.

Python, par contre, n'a pas besoin de compilation vers du code binaire. Vous *lancez* juste le programme directement à partir du code source. En interne, Python convertit le code source dans une forme intermédiaire appelée bytecode et ensuite le convertit dans le langage natif de l'ordinateur et le lance. Tout cela, en fait, rend l'utilisation de Python plus facile, vu que vous n'avez pas besoin de vous inquiéter de la compilation du programme, que les bonnes librairies sont liées et chargées, etc... Cela rend aussi les programmes Python plus portables, vu qu'il suffit de copier un programme Python d'un ordinateur sur un autre, et cela marche!

Orienté objet

Python supporte la programmation orientée procédure et la programmation orientée objet. Dans les langages *orienté-procedure*, le programme est construit autour de procédures ou fonctions qui sont des portions re-utilisables de programmes. Dans les langages *orienté-objet*, le programme est construit autour d'objets qui comprennent données et fonctionnalités. Python a une manière à la fois très puissante et très simple de faire de la Programmation Orientée Objet, en particulier comparé à des langages comme C++ ou Java.

Extensible

Si vous avez besoin qu'un morceau de code critique tourne très vite, ou qu'un algorithme reste caché, vous pouvez écrire cette partie de votre programme en C ou C++ et ensuite l'utiliser dans votre programme Python.

Embarqué

Vous pouvez embarquer Python dans vos programmes C/C++ pour donner des possibilités de *scripting* aux utilisateurs de vos programmes.

De nombreuses bibliothèques

La Python Standard Library est immense. Cela peut vous aider à faire de nombreuses choses comme les expressions régulières, la création de documentation, les tests unitaires, les threads, les bases de données, les navigateurs web, CGI, FTP, email, XML, XML-RPC, HTML, fichiers WAV, cryptographie, GUI (graphical user interfaces), Tk, et autres choses dépendant du système. Rappelez-vous, tout ceci est toujours disponible, quand Python est installé. Cela est appelé l'approche *Batteries Included* de Python.

En plus de la bibliothèque standard, il y a d'autres bibliothèques de grande qualité que vous pouvez trouver dans le [Python Package Index](#).

Récapitulatif

Python est vraiment un langage puissant et formidable. Il possède la bonne combinaison de performance et fonctionnalités, ce qui rend l'écriture de programmes Python à la fois amusante et aisée.

Python 3 par rapport à 2

Vous pouvez ignorer cette section si la différence entre *Python version 2* et *Python version 3* ne vous intéresse pas. Mais soyez conscient de la version que vous utilisez. Ce livre est écrit pour Python version 3.

N'oubliez pas qu'une fois que vous avez bien compris et appris à utiliser une version, vous pouvez facilement apprendre les différences et utiliser l'autre. Le plus difficile est d'apprendre à programmer et à comprendre les bases du langage Python lui-même. C'est notre objectif dans ce livre, et une fois que vous avez atteint cet objectif, vous pouvez facilement utiliser Python 2 ou Python 3 en fonction de votre situation.

Pour plus de détails sur les différences entre Python 2 et Python 3, voir:

- [Le futur de Python 2](#)
- [Porter du code de Python 2 vers Python 3](#)
- [Ecriture de code sous Python 2 et 3](#)
- [Prise en charge de Python 3: un guide détaillé](#)

Que disent les programmeurs

Vous trouverez intéressant de lire ce que des grands hackers comme ESR disent à propos de Python:

- *Eric S. Raymond* est l'auteur de [La Cathédrale et le Bazar](#) et aussi la personne qui a créé le terme « Open Source ». Il dit que [Python est devenu son langage de programmation préféré](#). Cet article a été la vraie inspiration pour mon premier contact avec Python.
- *Bruce Eckel* est l'auteur des livres célèbres *Thinking in Java* et *Thinking in C++*. Il dit qu'aucun langage ne l'a rendu plus productif que Python. Il dit que Python est peut-être le seul langage qui se concentre sur le fait simplifier le travail du programmeur. Lisez [l'interview complète](#) pour plus de détails.
- *Peter Norvig* est un auteur Lisp et directeur de Search Quality chez Google (merci à Guido van Rossum pour l'avoir signalé). Il dit que Python a toujours fait partie de Google. Vous pouvez vérifier cela en recherchant sur la page [Google Jobs](#) qui indique Python comme un pré-requis pour les ingénieurs logiciels.

Installation

Lorsque nous nous référons à « Python 3 » dans ce livre, nous ferons référence à toute version de Python égale ou supérieure à la version [Python 3.6.0](#).

Installation sous Windows

Visitez <https://www.python.org/downloads/> et téléchargez la dernière version. Au moment d'écrire ces lignes, c'était Python 3.6.0.

L'installation est comme n'importe quel autre logiciel basé sur Windows.

Notez que si votre version de Windows est antérieure à Vista, vous devez [télécharger Python 3.4 uniquement](#) car les versions ultérieures nécessitent des versions plus récentes de Windows.

ATTENTION: Assurez-vous de cocher l'option `Ajouter Python 3.6 au PATH`.

Pour changer l'emplacement d'installation, cliquez sur `Personnaliser l'installation`, puis sur `Suivant` et entrez `C:\python36` (ou un autre emplacement approprié) comme emplacement d'installation.

Si vous n'avez pas coché l'option `Ajouter Python 3.6 au PATH` plus tôt, cochez `Ajouter Python aux variables d'environnement`. Cela fait la même chose que `Ajouter Python 3.6 au PATH` sur le premier écran d'installation.

Vous pouvez choisir d'installer Launcher pour tous les utilisateurs ou non, peu importe. Launcher est utilisé pour basculer entre différentes versions de Python installées.

Si votre chemin n'a pas été défini correctement (en cochant les options `Ajouter Python 3.6 au PATH` ou `Ajouter Python aux variables d'environnement`), suivez les étapes décrites dans la section suivante (`Invite de commandes DOS`) pour y remédier. Sinon, allez à la section `Lancer l'interpréteur interactif Python sous Windows` de ce document.

REMARQUE: Si vous connaissez déjà la programmation et que vous êtes familiers avec Docker consultez [Python dans Docker](#) et [Docker sous Windows](#).

Invite de commandes DOS

Si vous voulez pouvoir utiliser Python à partir de la ligne de commande Windows, c'est-à-dire à l'invite de commande DOS, vous devez définir la variable PATH de manière appropriée.

Pour Windows 2000, XP, 2003, cliquez sur `Panneau de configuration -> Système -> Avancé -> Variables d'environnement`. Cliquez sur la variable nommée `PATH` dans la section `Variables système`, puis sélectionnez `Modifier` et ajoutez à la fin de ce qui est déjà là `;C:\Python36` (veuillez vérifier que ce dossier existe, il sera différent pour les versions plus récentes de Python). Bien sûr, utilisez le nom de répertoire approprié.

Pour les versions plus anciennes de Windows, ouvrez le fichier `C:\AUTOEXEC.BAT` et ajoutez la ligne `PATH=%PATH%;C:\Python36` et redémarrez le système. Pour Windows NT, utilisez le fichier `AUTOEXEC.NT`.

Pour Windows Vista:

- Cliquez sur Démarrer et choisissez `Panneau de configuration`
- Cliquez sur Système. À droite, vous verrez « Afficher les informations de base sur votre ordinateur ».
- A gauche, une liste de tâches dont la dernière est `Paramètres système avancés`. Cliquez dessus.
- L'onglet `Avancé` de la boîte de dialogue `Propriétés système` est affiché. Cliquez sur le bouton `Variables d'environnement` en bas à droite.
- Dans la zone inférieure intitulée `variables système`, faites défiler jusqu'à « PATH » et cliquez sur le bouton `Modifier`.
- Modifiez votre chemin au besoin.
- Redémarrez votre système. Sur mon ordinateur, Vista n'a pas pris en compte la modification de la variable d'environnement avant mon redémarrage.

Pour Windows 7 et 8:

- Cliquez avec le bouton droit sur Ordinateur de votre bureau et sélectionnez Propriétés ou cliquez sur Démarrer et choisissez Panneau de configuration -> Système et sécurité -> Système. Cliquez sur Paramètres système avancés sur la gauche, puis sur l'onglet Avancé. En bas, cliquez sur « Variables d'environnement » et sous « Variables système », recherchez la variable PATH, sélectionnez-la, puis appuyez sur Modifier .
- Allez à la fin de la ligne sous Valeur de la variable et ajoutez ;C:\Python36 à la fin de ce qui existe déjà (veuillez vérifier que ce dossier existe, il sera différent pour les versions les plus récentes de Python). Bien sûr, utilisez le nom de dossier approprié.
- Si la valeur était %SystemRoot%\system32; il deviendra maintenant %SystemRoot%\system32;C:\Python36
- Cliquez sur OK et vous avez terminé. Aucun redémarrage n'est requis, mais vous devrez peut-être fermer et rouvrir la ligne de commande.

Pour Windows 10:

- Menu Démarrer de Windows> Paramètres > À propos de > Informations système (tout à droite) > Paramètres système avancés > Variables d'environnement (vers le bas) > Choisissez Variable Path et cliquez sur Modifier)> New > (tapez le lieu de votre emplacement python. Par exemple, C:\Python36\)

Lancer l'interpréteur interactif Python sous Windows

Pour les utilisateurs Windows, vous pouvez exécuter l'interpréteur en ligne de commande si vous avez défini la variable PATH de manière appropriée.

Pour ouvrir le terminal sous Windows, cliquez sur le bouton Démarrer, puis sur Exécuter. Dans la boîte de dialogue, tapez cmd et appuyez sur la touche [Entrée] .

Ensuite, tapez python et assurez-vous qu'il n'y a pas d'erreur.

Installation sous Mac OS X

Pour les utilisateurs de Mac OS X, utilisez Homebrew: brew install python3 .

Pour vérifier, ouvrez le terminal en appuyant sur les touches [Commande + Espace] (pour ouvrir la recherche Spotlight), tapez Terminal et appuyez sur la touche [Entrée] . Maintenant, lancez python3 et assurez-vous qu'il n'y a pas d'erreur.

Installation sous GNU/Linux

Pour les utilisateurs GNU/Linux, utilisez le gestionnaire de paquets de votre distribution pour installer Python 3, par exemple. sur Debian et Ubuntu: sudo apt-get update && sudo apt-get install python3 .

Pour vérifier, ouvrez le terminal en ouvrant l'application Terminal ou en appuyant sur Alt + F2 et en entrant gnome-terminal . Si cela ne fonctionne pas, reportez-vous à la documentation de votre distribution GNU/Linux. Ensuite, lancez python3 et assurez-vous qu'il n'y a pas d'erreur.

Vous pouvez voir la version de Python à l'écran en lançant:

```
$ python3 -V  
Python 3.6.0
```

NOTE: \$ est l'invite du terminal. Ce sera différent pour vous selon le paramétrage du système d'exploitation de votre ordinateur, par conséquent, je vais indiquer l'invite uniquement par le symbole \$.

ATTENTION: la sortie peut être différente sur votre ordinateur, selon la version du logiciel Python installé sur votre ordinateur.

Récapitulatif

A partir de maintenant, nous supposerons que Python est installé sur votre système.

Ensuite, nous écrirons notre premier programme Python.

Premiers pas

Nous allons maintenant voir comment exécuter le traditionnel programme « Hello World » en Python. Cela vous apprendra comment écrire, enregistrer et exécuter un programme Python.

Il y a deux façons d'utiliser Python pour lancer votre programme - utiliser le l'interpréteur interactif ou un fichier source. Nous allons maintenant voir comment utiliser ces deux méthodes.

Utiliser l'interpréteur interactif

Démarrez l'interpréteur interactif en entrant `python` dans l'invite de commande.

Pour les utilisateurs Windows, vous pouvez lancer l'interpréteur interactif si vous avez modifié la variable `PATH` correctement.

Ouvrez le terminal de votre système d'exploitation (comme indiqué précédemment dans le chapitre [Installation](#)), puis ouvrez interpréteur interactif Python en tapant `python3` et en appuyant sur la touche `[Entrée]`.

Une fois que vous avez démarré Python, vous devriez voir `>>>` où vous pouvez commencer à taper des choses. Ceci s'appelle *l'invite de l'interpréteur Python*.

A l'invite de l'interpréteur Python, tapez:

```
print("Hello World")
```

suivi de la touche `[Entrée]`. Vous devriez voir les mots `Hello World` imprimés à l'écran.

Voici un exemple de ce que vous devriez voir lorsque vous utilisez un ordinateur Mac OS X. Les détails concernant le logiciel Python varieront en fonction de votre ordinateur, mais la partie à partir de l'invite (c'est-à-dire à partir de `>>>`) devrait être la même quel que soit le système d'exploitation.

```
$ python3
Python 3.6.0 (default, Jan 12 2017, 11:26:36)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
```

Notez que Python vous donne la sortie de la ligne immédiatement! Ce que vous venez de saisir est une seule *instruction* Python. Nous utilisons `print` pour (sans surprise) imprimer toute valeur que nous lui fournissons. Ici, nous fournissons le texte `Hello World` et celui-ci est rapidement imprimé à l'écran.

Comment quitter l'interpréteur interactif

Si vous utilisez un terminal GNU/Linux ou OS X, vous pouvez quitter interpréteur interactif en appuyant sur `[Ctrl + d]` ou en entrant `exit ()` (note: n'oubliez pas d'inclure les parenthèses, `()`), puis en appuyant sur `[Entrée]`.

Si vous utilisez l'invite de commande Windows, appuyez sur les touches `[Ctrl + z]` puis sur `[Entrée]`.

Choisir un éditeur

Nous ne pouvons pas taper nos programmes dans l'interpréteur interactif à chaque fois que nous voulons exécuter quelque chose. Nous devons donc les enregistrer dans des fichiers et pouvoir exécuter nos programmes autant de fois que nécessaire.

Pour créer nos fichiers source Python, nous avons besoin d'un logiciel d'édition où vous pouvez taper et enregistrer. Un bon éditeur de programmeur vous facilitera la tâche d'écrire les fichiers source. Par conséquent, le choix d'un éditeur est crucial. Vous devez choisir un éditeur comme vous choisiriez une voiture que vous achèteriez. Un bon éditeur vous aidera à écrire facilement des programmes Python, ce qui rendra votre voyage plus confortable et vous aidera à atteindre votre destination de manière beaucoup plus rapide et plus sûre.

Une des exigences de base est la *coloration syntaxique*, où toutes les différentes parties de votre programme Python sont colorisées afin que vous puissiez voir votre programme et visualiser son exécution.

Si vous ne savez pas par où commencer, je vous recommanderais d'utiliser le logiciel [PyCharm Educational Edition](#) disponible sous Windows, Mac OS X et GNU/Linux. Détails dans la section suivante.

Si vous utilisez Windows, *n'utilisez pas Notepad*. C'est un mauvais choix car il ne dispose pas de coloration syntaxique et, surtout, il ne prend pas en charge l'indentation du texte, ce qui est très important dans notre cas, comme nous le verrons plus tard. Les bons éditeurs le feront automatiquement.

Si vous êtes un programmeur expérimenté, vous devez déjà utiliser [Vim](#) ou [Emacs](#). Il va sans dire que ce sont deux des éditeurs les plus puissants et qu'il vous sera utile de les utiliser pour écrire vos programmes Python. Personnellement, j'utilise les deux pour la plupart de mes programmes et j'ai même écrit un [livre entier sur Vim](#).

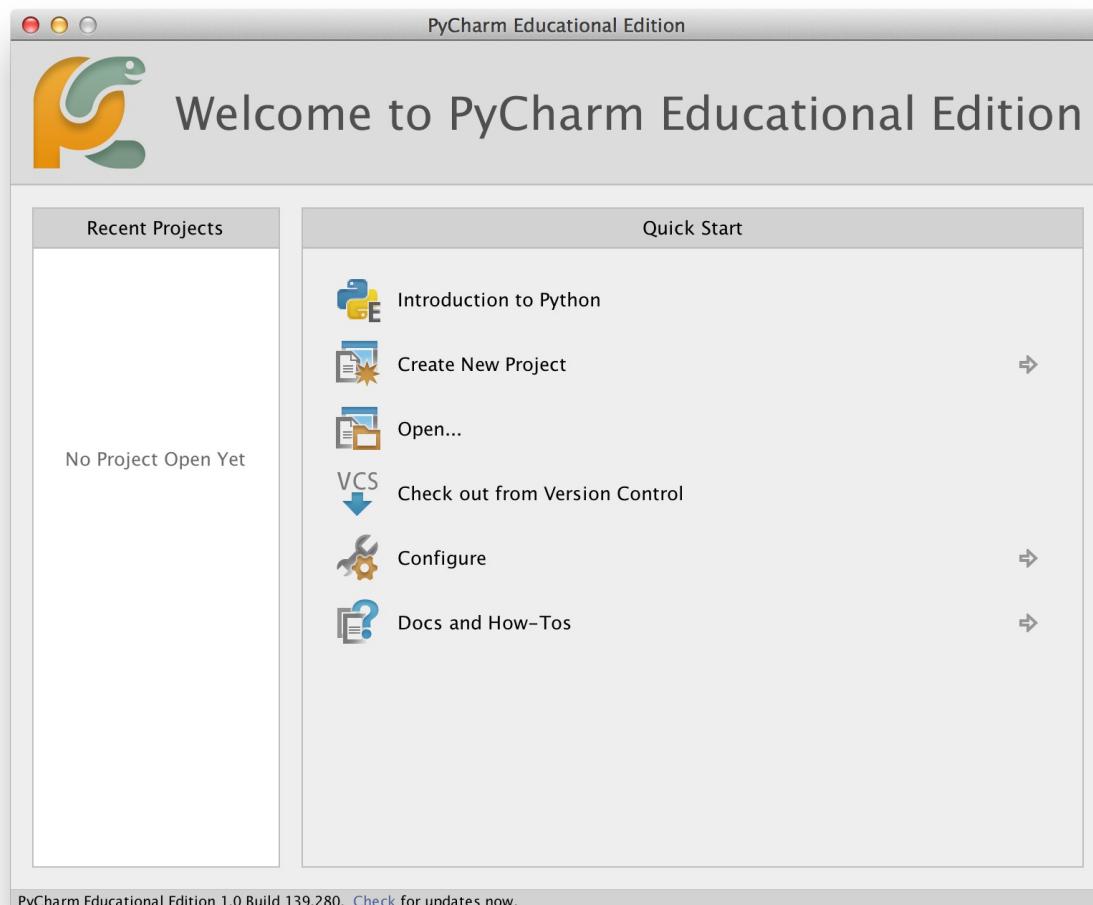
Si vous êtes prêt à consacrer du temps à apprendre Vim ou Emacs, je vous recommande fortement d'apprendre à les utiliser, car cela vous sera très utile à long terme. Cependant, comme je l'ai déjà mentionné, les débutants peuvent commencer par PyCharm et se concentrer sur l'apprentissage en Python plutôt que sur celui de leur éditeur en ce moment.

Pour réitérer, choisissez un éditeur approprié. L'écriture de programmes Python en sera d'autant plus amusante et plus facile.

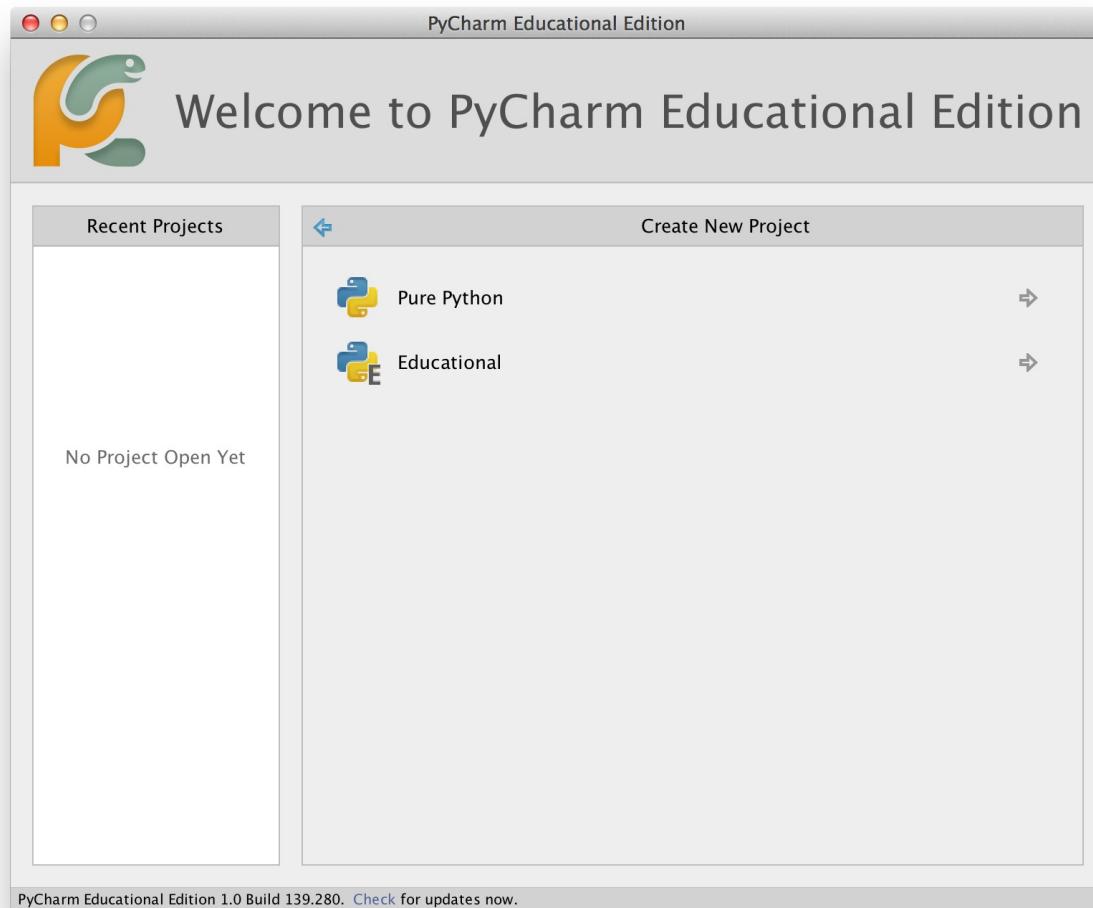
PyCharm

[PyCharm Educational Edition](#) est un éditeur gratuit que vous pouvez utiliser pour écrire des programmes Python.

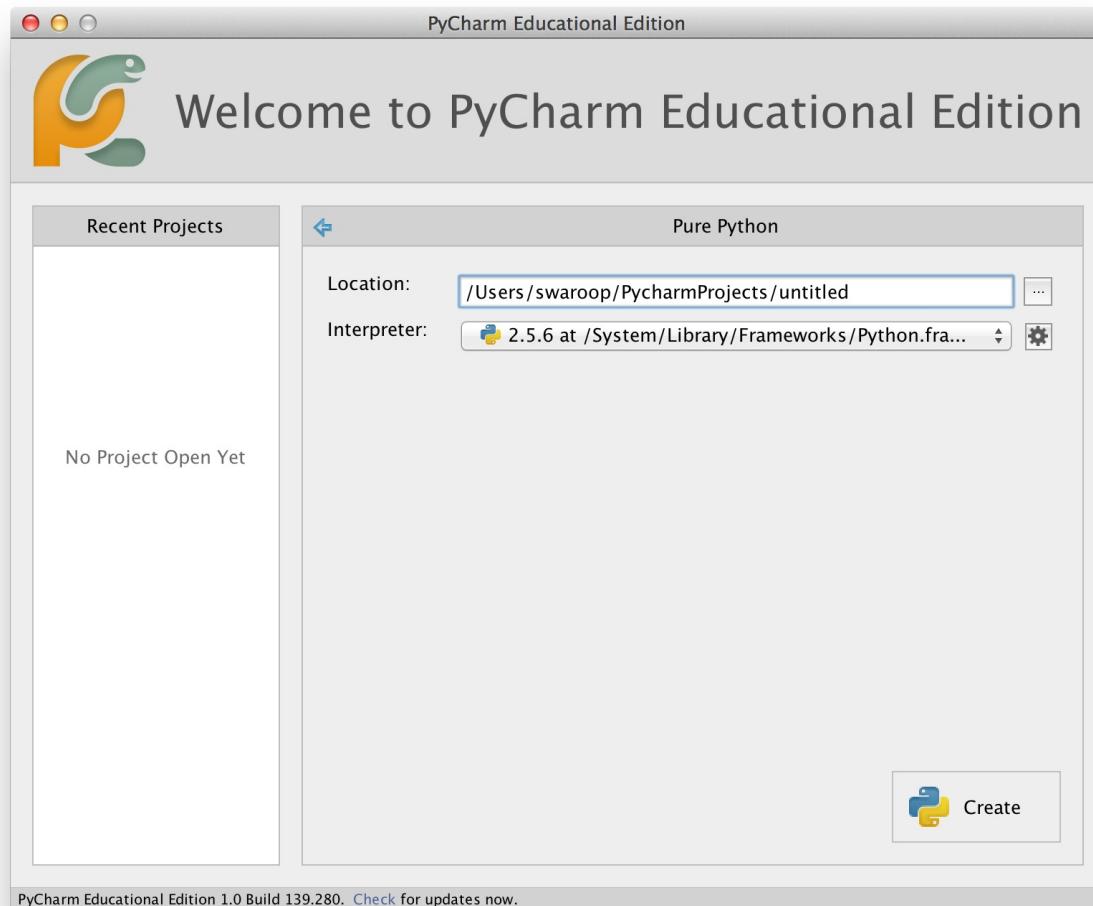
Lorsque vous ouvrez PyCharm, vous verrez ceci, cliquez sur `Create New Project` :



Sélectionnez Pure Python :

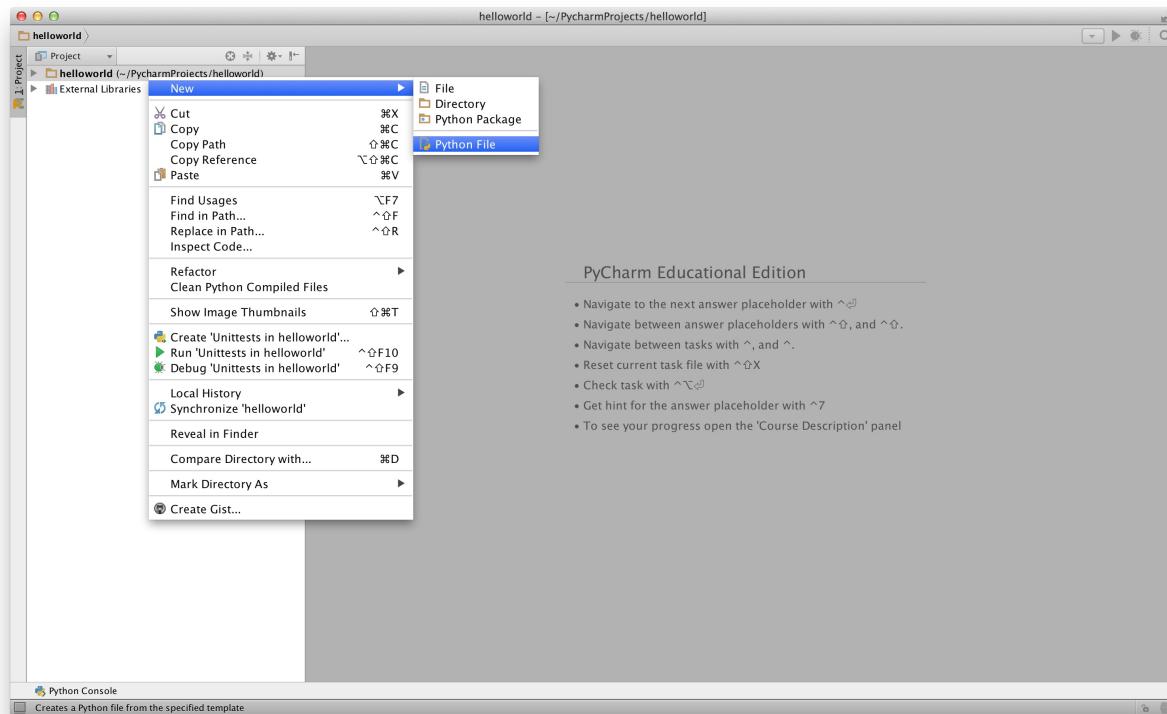


Changez `untitled` en `helloworld` comme emplacement du projet, vous devriez voir des détails similaires à ceux-ci:

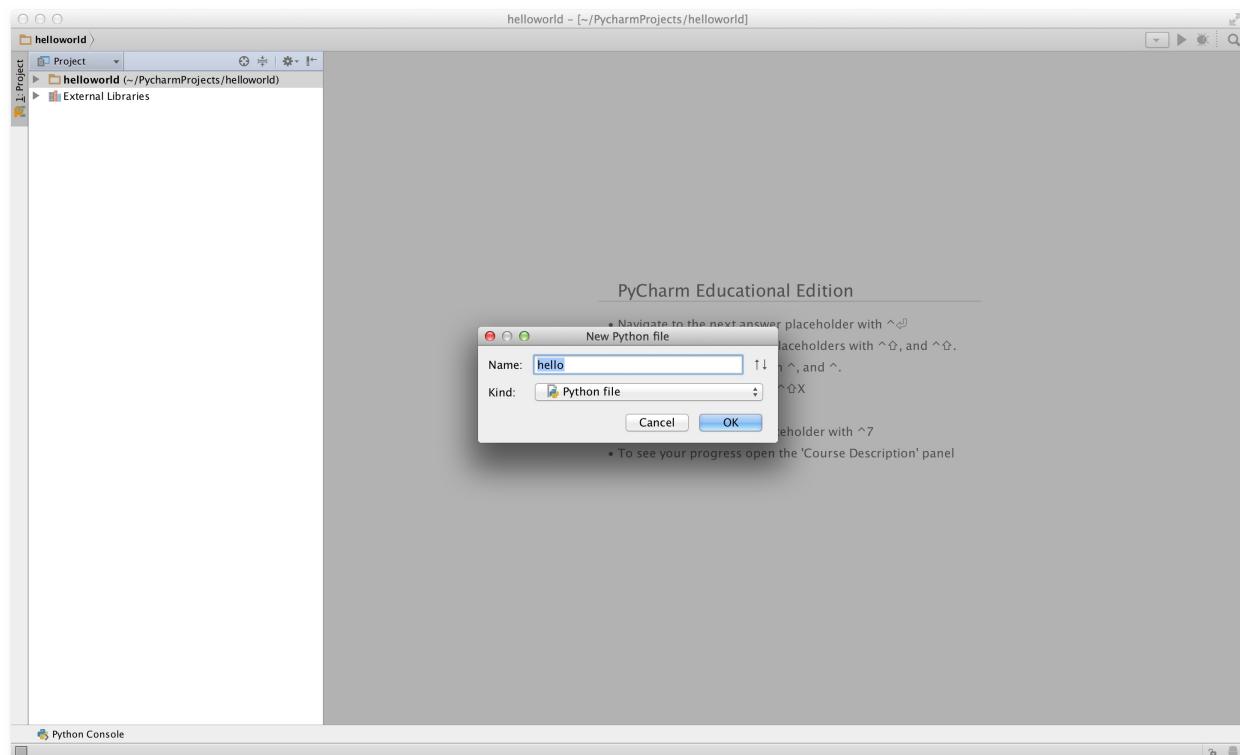


Cliquez sur le bouton `Create`.

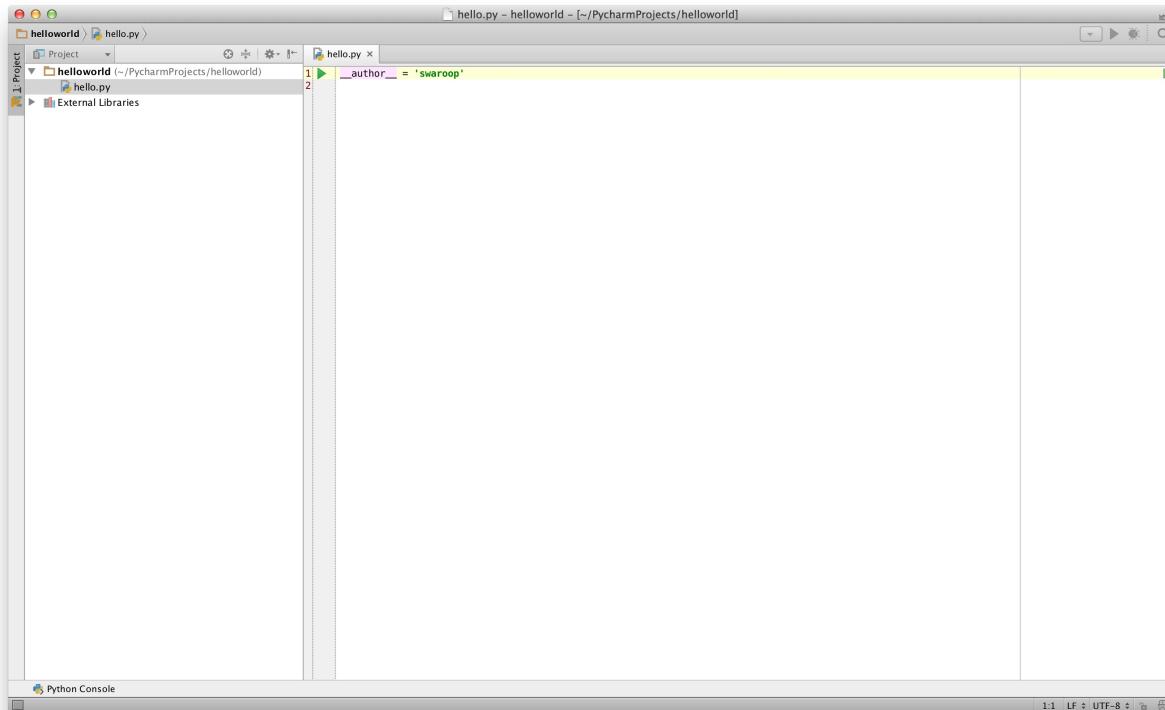
Cliquez avec le bouton droit de la souris sur le `helloworld` dans la barre latérale et sélectionnez `New -> Python File`:



Il vous sera demandé de taper le nom, tapez `hello` :



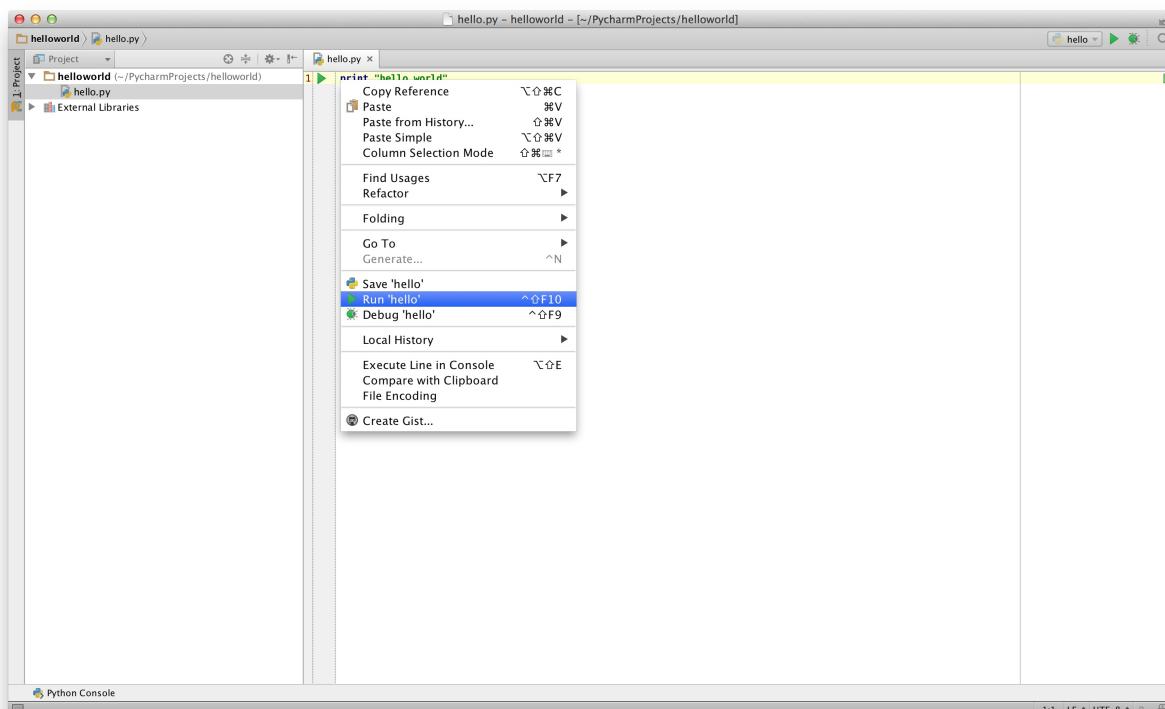
Vous pouvez maintenant voir un fichier ouvert pour vous:



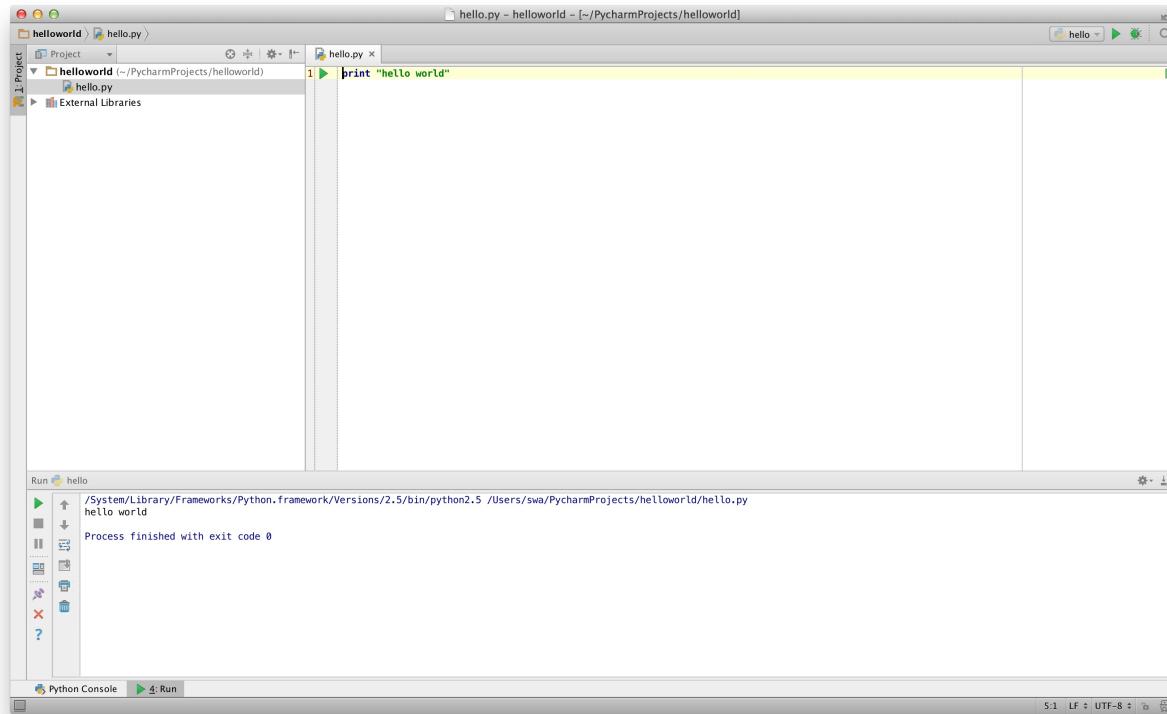
Supprimez les lignes déjà présentes et tapez maintenant le texte suivant:

```
print("hello world")
```

Maintenant, faites un clic droit sur ce que vous avez tapé (sans sélectionner le texte), puis cliquez sur `Run 'hello'`.



Vous devriez maintenant voir la sortie (ce qu'elle affiche) de votre programme:



Phew! Ça fait beaucoup d'étapes pour commencer, mais désormais, chaque fois que nous vous demandons de créer un nouveau fichier, n'oubliez pas de cliquer avec le bouton droit de la souris sur `helloworld` à gauche -> `New -> Python File` et de continuer. Suivez les mêmes étapes pour taper et exécuter comme indiqué ci-dessus.

Pour plus d'informations sur PyCharm, consultez la page [Démarrage rapide PyCharm](#).

Vim

1. Installez Vim

- Les utilisateurs de Mac OS X doivent installer le paquet `macvim` via [HomeBrew](#)
- Les utilisateurs Windows doivent télécharger l'*installateur executable* depuis le [site Web Vim](#).
- Les utilisateurs de GNU/Linux doivent obtenir Vim à partir du gestionnaire de paquets de leur distribution, par exemple, les utilisateurs de Debian et Ubuntu peuvent installer le paquet `vim` .

2. Installez le plug-in `jedi-vim` pour l'auto-complétion.

3. Installez le paquet python `jedi` correspondant: `pip install -U jedi`

Emacs

1. Installez [Emacs 24+](#).

- Les utilisateurs de Mac OS X doivent obtenir Emacs à l'adresse <http://emacsformacosx.com>
- Les utilisateurs de Windows doivent obtenir Emacs à l'adresse <http://ftp.gnu.org/gnu/emacs/windows/>
- Les utilisateurs de GNU/Linux doivent obtenir Emacs à partir des gestionnaires de paquets de leur distribution. Par exemple, les utilisateurs de Debian et Ubuntu peuvent installer le paquet `emacs24` .

2. Installez [ELPY](#)

Utiliser un fichier source

Revenons à la programmation. La tradition veut que, lors de l'apprentissage d'un nouveau langage, le premier programme écrit et lancé est le programme « Hello World ». Tout ce qu'il fait est d'afficher « Hello World » quand vous le lancez. Comme le dit Simon Cozens (l'auteur de l'incroyable livre « Beginning Perl »), c'est la « traditionnelle incantation aux dieux de la programmation pour vous aider à mieux apprendre le langage ».

Lancez l'éditeur de votre choix, saisissez le programme suivant et enregistrez-le en tant que `hello.py`

Si vous utilisez PyCharm, nous avons déjà [vu comment lancer un fichier source](#).

Pour les autres éditeurs, ouvrez un nouveau fichier `hello.py` et tapez ceci:

```
print("hello world")
```

Où devriez-vous sauvegarder le fichier? Dans n'importe quel dossier dont vous connaissez l'emplacement. Si vous ne comprenez pas ce que cela signifie, créez un nouveau dossier et utilisez cet emplacement pour enregistrer et exécuter tous vos programmes Python:

- `/tmp/py` sous Mac OS X
- `/tmp/py` sous GNU/Linux
- `C:\py` sous Windows

Pour créer le dossier ci-dessus (pour le système d'exploitation que vous utilisez), utilisez la commande `mkdir` dans un terminal, par exemple, `mkdir /tmp/py`.

IMPORTANT: veillez toujours à attribuer l'extension de fichier `.py`, par exemple, `foo.py`.

Pour exécuter votre programme Python:

1. Ouvrez une fenêtre de terminal (voir le chapitre précédent [Installation](#) pour savoir comment procéder).
2. Déplacez vous dans le répertoire où vous avez enregistré le fichier, par exemple, `cd /tmp/py`. En anglais `cd` est la sigle de **Change Directory**.
3. Exécutez le programme en entrant la commande `python hello.py`. La sortie est comme ci-dessous.

```
$ python hello.py
hello world
```

```
[20:33:32] [/tmp/py]
$ python hello.py
hello world

[20:33:36] [/tmp/py]
$ █
```

Si vous obtenez le résultat ci-dessus, félicitations! - vous avez exécuté avec succès votre premier programme Python. Vous avez traversé avec succès la partie la plus difficile de l'apprentissage de la programmation, à savoir, lancer votre premier programme!

Au cas où vous auriez une erreur, veuillez taper votre programme *exactement* comme indiqué ci-dessus et relancez le. Notez que Python est sensible à la casse, c'est-à-dire que `print` n'est pas identique à `Print` (notez la minuscule dans le premier et la majuscule dans le second). Assurez-vous également qu'il n'y a pas d'espaces ni de tabulations avant le premier caractère de chaque ligne - nous verrons plus tard pourquoi *cela est important*.

Comment ça marche

Un programme Python est composé d'*instructions*. Dans notre premier programme, nous n'avons qu'une instruction. Dans cette instruction, nous appelons l'*instruction* `print` à laquelle nous fournissons le texte « `hello world` ».

Obtenir de l'aide

Si vous besoin rapidement d'information sur n'importe quelle fonction ou instruction dans Python, alors vous pouvez utiliser la fonctionnalité `help`. Cela est très utile particulièrement quand on utilise l'interpréteur interactif. Par exemple, tapez `help(print)` - cela affiche l'aide pour la fonction `print` qui est utilisée pour afficher des choses à l'écran.

Note: Tapez `q` pour sortir de l'aide.

De la même manière, vous pouvez obtenir de l'information sur pratiquement n'importe quoi dans Python. Utilisez `help()` pour apprendre plus de choses sur le `help` lui-même !

Pour avoir de l'aide sur les opérateurs comme `return`, il faut mettre des quotes à l'intérieur comme dans `help('return')` pour que Python comprenne ce que l'on veut faire.

Récapitulatif

Vous devriez être maintenant capable d'écrire, enregistrer et exécuter facilement des programmes Python.

Maintenant que vous êtes un utilisateur Python, apprenons quelques concepts Python supplémentaires.

Les Bases

Afficher seulement « Hello World » n'est pas assez, n'est-ce pas ? Vous voulez faire plus que cela - vous voulez prendre des choses en entrée, les manipuler et en sortir quelque chose. Nous pouvons réussir cela avec Python en utilisant des constantes et des variables.

Commentaires

Les *commentaires* sont un texte quelconque à droite du symbole `#` et sont principalement utiles en tant que notes pour le lecteur du programme. Par exemple:

```
print('hello world') # Remarquez que print est une fonction
```

ou:

```
# Remarquez que print est une fonction
print('hello world')
```

Utilisez autant de commentaires utiles que possible dans votre programme pour:

- expliquer vos hypothèses
- expliquer les décisions importantes
- expliquer les détails importants
- expliquer les problèmes que vous essayez de résoudre
- expliquer les problèmes que vous essayez de surmonter dans votre programme, etc.

Le code vous dit comment, les commentaires devraient vous dire pourquoi.

Ceci est utile pour les lecteurs de votre programme afin qu'ils puissent facilement comprendre ce qu'il fait. Rappelez-vous que cette personne peut être vous-même six mois plus tard!

Constantes littérales

Un exemple d'une constante littérale est un nombre comme `5`, `1.23`, `9.25e-3` ou une chaîne de caractères comme `'Ceci est une chaîne'` ou `"C'est une chaîne!"`.

Cela s'appelle un littéral parce que c'est *littéral* - vous pouvez utiliser la valeur littéralement. Le nombre `2` représente toujours lui-même et rien d'autre - c'est une *constante* parce que sa valeur ne peut être changée. Donc, on se réfère à toutes ces valeurs en tant que constantes littérales.

Nombres

Les nombres dans Python sont principalement divisés en deux types - entiers, nombres en virgule flottante.

- Un exemple d'un entier est `2` qui est juste un nombre entier.
- Des exemples de nombres en virgule flottante (ou *floats* en abrégé) sont `3.23` et `52.3E-4`. La notation `E` indique des puissances de 10. Dans ce cas, `52.3E-4` signifie `52.3 * 10^-4`.

Note pour les programmeurs expérimentés

Il n'y a pas de type « long int » à part. Le type entier par défaut peut être une valeur de n'importe quelle longueur.

Chaînes de caractères

Une chaîne de caractères est une *suite de caractères*. Les chaînes de caractères sont juste un groupe de mots.

Vous utiliserez des chaînes de caractères dans pratiquement chaque programme que vous écrirez, donc soyez attentifs à ce qui suit.

Guillemets simples

Vous pouvez définir des chaînes de caractères en utilisant des guillemets simples comme 'Je suis entre guillemets' .

Tous les whitespaces, par exemple les espaces et des tabulations, sont gardés tels quels.

Doubles guillemets

Les chaînes de caractères entre doubles guillemets fonctionnent exactement de la même manière qu'avec les guillemets simples. Un exemple est "Comment tu t'appelles?"

Triples guillemets

Vous pouvez déclarer les chaînes de caractères sur plusieurs lignes en utilisant des triples guillemets - (""" ou '''). Vous pouvez utiliser des guillemets simples et des doubles guillemets librement à l'intérieur des triples guillemets. Un exemple est:

```
'''Ceci est une chaîne multi-lignes. Ceci est la première ligne.  
Ceci est la deuxième ligne.  
«&nbsp;Quel est votre nom?&nbsp;», Ai-je demandé.  
Il a dit «&nbsp;Bond, James Bond.&nbsp;»  
'''
```

Les chaînes de caractères sont immuables

Cela signifie que quand vous avez créé une chaîne, vous ne pouvez pas la changer. Bien que cela semble être une mauvaise chose, ce n'est pas le cas. Nous verrons pourquoi cela n'est pas une limitation dans les différents programmes à venir.

Note pour les programmeurs C/C++

Il n'y a pas de type de données `char` dans Python. Cela n'est pas vraiment nécessaire, et sûr qu'il ne vous manquera pas.

Note pour les programmeurs Perl/PHP

Rappelez-vous que les chaînes entre guillemets simples et les chaînes à guillemets doubles sont équivalentes. Elles ne diffèrent en aucune manière.

La méthode format

Nous avons parfois besoin de fabriquer des chaînes de caractères à partir d'autres informations. Dans ce cas la méthode `format()` est utile.

Save the following lines as a file `str_format.py` :

```
age = 20  
name = 'Swaroop'  
  
print('{0} avait {1} ans quand il a écrit ce livre'.format(name, age))  
print('Pourquoi {0} joue avec ce python?'.format(name))
```

Résultat:

```
$ python str_format.py  
Swaroop avait 20 ans quand il a écrit ce livre  
Pourquoi Swaroop joue avec ce python?
```

Comment cela marche

Une chaîne de caractères peut utiliser certaines spécifications et par la suite, la méthode "format" peut être appelée pour remplacer ces spécifications avec les arguments correspondants à la méthode `format`.

Observez le premier usage dans lequel nous utilisons `{0}` et cela correspond à la variable `name` qui est le premier argument de la méthode `format`. De la même manière, la deuxième spécification est `{1}` qui correspond à `age` qui est le deuxième argument de la méthode `format`.

Notez que nous pouvons arriver au même résultat en utilisant la concaténation de chaînes de caractères

```
name + ' a ' + str(age) + ' ans'
```

mais notez comme c'est plus laid et sujet à erreur. Ensuite, la conversion en chaîne de caractères serait faite automatiquement par la méthode `format` au lieu de la conversion explicite ici. Enfin, en utilisant la méthode `format`, nous pouvons changer le message sans avoir à s'occuper des variables utilisées et vice-versa.

Notez également que les numéros sont facultatifs, vous auriez donc pu écrire:

```
age = 20
name = 'Swaroop'

print('{} avait {} ans quand il a écrit ce livre'.format(name, age))
print('Pourquoi {} joue avec ce python?'.format(name))
```

qui donnera exactement le même résultat que le programme précédent.

Nous pouvons aussi nommer les paramètres:

```
age = 20
name = 'Swaroop'

print('{name} avait {age} ans quand il a écrit ce livre'.format(name=name, age=age))
print('Pourquoi {name} joue avec ce python?'.format(name=name))
```

qui donnera également le même résultat que le programme précédent.

Python 3.6 a introduit un moyen plus court de nommer les paramètres, appelé « f-strings »:

```
age = 20
name = 'Swaroop'

print(f'{name} avait {age} ans quand il a écrit ce livre') # remarquez le 'f' devant la chaîne de caractères
print(f'Pourquoi {name} joue avec ce python?') # remarquez le 'f' devant la chaîne de caractères
```

qui donnera encore le même résultat que le programme précédent.

La méthode `format` de Python substitue chaque valeur d'argument. Il peut y avoir des spécifications détaillées comme :

```
# décimal (.) avec une précision de 3 pour float '0.333'
print('{0:.3f}'.format(1.0/3))
# compléter avec des underscores (_) le texte centré (^) avec
# une largeur de 11 '__hello__'
print('{0:_^11}'.format('hello'))
# basé sur un mot-clé
print('{name} a écrit {book}'.format(name='Swaroop', book='A Byte of Python'))
```

donne :

```
0.333
__hello__
Swaroop a écrit A Byte of Python
```

Puisque nous discutons formatage, notez que `print` se termine toujours par un caractère invisible de « nouvelle ligne » (`\n`), de sorte que des appels répétés à `print` s'imprimeront sur des lignes distinctes. Pour empêcher l'impression de ce caractère de nouvelle ligne, vous pouvez spécifier vouloir finir avec une chaîne vide:

```
print('a', end=' ')
print('b', end=' ')
```

La sortie est:

```
ab
```

Ou vous pouvez terminer avec un espace:

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

La sortie est:

```
a b c
```

Caractère d'échappement et de contrôle

Supposons, vous voulez avoir une chaîne de caractères contenant un simple guillemet (''), comment faire? Par exemple, la chaîne de caractères `'Comment t'appelles-tu?'`. Vous ne pouvez pas déclarer `'Comment t'appelles-tu?'` parce que Python sera perdu, où commence et se termine la chaîne de caractères? Donc, il vous faut indiquer que ce simple guillemet n'indique pas la fin de la chaîne. Cela peut être fait avec l'aide de ce qu'on appelle un *caractère d'échappement*. Vous indiquez le simple guillemet comme `\'` - notez le *backslash* (barre oblique inversée). Maintenant, vous pouvez indiquer la chaîne de caractères comme `'Comment t\'appelles-tu?'`.

Une autre façon de faire serait `"Comment t'appelles-tu?"` c'est à dire utiliser des double guillemets. De la même manière, vous devez utiliser un *caractère d'échappement* pour utiliser une double guillemet elle-même dans une chaîne de caractères définie par des double guillemets. Pour indiquer le backslash, vous devez utiliser le *backslash* sur lui-même `\\"`.

Comment faire pour indiquer une chaîne de caractère sur deux lignes? Une façon est d'utiliser une chaîne de caractère entre des triple guillemets comme montré [précédemment](#) ou vous pouvez utiliser un *caractère de contrôle* comme le caractère `\n` pour indiquer le début d'une nouvelle ligne. Un exemple est:

```
'Ceci est la première ligne\nCeci est la deuxième ligne'
```

Un autre *caractère de contrôle* utile à connaître est la tabulation - `\t`. Il y a beaucoup d'autres *caractère de contrôle* mais j'ai seulement mentionné les plus utiles ici.

Une chose à noter est que, dans une chaîne de caractères, un backslash unique à la fin de la ligne indique que la chaîne de caractères continue à la ligne suivante, mais une nouvelle ligne n'est pas ajoutée, par exemple:

```
"Ceci est la première phrase. \
Ceci est la deuxième phrase."
```

est équivalent à

```
"Ceci est la première phrase. Ceci est la deuxième phrase."
```

Chaînes « brutes »

Si vous devez spécifier des chaînes pour lesquelles aucun traitement spécial, tel que les caractères d'échappement et de contrôle, ne sont traités, vous devez spécifier une chaîne *brute (raw)* en préfixant `r` ou `R` à la chaîne. Un exemple est:

```
r"Les retours à la ligne sont indiqués par \n"
```

Remarque pour les utilisateurs d'expressions régulières

Utilisez toujours des chaînes brutes lorsque vous utilisez des expressions régulières. Sinon, vous aurez besoin de beaucoup de caractères d'échappements. Par exemple, les références arrières peuvent être appelées `'\\1'` ou `r'\1'`.

Variables

Utiliser seulement des constantes littérales peut rapidement devenir ennuyeux - nous avons besoin de solutions pour ranger n'importe quelles informations et les manipuler. C'est là que les *variables* interviennent. Les variables sont exactement ce que leur nom implique - leur valeur peut changer, vous pouvez stocker n'importe quoi avec une variable. Les variables sont juste des endroits où vous rangez de l'information dans la mémoire de l'ordinateur. Contrairement aux constantes littérales, vous avez besoin d'une méthode pour accéder à ces variables et donc vous leur donnez des noms.

Nommage d'identifiants

Les variables sont des exemples d'identifiants. Les *identifiants* sont des noms donnés pour identifier *quelque chose*. Vous devez respecter quelques règles pour donner un nom aux identifiants:

- Le premier caractère de l'identifier doit être une lettre de l'alphabet (majuscule ASCII ou minuscule ASCII ou caractère Unicode) ou un underscore ('_').
- Le reste du nom de l'identifier peut être composé de lettres (majuscules ASCII ou minuscules ASCII ou caractère Unicode), underscores ('_') ou chiffres (0-9).
- Les noms des Identifiers sont sensibles à la casse. Par exemple, `myname` et `myName` ne sont pas identiques. Notez la minuscule `n` dans le premier cas et la majuscule `N` ensuite.
- Des exemples de noms d'identifiants valides sont `i`, `name_2_3`. Des exemples invalides sont `2things`, `this is spaced out`, `my-name`, et `>a1b2_c3`.

Types de données

Les variables peuvent contenir des valeurs de différents types appelés *types de données*. Les types de base sont les nombres et les chaînes de caractères, dont nous avons déjà parlé. Dans les chapitres suivants, nous verrons comment créer nos propres types de données en utilisant les [classes](#).

Objets

Rappelez-vous, Python fait référence à tout ce qui est utilisé dans un programme en tant que *objet*. Cela est compris dans le sens générique. Au lieu de dire « le *quelque chose* », nous disons « l'*objet* ».

Note pour les utilisateurs de la Programmation Orientée Objet

Python est fortement orientée objet, dans le sens que tout est un objet, en incluant les nombres, chaînes de caractères et fonctions.

Nous allons voir comment utiliser des variables avec des constantes littérales. Enregistrez l'exemple suivant et lancez le programme.

Comment écrire des programmes Python

Désormais, la procédure standard pour enregistrer et exécuter un programme Python est la suivante:

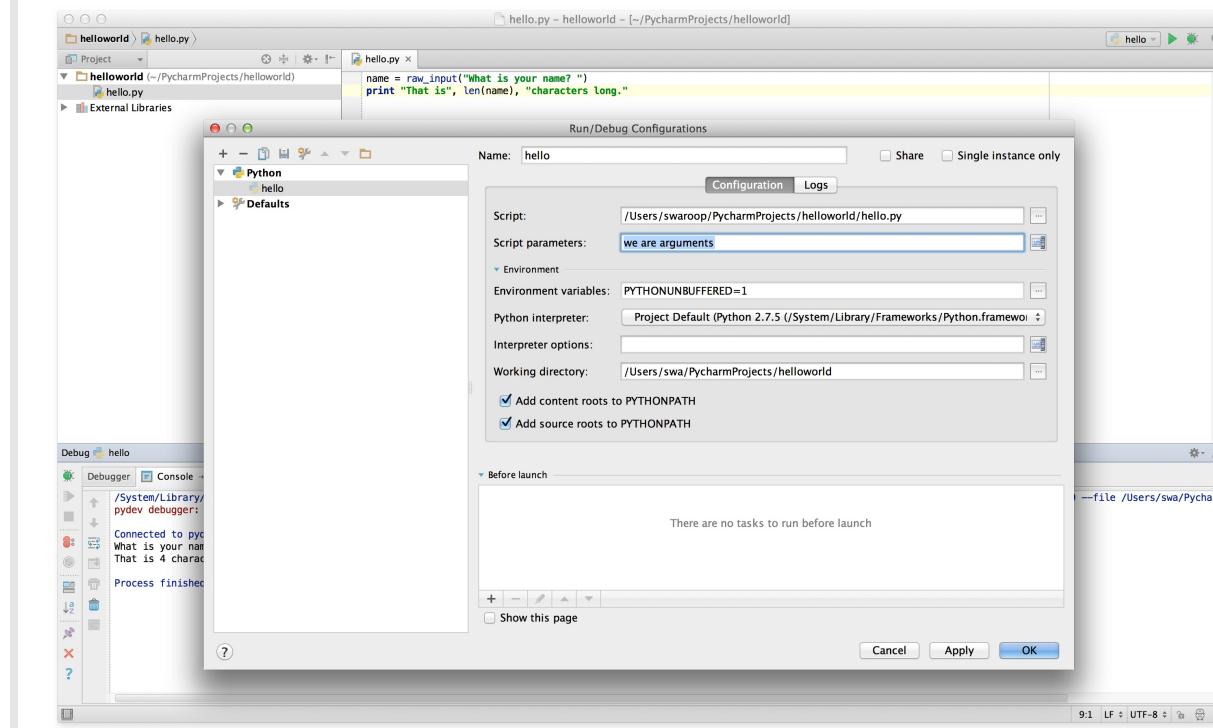
Pour PyCharm

1. Ouvrez [PyCharm](#).
2. Créez un nouveau fichier avec le nom de fichier mentionné.
3. Tapez le code du programme donné dans l'exemple.
4. Cliquez avec le bouton droit et exécutez le fichier actuel.

Note

Si vous devez fournir des [arguments de ligne de commande](#).

Cliquez sur `Run -> Edit Configurations` et tapez les arguments dans la section `Script parameters:` puis cliquez sur le bouton `OK`:



Pour les autres éditeurs

1. Ouvrez votre éditeur.
2. Tapez le code du programme donné dans l'exemple.
3. Enregistrez-le dans un fichier avec le nom de fichier mentionné.
4. Exécutez l'interpréteur avec la commande `python program.py` pour exécuter le programme.

Un exemple: utiliser des variables et des constantes littérales

Tapez et exécutez le programme suivant:

```
# Filename : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''Voici une chaîne multi-ligne.
Voici la deuxième ligne.'''
print(s)
```

Résultat:

```
5
6
Voici une chaîne multi-ligne.
Voici la deuxième ligne.
```

Comment ça marche

D'abord, nous affectons la valeur littérale constante `5` à la variable `i` en utilisant l'opérateur d'affectation (`=`). Cette ligne est appelée une instruction parce qu'elle indique qu'une action doit être faite, et dans ce cas, nous connectons la variable nommée `i` à la valeur `5`. Ensuite, nous affichons la valeur de `i` en utilisant l'instruction `print` qui, sans surprise, affiche juste la valeur de la variable à l'écran.

Ensuite nous ajoutons `1` à la valeur stockée dans `i` et nous la stockons à nouveau. Nous l'affichons et comme prévu, nous obtenons la valeur `6`.

De la même manière, nous affectons la chaîne littérale à la variable `s` et nous l'affichons.

Note pour les programmeurs dans des langages statiques

Les variables sont créées en leur donnant juste une valeur. Il n'y a pas de déclaration ou de définition de type de données.

Lignes physiques et lignes logiques

Une ligne physique est ce que vous voyez quand vous écrivez le programme. Une ligne logique est ce que *Python voit* comme une seule instruction. Python suppose implicitement que chaque *ligne physique* correspond à une *ligne logique*.

Un exemple d'une ligne logique est une instruction comme `print('Hello World')` - si c'était une ligne (comme vous la voyez dans un éditeur), alors cela correspondrait aussi à une ligne physique.

Implicitement, Python encourage l'utilisation d'une seule instruction par ligne, ce qui rend le code plus lisible.

Si vous voulez spécifier plus d'une ligne logique sur une seule ligne physique, alors il vous faut l'indiquer avec un point-virgule (`;`) qui indique la fin d'une ligne/instruction logique. Par exemple:

```
i = 5
print(i)
```

est la même chose que

```
i = 5;
print(i);
```

et peut être écrit

```
i = 5; print(i);
```

ou même

```
i = 5; print(i)
```

Cependant, je recommande fortement que vous continuiez à écrire une seule ligne logique dans seulement une seule ligne physique. Utilisez plus d'une ligne physique pour une seule ligne logique seulement si la ligne logique est vraiment longue. L'idée est d'éviter le point-virgule autant que possible vu que cela rend le code moins lisible. En fait, je n'ai jamais utilisé ou même vu un point virgule dans un programme Python.

Un exemple d'une ligne logique s'étendant sur plusieurs lignes physiques suit. Cela s'appelle *jonction de ligne explicite*.

```
s = 'Ceci est une chaîne de caractères. \
Ceci continue la chaîne.'
print(s)
```

Cela donne l'affichage :

```
Ceci est une chaîne de caractères. Ceci continue la chaîne.
```

De la même manière,

```
i = \
5
```

est la même chose que

```
i = 5
```

Parfois, il y a une supposition implicite quand vous n'avez pas besoin d'utiliser un backslash. C'est le cas quand les lignes logiques utilisent des parenthèses, entre crochets ou accolades. Cela s'appelle *jonction de ligne implicite*. Vous pouvez voir cela en action quand nous écrirons des programmes utilisant [des listes](#) dans les chapitres suivants.

Indentation

Les espaces sont importants dans Python. En fait *les espaces au début de la ligne sont importants*. Cela s'appelle *l'indentation*. Les espaces (espaces et tabulations) au début de la ligne logique sont utilisés pour déterminer le niveau d'indentation de la ligne logique, qui est à son tour utilisée pour déterminer le regroupement des instructions.

Cela signifie que les instructions qui vont ensemble doivent avoir la même indentation. Chaque jeu d'instructions est appelé un *bloc*. Nous verrons des exemples de l'importance des blocs dans les chapitres suivants.

Un chose à retenir est qu'une fausse indentation va mener à des erreurs. Par exemple:

```
i = 5
print('La valeur est ', i) # Erreur! Notez l'espace en début de ligne
print('Je repète, la valeur est ', i)
```

A l'exécution, vous obtenez l'erreur suivante :

```
File "whitespace.py", line 4
    print('La valeur est ', i) # Erreur! Notez l'espace en début de ligne
    ^
IndentationError: unexpected indent
```

Notez qu'il y a un espace au début de la deuxième ligne. L'erreur indiquée par Python nous dit que la syntaxe est invalide, c'est-à-dire que le programme n'est pas correctement écrit. Cela vous dit que *vous ne pouvez pas commencer des nouveaux blocs n'importe où* (à part pour le bloc principal par défaut que vous avez constamment utilisé, bien sûr). Les cas dans lesquels vous pouvez utiliser des nouveaux blocs seront détaillés dans les chapitres suivants comme le chapitre sur le [contrôle de flux](#)..

Comment indenter

Utilisez quatre espaces pour indenter. C'est la recommandation officielle du langage Python. Les bons éditeurs le feront automatiquement pour vous. Soyez sûr d'utiliser un nombre cohérent d'espaces pour indenter, sinon votre programme ne fonctionnera pas, ou aura des comportements inattendus.

Note pour les programmeurs en langage statique

Python utilisera l'indentation pour les blocs et n'utilisera jamais les accolades. Lancez `from __future__ import braces` pour en savoir plus.

Récapitulatif

Maintenant que nous avons vu les détails essentiels, nous pouvons passer à des choses plus intéressantes comme les instructions de contrôle de flux. Soyez certains d'être à l'aise avec les notions de ce chapitre.

Opérateurs et expressions

La plupart des instructions (lignes logiques) que vous écrirez contiendront *des expressions*. Un exemple simple d'une expression est `2 + 3`. Une expression peut être décomposée en opérateurs et opérandes.

Les *Opérateurs* sont la fonctionnalité qui fait quelque chose et peuvent être représentés par des symboles comme `+` ou des mots-clés spéciaux. Les opérateurs ont besoins de données pour agir et ces données sont appelées des *opérandes*. Dans ce cas, `2` et `3` sont les opérandes.

Opérateurs

Nous jetterons un coup d'oeil rapide aux opérateurs et à leur utilisation.

Notez que vous pouvez évaluer de manière interactive les expressions données dans les exemples en utilisant l'interpréteur. Par exemple, pour tester l'expression `2 + 3`, utilisez l'interpréteur interactif python:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Voici un aperçu rapide des opérateurs disponibles:

- `+` (plus)
 - Additionne les deux objets
 - `3 + 5` vaut `8`. `'a' + 'b'` vaut `'ab'`.
- `-` (moins)
 - Donne le résultat de la soustraction entre deux nombres; si le premier opérande est absent, on suppose qu'il vaut zéro.
 - `-5.2` vaut un nombre négatif et `50 - 24` vaut `26`.
- `*` (multiplication)
 - Donne le produit de deux nombres, ou bien une chaîne de caractères répétée le nombre de fois indiqué.
 - `2 * 3` vaut `6`. `'la' * 3` vaut `'lalala'`.
- `**` (puissance)
 - Renvoie `x` à la puissance `y`
 - `3 ** 4` vaut `81` (c'est-à-dire `3 * 3 * 3 * 3`)
- `/` (division)
 - Divise `x` par `y`
 - `13 / 3` vaut `4.333333333333333`
- `//` (quotient)
 - Divise `x` par `y` et arrondi le résultat à l'entier `inférieur`. Si l'une des opérandes est un flottant, le résultat sera également un flottant.
 - `13 // 3` vaut `4`
 - `-13 // 3` vaut `-5`
 - `9//1.81` vaut `4.0`
- `%` (modulo)
 - Renvoie le reste d'une division
 - `13 % 3` vaut `1`. `-25.5 % 2.25` vaut `1.5`.
- `<<` (décalage de bits à gauche)

- Fait un décalage à gauche du nombre de bits indiqué. (Chaque nombre est représenté en mémoire par des bits, c'est-à-dire 0 et 1)
 - `2 << 2` vaut `8`. `2` est représenté par `10` en binaire.
 - Le décalage à gauche de 2 bits donne `1000` qui représente le nombre `8` en décimale.
- `>>` (décalage de bits à droite)
 - Fait un décalage à droite du nombre de bits indiqué.
 - `11 >> 1` vaut `5`.
 - `11` est représenté par `1011` en binaire, qui décallé à droite d'un bit donne `101`, qui vaut `5` en décimale.
- `&` (et logique bit à bit)
 - `ET` bit à bit des nombres
 - `5 & 3` vaut `1`.
- `|` (ou logique bit à bit)
 - `ou` bit à bit des nombres
 - `5 | 3` vaut `7`
- `^` (ou exclusif bit à bit)
 - Calcule le ou exclusif bit à bit des nombres (`XOR`)
 - `5 ^ 3` vaut `6`
- `~` (inversion bit à bit)
 - L'inversion bit à bit de `x` vaut `-(x+1)`
 - `-5` vaut `-6`. Plus de détails à l'adresse <http://stackoverflow.com/a/11810203>
- `<` (moins que)
 - Indique si `x` est plus petit que `y`. Tous les opérateurs de comparaison renvoient `True` ou `False`. Notez la première lettre en majuscule.
 - `5 < 3` vaut `False` et `3 < 5` vaut `True`.
 - Les comparaisons peuvent être chaînées de manière arbitraire: `3 < 5 < 7` vaut `True`.
- `>` (plus grand que)
 - Indique si `x` est plus grand que `y`
 - `5 > 3` renvoie `True`. Si les deux opérandes sont des nombres, ils sont d'abord convertis en un type commun. Sinon, il renvoie toujours `False`.
- `<=` (inférieur ou égal à)
 - Indique si `x` est inférieur ou égal à `y`
 - `x = 3; y = 6; x <= y` vaut `True`
- `>=` (supérieur ou égal à)
 - Indique si `x` est supérieur ou égal à `y`
 - `x = 4; y = 3; x >= 3` vaut `True`
- `==` (égal à)
 - Compare deux objets pour voir s'ils sont égaux
 - `x = 2; y = 2; x == y` vaut `True`
 - `x = 'str'; y = 'stR'; x == y` vaut `False`
 - `x = 'str'; y = 'str'; x == y` vaut `True`
- `!=` (différent de)
 - Compare deux objets pour voir s'ils sont différents
 - `x = 2; y = 3; x != y` vaut `True`
- `not` (non)
 - Si `x` vaut `True`, il renvoie `False`. Si `x` est `False`, il renvoie `True`.
 - `x = True; not x` vaut `False`.
- `and` (et booléen)

- `x and y` vaut `False` si `x` est `False`, sinon il vaut l'évaluation de `y`
- `x = False; y = True; x and y` vaut `False` vu que `x` est `False`. Dans ce cas, Python ne va pas évaluer `y` car il sait que la partie gauche de l'expression '`and`' est `False` ce qui implique que l'expression complète sera `False` quelles que soient les autres valeurs. C'est ce que l'on appelle *short-circuit evaluation*.
- `or` (ou booléen)
 - Si `x` est `True`, il renvoie `True`, sinon il renvoie l'évaluation de `y`
 - `x = True; y = False; x or y` vaut `True`. La *short-circuit evaluation* s'applique ici aussi.

Raccourci pour les opérations mathématiques et les affectations

Il est fréquent de faire une opération mathématique avec une variable et ensuite affecter à nouveau le résultat de l'opération à la variable, et donc il y a un raccourci pour de telles expressions:

```
a = 2
a = a * 3
```

s'écrit également:

```
a = 2
a *= 3
```

Notez que `variable = variable operation expression` devient `variable operation= expression`.

Ordre d'évaluation

Si vous avez une expression comme `2 + 3 * 4`, fait-on d'abord l'addition ou la multiplication? Nos souvenirs de maths à l'école nous disent qu'il faut d'abord faire la multiplication. Cela signifie que l'opérateur de multiplication est prioritaire par rapport à l'opérateur d'addition.

Le tableau suivant donne les priorités pour Python, de la moins liante (ndlt: *least binding*) à la plus liante (ndlt: *most binding*). Cela signifie que dans une expression donnée, Python va d'abord évaluer les opérateurs et expressions les plus bas dans ce tableau, avant ceux placés plus en haut.

Le tableau suivant, extrait du [Manuel de référence de Python](#), est fourni afin d'être complet. Il est bien meilleur d'utiliser les parenthèses pour grouper correctement les opérateurs et les opérandes afin d'indiquer explicitement la précédence. Cela rend le programme plus lisible. Voyez [Changer l'ordre d'évaluation](#) pour plus de détails.

- `lambda` : Expression Lambda
- `if - else` : Expression conditionnelle
- `or` : Ou booléen
- `and` : Et booléen
- `not x` : Non booléen
- `in, not in, is, is not, <, <=, >, >=, !=, ==` : Comparaisons, en incluant les tests d'appartenance et les tests d'identité
- `|` : Ou bit à bit
- `^` : Ou exclusif bit à bit (XOR)
- `&` : Et bit à bit
- `<<, >>` : Décalage
- `+, -` : Addition et soustraction
- `*, /, //, %` : Multiplication, Division, Quotient et Reste
- `+x, -x, ~x` : Positif, Négatif, non bit à bit
- `**` : Exponentiation
- `x[index], x[index:index], x(arguments...), x.attribute` : Subscription, slicing, appel, référence aux attributs
- `(expressions...), [expressions...], {key: value...}, {expressions...}` : Création de tuples, listes, dictionnaires, sets.

Les opérateurs que nous n'avons pas encore rencontrés seront expliqués dans les chapitres suivants.

Les opérateurs avec la *même précédence* sont listés dans la même ligne dans le tableau ci-dessus. Par exemple, `+` et `-` ont la même précédence.

Changer l'ordre d'évaluation

Pour rendre les expressions plus lisibles, nous pouvons utiliser des parenthèses. Par exemple, `2 + (3 * 4)` est bien plus facile à comprendre que `2 + 3 * 4` qui demande de connaître la précédence des opérateurs. Comme pour toute chose, les parenthèses doivent être utilisées à bon escient (n'en abusez pas) et ne doivent pas être redondantes comme dans `(2 + (3 + 4))`.

Il y a un autre avantage à utiliser des parenthèses - cela nous aide à changer l'ordre d'évaluation. Par exemple, si vous évaluer une addition avant une multiplication dans une expression, alors vous pouvez écrire quelque chose comme `(2 + 3) * 4`.

Associativité

Les opérateurs sont en général associés de gauche à droite, c'est-à-dire que les opérateurs avec la même précédence sont évalués de la gauche vers la droite. Par exemple, `2 + 3 + 4` est évalué comme `(2 + 3) + 4`.

Expressions

Exemple (enregistrez en nommant le fichier `expression.py`):

```
length = 5
breadth = 2

area = length * breadth
print("L'aire vaut", area)
print("Le périmètre vaut", 2 * (length + breadth))
```

Résultat:

```
$ python expression.py
L'aire vaut 10
Le périmètre vaut 14
```

Comment ça marche

La longueur et la largeur du rectangle sont rangées dans des variables du même nom (`length` et `breadth`). Nous les utilisons pour calculer la surface et le périmètre du rectangle avec l'aide des expressions. Nous rangeons le résultat de l'expression `length * breadth` dans la variable `area` et ensuite nous l'affichons en utilisant la fonction `print`. Dans le deuxième cas, nous utilisons directement la valeur de l'expression `2 * (length + breadth)` dans la fonction `print`.

Notez également comment Python *affiche joliment* le résultat. Même si nous n'avons pas ajouté un espace entre "l'aire est de" et la variable `area`, Python l'insère pour nous afin d'avoir un joli affichage et le programme est bien plus lisible ainsi (vu que nous n'avons pas à nous inquiéter des espaces dans les chaînes de caractères utilisées à l'affichage). Voici un exemple de ce que fait Python pour rendre la vie du programmeur plus facile.

Récapitulatif

Nous avons vu comment utiliser les opérateurs, opérandes et expressions - ce sont les briques de base de n'importe quel programme. Ensuite, nous verrons comment les utiliser dans nos programmes avec des instructions.

Structures de contrôle

Dans les programmes que nous avons vus jusqu'à présent, il y a toujours eu une série d'instructions fidèlement exécutées par Python de haut en bas sans exceptions. Et si vous vouliez changer le flux de son fonctionnement? Par exemple, vous voulez que le programme prenne des décisions et fasse des choses différentes en fonction de situations différentes, comme afficher « Bonjour » ou « Bonsoir » en fonction de l'heure de la journée?

Comme vous l'avez peut-être deviné, ceci est réalisé à l'aide de structure de contrôle. Il existe trois instructions de structure de contrôle en Python - `if`, `for` et `while`.

L'instruction `if`

L'instruction `if` est utilisée pour vérifier une condition: si la condition est vraie, nous exécutons un bloc d'instructions (appelé le *bloc if*), sinon nous traitons un autre bloc d'instructions (appelé le *bloc else*). La clause *else* est facultative.

Exemple (enregistrer sous `if.py`):

```
number = 23
guess = int(input('Saisissez un entier: '))

if guess == number:
    # Le nouveau bloc commence ici
    print('Bravo, vous avez deviné.')
    print('(mais il n\'y a rien à gagner!)')
    # Le nouveau bloc fini ici
elif guess < number:
    # Un autre bloc
    print('Non, c\'est un peu plus que ça')
    # Vous pouvez faire ce que vous voulez dans un bloc...
else:
    print('Non, c\'est une peu moins que ça')
    # Il faut que guess > number pour être ici

print('Fini')
# Cette dernière instruction est toujours exécutée,
# après que l'instruction if soit exécuté.
```

Résultat:

```
$ python if.py
Saisissez un entier: 50
Non, c'est un peu moins que ça
Fini

$ python if.py
Saisissez un entier: 22
Non, c'est un peu plus que ça
Fini

$ python if.py
Saisissez un entier: 23
Bravo, vous avez deviné.
(mais il n'y a rien à gagner!)
Fini
```

Comment ça marche

Dans ce programme, nous prenons des conjectures de l'utilisateur et vérifions si c'est le nombre que nous avons. Nous définissons la variable `number` avec le nombre entier voulu, par exemple, `23`. Ensuite, nous prenons la conjecture de l'utilisateur en utilisant la fonction `input()`. Les fonctions ne sont que des morceaux de programmes réutilisables. Nous en apprendrons plus à leur sujet dans le

[chapitre suivant.](#)

Nous fournissons une chaîne à la fonction `input` intégrée, qui l'affiche à l'écran et attend l'entrée de l'utilisateur. Une fois que nous avons entré quelque chose et appuyé sur la touche [Entrée], la fonction `input()` renvoie ce que nous avons entré, sous forme de chaîne. Nous convertissons ensuite cette chaîne en un entier en utilisant `int`, puis nous l'enregistrons dans la variable `guess`. En fait, le `int` est une classe mais tout ce que vous devez savoir maintenant c'est que vous pouvez l'utiliser pour convertir une chaîne en entier (en supposant que la chaîne contienne un entier valide dans le texte).

Ensuite, nous comparons la conjecture de l'utilisateur avec le nombre que nous avons choisi. S'ils sont égaux, nous imprimons un message de réussite. Notez que nous utilisons des niveaux d'indentation pour indiquer à Python quelles instructions appartiennent à quel bloc. C'est pourquoi l'indentation est si importante en Python. J'espère que vous vous en tenez à la règle de « l'indentation systématique ». Le faites-vous?

Notez que l'instruction `if` contient deux points à la fin: nous indiquons à Python qu'un bloc d'instructions suit.

Ensuite, nous vérifions si l'estimation est inférieure au nombre et, dans l'affirmative, nous informons l'utilisateur qu'il doit deviner un peu plus haut que cela. Ce que nous avons utilisé ici est la clause `elif` qui combine en fait deux instructions `if` `else-if` `else` liées en une seule instruction `if-elif-else` combinée. Cela facilite le programme et réduit la quantité d'indentation requise.

Les instructions `elif` et `else` doivent également comporter un signe deux-points à la fin de la ligne logique, suivies du bloc d'instructions correspondant (avec l'indentation appropriée, bien sûr)

Vous pouvez avoir une autre instruction `if` dans le bloc if d'une instruction `if` et ainsi de suite. Cette instruction est appelée une instruction `if` imbriquée.

Rappelez-vous que les parties `elif` et `else` sont optionnelles. Une instruction `if` valide minimale est:

```
if True:
    print('Oui, c\'est vrai')
```

Une fois que Python a fini d'exécuter l'instruction `if` complète ainsi que les clauses `elif` et `else` associées, il passe à l'instruction suivante du bloc contenant l'instruction `if`. Dans ce cas, il s'agit du bloc principal (où commence l'exécution du programme) et l'instruction suivante est l'instruction `print('Fini')`. Après cela, Python voit la fin du programme et finit simplement.

Même s'il s'agit d'un programme très simple, j'ai souligné beaucoup de choses que vous devriez remarquer. Tout cela est assez simple (et étonnamment simple pour ceux d'entre vous qui programment en C/C++). Vous aurez besoin d'appliquer tous ces éléments au début, mais après quelques exercices, vous vous sentirez à l'aise avec elles et elles vous paraîtront « naturelles ».

Remarque pour les programmeurs C/C++

Il n'y a pas d'instruction `switch` en Python. Vous pouvez utiliser une instruction `if..elif..else` pour faire la même chose (et dans certains cas, utiliser un [dictionnaire](#) pour le faire rapidement)

L'instruction while

L'instruction `while` vous permet d'exécuter plusieurs fois un bloc d'instructions tant qu'une condition est vraie. Une instruction `while` est un exemple de ce que l'on appelle une instruction de *boucle*. Une instruction `while` peut avoir une clause optionnelle `else`.

Exemple (enregistrer sous `while.py`):

```

number = 23
running = True

while running:
    guess = int(input('Saisissez un entier: '))

    if guess == number:
        print('Bravo, vous avez deviné.')
        # Ceci va arrêter la boucle while
        running = False
    elif guess < number:
        print('Non, c\'est un peu plus que ça.')
    else:
        print('Non, c\'est un peu moins que ça.')
else:
    print('La boucle while est finie.')
    # Faites tout ce que vous voulez faire d'autre ici

print('Fini')

```

Résultat:

```

$ python while.py
Saisissez un entier: 50
Non, c'est un peu moins que ça.
Saisissez un entier: 22
Non, c'est un peu plus que ça.
Saisissez un entier: 23
Bravo, vous avez deviné.
La boucle while est finie.
Fini

```

Comment ça marche

Dans ce programme, nous continuons à jouer aux devinettes, mais l'avantage est que l'utilisateur est autorisé à continuer à essayer jusqu'à ce qu'il devine correctement: il n'est pas nécessaire de relancer le programme à chaque fois, comme nous l'avons fait dans la section précédente. Cela montre bien l'utilisation de l'instruction `while`.

Nous déplaçons les instructions `input` et `if` dans la boucle `while` et définissons la variable `running` à `True` avant la boucle `while`. Premièrement, nous vérifions si la variable `running` est `True`, puis exécutons le *bloc while* correspondant. Une fois ce bloc exécuté, la condition est à nouveau vérifiée, qui dans ce cas est la variable `running`. Si elle vaut vrai, nous exécutons à nouveau le *bloc while*, sinon, nous continuons d'exécuter le bloc optionnel `else`, puis nous passons à l'instruction suivante.

Le bloc `else` est exécuté lorsque la condition de boucle `while` devient `False`: il peut même s'agir de la première fois que la condition est vérifiée. S'il existe une clause `else` pour une boucle `while`, elle est toujours exécutée à moins que vous ne sortiez de la boucle avec une instruction `break`.

Les types `True` et `False` sont appelés types booléens et vous pouvez les considérer comme équivalents aux valeurs `1` et `0` respectivement.

Remarque pour les programmeurs C/C++

N'oubliez pas la clause `else` des boucles `while`.

La boucle `for`

L'instruction `for...in` est une autre instruction pour réaliser des boucles qui *itère* sur une séquence d'objets, c'est-à-dire passe en revue chaque élément d'une séquence. Nous verrons les *séquences* en détail dans les chapitres suivants. Ce que vous devez savoir maintenant, c'est qu'une séquence est simplement une collection ordonnée d'objets.

Exemple (enregistrer sous `for.py`):

```

for i in range(1, 5):
    print(i)
else:
    print('La boucle for est finie')

```

Résultat:

```

$ python for.py
1
2
3
4
La boucle for est finie

```

Comment ça marche

Dans ce programme, nous affichons une *séquence* de nombres. Nous générions cette séquence de nombres en utilisant la fonction intégrée `range`.

Nous fournissons ici deux nombres et `range` renvoie une séquence de nombres en commençant par le premier nombre indiqué et jusqu'au deuxième nombre indiqué. Par exemple, `range(1,5)` donne la séquence `[1, 2, 3, 4]`. Par défaut, `range` prend un pas de 1. Si nous fournissons un troisième nombre à `range`, alors cela devient le compteur de pas. Par exemple, `range(1,5,2)` donnera `[1, 3]`. Souvenez-vous que l'intervalle va *jusqu'au* second nombre, mais ne l'inclut *pas*.

Notez que `range()` génère une séquence de nombres, mais il va générer seulement un nombre à la fois, quand la boucle `for` demande l'item suivant. Si vous voulez voir la séquence complète des nombres immédiatement, utilisez `list(range())`. les listes sont expliquées dans le [chapitre des structures de données](#).

La boucle `for` itère dans cet intervalle - `for i in range(1,5)` est équivalent à `for i in [1, 2, 3, 4]` ce qui est comme affecter chaque nombre (ou objet) dans la séquence à `i`, un à la fois, et ensuite exécuter le bloc d'instructions pour chaque valeur de `i`. Dans ce cas, nous affichons juste la valeur dans le bloc d'instructions.

Souvenez-vous que la partie `else` est optionnelle. Quand elle existe, elle est toujours exécutée une fois après la fin de la boucle `for` à moins qu'une instruction `break` ne soit présente.

Souvenez-vous que la boucle `for..in` fonctionne avec n'importe quelle séquence. Ici, nous avons une liste de nombres générée par la fonction intégrée `range`, mais en général nous pouvons utiliser n'importe quel type de séquence sur n'importe quel type d'objet! Nous explorerons cette idée en détail dans les prochains chapitres.

Note pour les programmeurs C/C++/Java/C#

La boucle `for` en Python est complètement différente de la boucle `for` en C/C++. les programmeurs C# noteront que la boucle `for` en Python est similaire à la boucle `foreach` en C#. Les programmeurs Java noteront que cela est similaire à `for (int i : IntArray)` en Java 1.5 .

En C/C++, quand vous écrivez `for (int i = 0; i < 5; i++)`, alors en Python vous écrivez juste `for i in range(0,5)`. Comme vous le voyez, la boucle `for` est plus simple, plus expressive et moins sujette à l'erreur en Python.

L'instruction `break`

L'instruction `break` est utilisée pour *sortir* d'une instruction de boucle, par exemple arrêter l'exécution d'une instruction qui boucle, même si la condition de la boucle n'est pas devenue `False` ou si la séquence d'items n'est pas complètement consommée.

Une chose importante à noter : dans le cas d'un `break` en dehors d'une boucle `for` ou `while`, n'importe quel bloc `else` associé n'est **pas** exécuté.

Exemple (enregistrer sous `break.py`):

```

while True:
    s = input('Entrez quelque chose :')
    if s == 'quit':
        break
    print ('La longueur de la chaîne est', len(s))
print('Terminé')

```

Résultat:

```

$ python break.py
Entrez quelque chose : La programmation est drôle
La longueur de la chaîne est 26
Entrez quelque chose : Quand le travail est fait
La longueur de la chaîne est 25
Entrez quelque chose : si vous voulez rendre votre travail drôle:
La longueur de la chaîne est 42
Entrez quelque chose : utiliser Python sera parfait!
La longueur de la chaîne est 33
Entrez quelque chose : quit
Terminé

```

Comment ça marche

Dans ce programme, nous récupérons la saisie de l'utilisateur de manière répétitive et affichons la longueur saisie à chaque fois. Nous fournissons une condition spéciale pour arrêter le programme en testant si l'utilisateur a saisi `quit`. Nous arrêtons le programme en *cassant* la boucle (`break`) et en atteignant la fin du programme.

La longueur de la chaîne de caractères saisie peut être trouvée en utilisant la fonction intégrée `len`.

Souvenez-vous que l'instruction `break` peut également être utilisée avec une boucle `for`.

Le Python Poétique de Swaroop

L'entrée que j'ai utilisée ici est un mini poème que j'ai écrit:

```

La programmation est drôle
Quand le travail est fait
si vous voulez rendre votre travail drôle:
    utiliser Python sera parfait!

```

L'instruction `continue`

L'instruction `continue` est utilisée pour indiquer à Python de sauter le reste des instructions dans la boucle courante du bloc et de *continuer* jusqu'à la prochaine itération de la boucle.

Exemple (enregistrer sous `continue.py`):

```

while True:
    s = input('Entrez quelque chose: ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('Trop petit')
        continue
    print('Longueur suffisante')
    # Faites d'autres choses ici...

```

Résultat:

```
$ python test.py
Entrez quelque chose : a
Trop petit
Entrez quelque chose : 12
Trop petit
Entrez quelque chose : abc
Longueur suffisante
Entrez quelque chose : quit
```

Comment ça marche

Dans ce programme, l'utilisateur saisit une valeur, mais nous l'acceptons seulement si sa longueur est au moins de 3 caractères. Donc, nous utilisons la fonction intégrée `len` pour récupérer la longueur et si elle est inférieure à 3, nous sautons le reste des instructions du bloc en utilisant l'instruction `continue`. Sinon, le reste des instructions de la boucle est exécuté et nous pouvons faire tous les traitements que nous voulons ici.

Notez que l'instruction `continue` fonctionne aussi avec `for`.

Récapitulatif

Nous avons vu comment utiliser les trois structures de contrôle - `if`, `while` et `for` avec les instructions associées `break` et `continue`. Ces instructions sont parmi les plus utilisées dans Python et donc, être à l'aise avec ces instructions est fondamental.

Ensuite, nous verrons comment créer et utiliser des fonctions.

Fonctions

Les fonctions sont des morceaux re-utilisables de programmes. Ils vous permettent de donner un nom à un bloc d'instructions et vous pouvez exécuter ce bloc n'importe où et autant de fois que vous le voulez. C'est ce qu'on nomme *appeler* la fonction. Nous avons déjà utilisé des fonctions intégrées comme `len` et `range`.

Le concept de fonction est probablement *le* plus important bloc de base de n'importe quel logiciel un peu complexe (dans n'importe quel langage), donc nous explorerons divers aspects des fonctions dans ce chapitre.

Les fonctions sont définies en utilisant le mot-clé `def`. Cela est suivi par un *identifiant* pour la fonction suivi d'une paire de parenthèses qui peuvent inclure des noms de variables et par un caractère deux-points qui termine la ligne. Puis suit un bloc d'instructions qui font partie de la fonction. Un exemple va montrer que cela est en fait très simple:

Exemple (enregistrez sous `function1.py`):

```
def say_hello():
    # bloc appartenant à la fonction
    print('Hello World')
# Fin de la fonction

say_hello() # appel de la fonction
say_hello() # nouvel appel de la fonction
```

Résultat:

```
$ python function1.py
Hello World!
Hello World!
```

Comment ça marche

Nous avons défini une fonction appelée `say_hello` qui utilise la syntaxe expliquée ci-dessus. La fonction ne prend pas de paramètre et donc il n'y a pas de variables déclarées entre les parenthèses. Les paramètres d'une fonction sont juste des données en entrée de la fonction afin de passer des valeurs et de récupérer les résultats correspondants.

Notez que nous pouvons appeler la même fonction deux fois ce qui implique que nous n'avons pas à écrire le même code à nouveau.

Paramètres de fonction

Une fonction peut prendre des paramètres, qui sont des valeurs fournies à la fonction afin que la fonction *fasse* quelque chose en utilisant ces valeurs. Ces paramètres sont comme des variables, sauf que les valeurs de ces variables sont définies quand nous appelons la fonction et ont déjà des valeurs affectées quand la fonction est exécutée.

Les paramètres sont spécifiés à l'intérieur de la paire de parenthèses de la définition de la fonction, séparées par des virgules. Quand nous appelons la fonction, nous fournissons les valeurs de la même manière. Notez la terminologie utilisée - les noms donnés dans la définition de la fonction sont appelés des *paramètres* quand les valeurs que vous fournissez à l'appel de la fonction sont des *arguments*.

Exemple (enregistrez sous `function_param.py`):

```

def print_max(a, b):
    if a > b:
        print(a, 'est le plus grand')
    elif a == b:
        print(a, 'est égal à', b)
    else:
        print(b, 'est le plus grand')

# fournit des constantes littérales en tant qu'arguments
print_max(3, 4)

x = 5
y = 7

# fournit des variables en tant qu'arguments
print_max(x, y)

```

Résultat:

```
$ python func_param.py
4 est le plus grand
7 est le plus grand
```

Comment ça marche

Ici, nous définissons une fonction appelée `print_max` qui prend deux paramètres appelés `a` et `b`. Nous trouvons le nombre le plus grand en utilisant une simple instruction `if..else` et nous l'affichons.

Dans la première utilisation de `print_max`, nous fournissons directement les nombres, c'est-à-dire les arguments. Dans la deuxième utilisation, nous appelons la fonction en utilisant des variables. `print_max(x, y)` fait que la valeur de l'argument `x` est affectée au paramètre `a` et la valeur de l'argument `y` affectée au paramètre `b`. La fonction `print_max` agit de la même manière dans les deux cas.

Variables locales

Quand vous déclarez des variables à l'intérieur de la définition d'une fonction, elles ne sont en aucun cas en rapport avec d'autres variables portant le même nom, utilisées en dehors de la fonction, c'est-à-dire que les noms de variables sont *locaux* à la fonction. Cela est appelé la *portée* de la variable. Toutes les variables ont la portée du bloc dans lequel elles sont déclarées, à partir du point de définition du nom.

Note de la traduction

Le terme anglais *scope* est très utilisé, et à le même sens que *portée*.

Exemple (enregistrez sous `function_local.py`):

```

x = 50

def func(x):
    print('x vaut', x)
    x = 2
    print('Nous avons changé le x local à', x)

func(x)
print('x vaut toujours', x)

```

Résultat:

```
$ python func_local.py
x est 50
Nous avons changé le x local à 2
x est toujours 50
```

Comment ça marche

La première fois que nous imprimons la *valeur* du nom `x` avec la première ligne du corps de la fonction, Python utilise la valeur du paramètre déclaré dans le bloc principal, au-dessus de la définition de la fonction.

Ensuite, nous affectons la valeur `2` à `x`. Le nom `x` est local dans notre fonction. Ainsi, lorsque nous changeons la valeur de `x` dans la fonction, le `x` défini dans le bloc principal reste inchangé.

Avec la dernière instruction `print`, nous affichons la valeur de `x` telle que définie dans le bloc principal, confirmant ainsi qu'elle n'est en réalité pas affectée par l'affectation locale dans la fonction précédemment appelée.

L'instruction `global`

Si vous voulez affecter une valeur à un nom défini au niveau supérieur de votre programme (c'est-à-dire en dehors de la portée de n'importe quelle fonction ou classe), alors vous devez indiquer à Python que ce nom n'est pas local, mais qu'il est *global*. Vous faites cela avec l'instruction `global`. Il est impossible d'affecter une valeur à une variable définie en dehors d'une fonction sans l'instruction `global`.

Vous pouvez utiliser les valeurs de telles variables définies en dehors d'une fonction (en supposant qu'il n'existe pas de variable avec le même nom à l'intérieur de la fonction). Cependant, cela est déconseillé et devrait être évité car le programme devient confus et le lecteur ne sait plus où est la définition de la variable. Utiliser l'instruction `global` indique clairement que la variable est définie dans un bloc éloigné.

Exemple (enregistrez sous `function_global.py`):

```
x = 50

def func():
    global x

    print('x vaut', x)
    x = 2
    print('Nous avons changé la valeur de la variable globale x à', x)

func()
print('x vaut', x)
```

Résultat:

```
$ python func_global.py
x vaut 50
Nous avons changé la valeur de la variable globale x à 2
x vaut 2
```

Comment ça marche

L'instruction `global` est utilisée pour déclarer que `x` est une variable globale - d'où, quand nous affectons une valeur à `x` à l'intérieur de la fonction, ce changement est mis en évidence quand nous utilisons la valeur de `x` dans le bloc principal.

Vous pouvez déclarer plusieurs variables globales en utilisant la même instruction `global`. Par exemple, `global x, y, z`.

Valeurs d'arguments par défaut

Pour certaines fonctions, vous voudrez que certains des arguments soient *optionnels* et utilisent des valeurs par défaut si l'utilisateur ne précise pas leur valeur. Cela est fait avec des valeurs d'arguments par défaut. Vous pouvez spécifier ces valeurs par défaut en ajoutant au nom du paramètre dans la définition de la fonction l'opérateur d'affectation (=) suivi de la valeur par défaut.

Notez que la valeur d'argument par défaut doit être une constante. Plus précisément, la valeur d'argument par défaut doit être immuable - cela est expliqué en détail dans des chapitres ultérieurs. Pour l'instant, retenez juste ce qui précède.

Exemple (enregistrez sous `function_default.py`):

```
def say(message, times = 1):
    print(message * times)

say('Hello')
say('World', 5)
```

Résultat:

```
$ python func_default.py
Hello
WorldWorldWorldWorldWorld
```

Comment ça marche

Le nom de la fonction `say` est utilisé pour afficher une chaîne de caractères autant de fois qu'indiqué. Si nous ne fournissons pas de valeur pour ce nombre de fois, alors par défaut la chaîne de caractères est affichée juste une fois. Nous obtenons ceci en indiquant une valeur par défaut de `1` pour le paramètre `times`.

Au premier appel de `say`, nous fournissons seulement la chaîne de caractères et elle est imprimée une fois. Au deuxième appel de `say`, nous fournissons à la fois la chaîne de caractères et un argument `5` indiquant que nous voulons dire le message de la chaîne de caractères 5 fois.

Important

Seuls les paramètres à la fin de la liste de paramètres peuvent recevoir une valeur par défaut, c'est-à-dire que vous ne pouvez avoir un paramètre avec une valeur par défaut avant un paramètre sans valeur par défaut dans la liste des paramètres de la fonction.

La raison est que les valeurs sont affectées aux paramètres par position. Par exemple, `def func(a, b=5)` est valide, mais `def func(a=5, b)` n'est *pas valide*.

Paramètres nommés

Si vous avez des fonctions avec de nombreux paramètres et que vous en spécifiez seulement certains, vous pouvez donner des valeurs à ces paramètres en les nommant - cela est appelé *paramètres nommés* - nous utilisons le nom (mot-clé) au lieu de la position (que nous avons utilisée jusque-là) pour spécifier les arguments de la fonction.

Il y a deux avantages - un, utiliser la fonction est plus facile car nous n'avons pas à nous soucier de l'ordre des paramètres. Deux, nous pouvons donner des valeurs seulement aux paramètres que nous voulons, en supposant que les autres paramètres ont des valeurs par défaut.

Exemple (enregistrez sous `function_keyword.py`):

```
def func(a, b=5, c=10):
    print('a vaut', a, 'et b vaut', b, 'et c vaut', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Résultat:

```
$ python func_key.py
a vaut 3 et b vaut 7 et c vaut 10
a vaut 25 et b vaut 5 et c vaut 24
a vaut 100 et b vaut 5 et c vaut 50
```

Comment ça marche

La fonction `func` a un paramètre par défaut sans valeur, suivi de deux paramètres avec des valeurs par défaut.

Dans le premier cas, `func(3, 7)`, le paramètre `a` prend la valeur `3`, le paramètre `b` la valeur `7` et `c` la valeur par défaut `10`.

Dans le deuxième cas, `func(25, c=24)`, la variable `a` prend la valeur `25` à cause de la position du paramètre. Ensuite, le paramètre `c` prend la valeur `24` à cause du nom des paramètres. La variable `b` prend la valeur par défaut de `5`.

Dans le troisième cas, `func(c=50, a=100)`, nous utilisons le mot-clé argument pour spécifier les valeurs. Notez que nous spécifions la valeur du paramètre `c` avant celle de `a` même si `a` est défini avant `c` dans la définition de la fonction.

Nombre d'arguments arbitraire

Vous voudrez parfois définir une fonction qui peut prendre *n'importe quel* nombre de paramètres, cela peut être obtenu en utilisant les étoiles (enregistrez sous `function_varargs.py`):

```
def total(a=5, *numbers, **phonebook):
    print('a', a)

    # parcourt tous les éléments du tuple
    for single_item in numbers:
        print('single_item', single_item)

    # parcourt tous les éléments du dictionnaire
    for first_part, second_part in phonebook.items():
        print(first_part, second_part)

total(10, 1, 2, 3, Jack=1123, John=2231, Inge=1560)
```

Résultat:

```
$ python function_varargs.py
a 10
single_item 1
single_item 2
single_item 3
Inge 1560
John 2231
Jack 1123
```

Comment ça marche

Quand nous déclarons un paramètre arbitraire comme `*param`, alors tous les paramètres à partir de cette position jusqu'à la fin sont regroupés dans un tuple appelé 'param'.

De la même manière, quand nous déclarons un paramètre non-explicite comme `**param`, alors tous les mots-clés jusqu'à la fin sont regroupés dans un dictionnaire appelé 'param'.

Nous explorerons les tuples et les dictionnaires dans un [chapitre suivant](#).

L'instruction `return`

L'instruction `return` est utilisée pour *revenir* d'une fonction, c'est à dire sortir de la fonction. Vous pouvez optionnellement *retourner une valeur* de la fonction.

Exemple (enregistrez sous `function_return.py`):

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        print('Les nombres sont égaux')
    else:
        return y

print(maximum(2, 3))
```

Résultat:

```
$ python func_return.py
3
```

Comment ça marche

La fonction `maximum` renvoie la valeur maximum des paramètres, dans ce cas les nombres fournis à la fonction. Elle utilise une simple instruction `if..else` pour trouver la plus grande valeur et ensuite *retourne* cette valeur.

Notez qu'une instruction `return` sans une valeur est équivalente à `return None`. `None` est un type spécial en Python, qui représente le néant. Par exemple, il est utilisé pour indiquer qu'une valeur n'a pas de valeur, si elle a une valeur de `None`.

Chaque fonction contient implicitement une instruction `return None` à la fin, à moins que vous ayez écrit votre propre instruction `return`. Vous pouvez voir cela en lançant `print(someFunction())` où la fonction `someFunction` n'utilise pas l'instruction `return` comme:

```
def some_function():
    pass
```

L'instruction `pass` est utilisée en Python pour indiquer un bloc d'instructions vide.

CONSEIL: Il y a une fonction intégrée appelée `max` qui implémente déjà la fonctionnalité 'trouver le maximum', donc utilisez-la de préférence autant que possible.

DocStrings

Python a une chic fonctionnalité appelée *documentation strings*, communément appelée *docstrings*. Les DocStrings sont un outil important que vous devriez utiliser, car cela vous aide à mieux documenter le programme et le rend plus facile à comprendre. Etonnamment, nous pouvons même récupérer les docstrings en revenant, disons d'une fonction, pendant que le programme s'exécute!

Exemple (enregistrez sous `function_docstring.py`):

```

def print_max(x, y):
    '''Affiche le plus grand de deux nombres

    Les deux valeurs doivent être des entiers.'''
    x = int(x) # conversion vers un entier, si possible
    y = int(y)

    if x > y:
        print(x, 'est le plus grand')
    else:
        print(y, 'est le plus grand')

print_max(3, 5)
print(print_max.__doc__)

```

Résultat:

```

$ python func_doc.py
5 est le plus grand
Affiche le plus grand de deux nombres

Les deux valeurs doivent être des entiers.

```

Comment ça marche

Une chaîne de caractères sur la première ligne logique de la fonction est la *docstring* pour cette fonction. Notez que les docstrings s'appliquent aussi aux [modules](#) et aux [classes](#) que nous verrons dans les chapitres suivants.

La convention pour une docstring est une chaîne de caractères sur plusieurs lignes, la première ligne commençant avec une majuscule et se terminant par un point. La deuxième ligne est vide suivie par une explication détaillée commençant sur la troisième ligne. Vous êtes *fortement invité* à suivre cette convention pour toutes vos docstrings pour toutes vos fonctions non-triviales.

Nous pouvons accéder la docstring de la fonction `print_max` en utilisant les attributs (nom appartenant à) `__doc__` (notez les *double underscores*) de la fonction. Souvenez-vous juste que Python traite *tout* en tant qu'objet et cela inclut les fonctions. Nous en apprendrons plus sur les objets dans un prochain chapitre sur [classes](#).

Si vous avez utilisé `help()` en Python, alors vous avez déjà utilisé les docstrings! Les docstrings vont juste chercher l'attribut `__doc__` de la fonction et vous l'affichent d'une manière soignée. Vous pouvez essayer sur la fonction au-dessus - incluez juste `help(print_max)` dans votre programme. Souvenez-vous d'appuyer sur `q` pour sortir du `help`.

Des outils automatiques peuvent récupérer la documentation de votre programme de cette manière. Par conséquent, je *recommande fortement* que vous utilisez les docstrings pour tout fonction non-triviale que vous écrivez. La commande `pydoc` fournie avec Python fonctionne de manière similaire au `help()` en utilisant les docstrings.

Récapitulatif

Nous avons couvert de nombreux aspects des fonctions mais il nous reste des choses à voir. Cependant nous avons vu l'essentiel de ce que vous utiliserez couramment concernant les fonctions dans Python.

Nous allons maintenant voir comment créer des modules Python.

Modules

Vous avez vu comment vous pouvez re-utiliser du code dans votre programme en définissant les fonctions une fois. Et si vous vouliez re-utiliser un certain nombre de fonctions dans d'autres programmes que vous écrivez ? Comme vous l'avez deviné, la réponse est d'utiliser les modules.

Il y a plusieurs méthodes pour écrire des modules, mais la manière la plus simple est de créer un fichier d'extension `.py` qui contient les fonctions et variables.

Une autre méthode est d'écrire les modules dans le langage natif de l'interpréteur Python. Par exemple, vous pouvez écrire des modules dans le [Langage C](#) et une fois compilés, ils peuvent être utilisés à partir de votre code Python quand vous utilisez l'interpréteur Python.

Un module peut être *importé* par un autre programme pour utiliser ses fonctionnalités. De la même manière, nous pouvons utiliser la bibliothèque Python standard. Nous allons d'abord voir comment utiliser les modules de la bibliothèque standard.

Exemple (enregistrez sous `module_using_sys.py`):

```
import sys

print('Les arguments en ligne de commande sont:')
for i in sys.argv:
    print(i)

print('\n\nThe PYTHONPATH is', sys.path, '\n')
```

Résultat:

```
$ python using_sys.py nous sommes des arguments
Les arguments en ligne de commande sont :
using_sys.py
nous
sommes
des
arguments

Le PYTHONPATH est ['/tmp/py',
# ...
'/Library/Python/3.6/site-packages',
'/usr/local/lib/python3.6/site-packages']
```

Comment ça marche

D'abord, nous *importons* le module `sys` avec l'instruction `import`. Fondamentalement, nous disons à Python que nous voulons utiliser un module. Le module `sys` contient des fonctionnalités relatives à l'interpréteur Python et son environnement, c'est-à-dire le système.

Quand Python exécute l'instruction `import sys`, il va chercher le module `sys`. Dans ce cas, c'est un module intégré, et donc Python sait où le trouver.

Si ce n'était pas un module compilé, c'est-à-dire un module écrit en Python, alors l'interpréteur Python le chercherait dans les répertoires de la variable `sys.path`. Si le module est trouvé, alors les instructions dans le corps de ce module sont exécutées et le module est rendu *disponible* pour vous à l'utilisation. Notez que l'initialisation est exécutée seulement la *première* fois qu'un module est importé.

La variable `argv` du module `sys` est accédée en utilisant la dotted notation, c'est-à-dire `sys.argv`. Cela indique clairement que ce nom fait partie du module `sys`. Un autre avantage de cette approche est que le nom n'entre pas en conflit avec n'importe quelle variable `argv` utilisée dans votre programme.

La variable `sys.argv` est une *liste* de chaîne de caractères (les listes sont expliquées en détail dans un [chapitre ultérieur](#)). Plus spécifiquement, `sys.argv` contient la liste des *command line arguments* c'est-à-dire les arguments passés à votre programme en utilisant la ligne de commande.

Si vous utilisez un EDI pour écrire et exécuter ces programmes, cherchez une manière de passer des command line arguments au programme dans les menus.

Ici, quand nous exécutons `python using_sys.py nous sommes des arguments`, nous lançons le module `using_sys.py` avec la commande `python` et les autres choses qui suivent sont des arguments passés au programme. Python stocke les arguments de la ligne de commande dans la variable `sys.argv` pour nous afin de l'utiliser.

Souvenez-vous, le nom du script qui s'exécute est toujours le premier argument dans la liste `sys.argv`. Donc, dans ce cas nous aurons '`using_sys.py`' en tant que `sys.argv[0]`, 'nous' en tant que `sys.argv[1]`, 'sommes' en tant que `sys.argv[2]`, 'des' en tant que `sys.argv[3]` et 'arguments' en tant que `sys.argv[4]`. Notez que Python commence à compter à partir de 0 et pas 1.

Le `sys.path` contient la liste des noms de répertoires d'où les modules sont importés. Notez que la première chaîne de caractères dans `sys.path` est vide - cette chaîne vide indique que le répertoire courant fait partie de `sys.path` qui est comme la variable d'environnement `PYTHONPATH`. Cela signifie que vous pouvez directement importer les modules situés dans le répertoire courant. Sinon, vous devez placer votre module dans un des répertoires listés dans `sys.path`.

Notez que le répertoire courant est le répertoire à partir duquel le programme est lancé. Lancez `import os; print(os.getcwd())` pour trouver le répertoire courant de votre programme.

Fichiers .pyc byte-compilés

Importer un module est assez couteux, donc Python fait des ruses pour être plus rapide. Une façon est de créer des fichiers "byte-compiled" avec l'extension `.pyc` qui est une forme intermédiaire dans laquelle Python transforme le programme (souvenez-vous du chapitre [introduction](#) sur la manière de fonctionner de Python ?). Ce fichier `.pyc` est utile quand vous importez le module une autre fois à partir d'un autre programme - cela sera beaucoup plus rapide vu qu'une partie du traitement nécessaire à l'importation d'un module est déjà fait. Egalelement, ces fichiers byte-compiled sont indépendants de la plateforme.

NOTE: Ces fichiers `.pyc` sont en général créés dans le même répertoire que les fichiers correspondants `.py`. Si Python n'a pas l'autorisation d'écrire dans ce répertoire, alors les fichiers `.pyc` ne seront pas créés.

L'instruction `from..import`

Si vous voulez importer directement la variable `argv` dans votre programme (pour éviter de taper `sys.` à chaque fois), vous pouvez utiliser l'instruction `from sys import argv`. Si vous voulez importer tous les noms utilisés dans le module `sys`, alors vous pouvez utiliser l'instruction `from sys import *`. Cela fonctionne pour n'importe quel module.

En général, évitez d'utiliser cette instruction et utilisez à sa place l'instruction `import` ainsi votre programme évitera les conflits de noms et sera plus lisible.

Exemple:

```
from math import sqrt
print("La racine carré de 16 vaut", sqrt(16))
```

Nommage des modules

Chaque module a un nom et les instructions dans un module peuvent retrouver le nom du module. Cela est pratique pour déterminer si le module s'exécute tout seul ou s'il est importé. Comme précédemment indiqué, quand un module est importé pour la première fois, le code dans ce module est exécuté. Nous pouvons utiliser ce concept pour modifier le comportement du module selon que le programme s'exécute tout seul ou qu'il est importé à partir d'un autre module. Cela est obtenu avec l'attribut `__name__` du module.

Exemple (enregistrez sous `module_using_name.py`):

```

if __name__ == '__main__':
    print('Ce programme est lancé par lui-même')
else:
    print('Je suis importé à partir d\'un autre module')

```

Résultat:

```

$ python using_name.py
Ce programme est lancé par lui-même

$ python
>>> import using_name
Je suis importé à partir d'un autre module
>>>

```

Comment ça marche

Chaque module Python a son `__name__` défini et si c'est son `'__main__'`, cela implique que le module s'exécute standalone par l'utilisateur et nous pouvons agir en conséquence.

Créer vos propres modules

Créer vos propres modules est facile, vous l'avez déjà fait ! C'est parce que chaque programme Python est aussi un module. Vérifiez juste que son extension est `.py`. L'exemple suivant devrait clarifier cela.

Exemple (enregistrez sous `mymodule.py`):

```

def sayhi():
    print('Bonjour, voici mon module qui s\'exprime.')

__version__ = '0.1'

```

Ce qui précède était un exemple de *module*. Comme vous pouvez le constater, cela n'a rien de particulièrement spécial par rapport à notre programme Python habituel. Nous verrons ensuite comment utiliser ce module dans nos autres programmes Python.

N'oubliez pas que le module doit être placé dans le même répertoire que le programme à partir duquel nous l'importons, ou dans l'un des répertoires répertoriés dans `sys.path`.

Un autre module (enregistrer sous `mymodule_demo.py`):

```

import mymodule

mymodule.say_hi()
print('Version', mymodule.__version__)

```

Résultat:

```

$ python mymodule_demo.py
Bonjour, voici mon module qui s'exprime.
Version 0.1

```

Comment ça marche

Notez que nous utilisons la notation pointée pour accéder les membres du module. Python re-utilise cette notation pour donner un aspect 'Pythonique' pour nous simplifier la vie.

Voici une version utilisant la syntaxe `from..import :` (enregistrez sous `mymodule_demo2.py`):

```
from mymodule import say_hi, __version__
say_hi()
print('Version', __version__)
```

L'affichage de `monmodule_demo2.py` est le même que celui de `mymodule_demo.py`.

Notez que s'il y avait déjà un nom `__version__` déclaré dans le module qui importe `mymodule`, il y aurait une collision entre les deux. Cela est aussi vraisemblable parce que l'usage est que chaque module doit déclarer son numéro de version en utilisant ce nom. C'est pour cette raison qu'il est toujours recommandé de préférer l'instruction `import`, même si elle peut rendre votre programme un peu plus long.

Vous pourriez aussi utiliser :

```
from mymodule import *
```

Cela va importer tous les noms publics comme `sayhi` mais ne va pas importer `__version__` parce qu'il commence par des double underscores.

WARNING: N'oubliez pas que vous devriez éviter d'utiliser cette fonctionnalité `from mymodule import *`.

Zen de Python

L'un des principes directeurs de Python est que « Explicite vaut mieux qu'implicite ». Exécutez `import this` en Python pour en savoir plus.

La fonction `dir`

Vous pouvez utiliser la fonction intégrée `dir()` pour lister les noms définis par un objet. Si l'objet est un module, cette liste incluera les fonctions, classes et variables définies dans le module.

Cette fonction peut accepter des arguments. Si l'argument est le nom d'un module, la fonction renvoie la liste des noms de ce module spécifié. S'il n'y a pas d'argument, la fonction renvoie la liste des noms du module courant.

Exemple:

```
$ python
>>> import sys

# obtient la liste des attributs du module sys
>>> dir(sys)
['__displayhook__', '__doc__',
'argv', 'builtin_module_names',
...,
'version', 'version_info']

# récupère la liste des attributs pour le module courant
>>> dir()
['__builtins__', '__doc__',
'__name__', '__package__', 'sys']

# crée une nouvelle variable 'a'
>>> a = 5

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys', 'a']

# détruit/enlève un nom
>>> del a

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
```

Comment ça marche

D'abord, nous voyons l'utilisation de `dir` sur le module importé `sys`. Nous pouvons voir l'immense liste des attributs qu'il contient.

Ensuite, nous utilisons la fonction `dir` sans lui passer de paramètres. Par défaut, elle renvoie la liste des attributs du module courant. Notez que la liste des modules importés fait aussi partie de cette liste.

Afin d'étudier `dir` en action, nous définissons une nouvelle variable `a` et nous lui donnons une valeur et nous vérifions `dir` et nous notons qu'il y a une valeur supplémentaire dans la liste du même nom. Nous enlevons la variable/attribut du module courant en utilisant l'instruction `del` et la modification se voit lors de l'affichage de la fonction `dir`.

Une note sur `del` - cette instruction est utilisée pour *détruire* (*ndlt: delete*) un nom de variable et après l'exécution de l'instruction, dans ce cas `del a`, vous ne pouvez plus accéder à la variable `a` - c'est comme si elle n'avait jamais existé.

Notez que la fonction `dir()` marche avec *n'importe quel* objet. Par exemple, lancez `dir(print)` pour connaître les attributs de la fonction `print`, ou `dir(str)` pour les attributs de la classe `str` (chaîne de caractères).

Il existe également une fonction `vars()` qui peut potentiellement vous donner les attributs et leurs valeurs, mais elle ne fonctionnera pas dans tous les cas.

Packages

Maintenant, vous commencez à voir la hiérarchie pour organiser vos programmes. Les variables vont en général à l'intérieur des fonctions. Les fonctions et les variables globales vont en général à l'intérieur des modules. Et comment organiser des modules? C'est ici que les packages entrent en jeu.

Les packages sont juste des dossiers de modules avec un fichier spécial `__init__.py` qui indique à Python que ce dossier est spécial parce qu'il contient des modules Python.

Supposons que vous voulez créer un package appelé « `world` » avec des subpackages '`asia`', '`africa`', etc. et à leur tour ces subpackages contiennent des modules comme '`india`', '`madagascar`', etc.

Voici comment vous allez organiser les dossiers :

```
- <un dossier présent dans le sys.path>
  - world/
    - __init__.py
    - asia/
      - __init__.py
      - india/
        - __init__.py
        - foo.py
    - africa/
      - __init__.py
      - madagascar/
        - __init__.py
        - bar.py
```

Les packages sont juste une commodité pour organiser de manière hiérarchique les modules. Vous en verrez de nombreux cas dans la [bibliothèque standard](#).

Récapitulatif

Comme les fonctions sont des morceaux réutilisables de programmes, les modules sont des programmes réutilisables. Les packages sont une autre hiérarchie pour organiser les modules. La bibliothèque standard livrée avec Python est un exemple d'un tel ensemble de packages et modules.

Nous avons vu comment utiliser les modules et créer nos propres modules.

Nous allons maintenant voir des concepts intéressants appelés les structures de données.

Structures de données

Les structures de données sont juste cela - ce sont des *structures* qui peuvent contenir des *données*. En d'autres termes, elles sont utilisées pour stocker un ensemble de données liées entre elles.

Il y a quatre structures de données fournies dans Python - *les listes, les tuples, les dictionnaires et les ensembles*. Nous verrons maintenant comment utiliser chacune et comment nous simplifier la vie.

Liste

Une `liste` est une structure de données qui contient un ensemble ordonné d'objets, c'est-à-dire que vous pouvez stocker une *séquence* d'objets dans une liste. C'est facile à imaginer si vous pensez à une liste de commissions qui est une liste de choses à acheter, sauf que vous avez une chose par ligne, mais Python les présentera séparées par des virgules.

La liste de choses à acheter doit être entourée de crochets afin que Python comprenne que vous créez une liste. Une fois que vous avez créé une liste, vous pouvez ajouter, enlever ou rechercher un élément dans la liste. Comme nous pouvons ajouter ou enlever un élément, nous disons qu'une liste est un type de données modifiable (ndlt: *mutable* en anglais).

Introduction rapide aux objets et classes

Bien que j'aie retardé jusqu'à maintenant la discussion sur les objets et les classes, quelques explications vont vous permettre de mieux comprendre les listes. Nous verrons cela en détail plus tard dans [le chapitre dédié](#).

Une liste est un exemple de l'utilisation des objets et des classes. Quand nous utilisons une variable `i` et lui donnons une valeur, disons un entier `5`, nous pouvons voir cela comme créer un *objet* (c'est-à-dire une instance) `i` de *classe* (c'est-à-dire de type) `int`. En fait, vous pouvez lire `help(int)` pour mieux comprendre cela.

Une classe peut aussi avoir des *méthodes* c'est-à-dire des fonctions définies pour être utilisées seulement avec cette classe. Vous pouvez utiliser ces fonctionnalités seulement avec un objet de cette classe. Par exemple, Python fournit une méthode `append` pour la classe `list` ce qui vous permet d'ajouter un objet à la fin de la liste. Par exemple, `mylist.append('un objet')` va ajouter cette chaîne de caractères à la liste `mylist`. Notez l'utilisation de la notation pointée pour accéder aux méthodes des objets.

Une classe peut aussi avoir des *champs* qui sont juste des variables définies pour être utilisées dans le cadre de cette classe uniquement. Vous pouvez utiliser ces variables/noms seulement quand vous avez un objet de cette classe. Les champs sont aussi accédés avec la notation pointée, par exemple, `mylist.field`.

Exemple (enregistrez sous `ds_using_list.py`):

```
# Voici ma liste de commissions
shoplist = ['pomme', 'mangue', 'carotte', 'banane']

print('J\'ai', len(shoplist), 'objets à acheter.')

print('Ces objets sont :', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\nJe dois acheter du riz.')
shoplist.append('riz')
print('Ma liste de commissions contient maintenant ', shoplist)

print('Je vais trier ma liste maintenant')
shoplist.sort()
print('Ma liste triée est', shoplist)

print('Le premier objet que je vais acheter est', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('J\'ai acheté', olditem)
print('Ma liste de commissions est', shoplist)
```

Résultat:

```
$ python using_list.py
J'ai 4 objets à acheter.
Ces objets sont : pomme mangue carotte banane
Je dois acheter du riz.
Ma liste de commissions contient maintenant  ['pomme', 'mangue', 'carotte', 'banane', 'riz']
Je vais trier ma liste maintenant
Ma liste triée est ['banane', 'carotte', 'mangue', 'pomme', 'riz']
Le premier objet que je vais acheter est banane
J'ai acheté banane
Ma liste de commissions est ['carotte', 'mangue', 'pomme', 'riz']
```

Comment ça marche

La variable `shoplist` est une liste pour quelqu'un qui va faire ses courses au marché. Dans `shoplist`, nous stockons seulement les chaînes de caractères des noms des objets à acheter mais vous pouvez ajouter *n'importe quel type d'objet* à une liste, incluant des nombres et même d'autres listes.

Nous utilisons aussi la boucle `for..in` pour itérer les éléments de la liste. Vous réalisez maintenant qu'une liste est aussi une séquence. La spécificité des séquences sera vue plus tard dans la [section dédiée](#).

Notez l'utilisation du mot-clé `end` de la fonction `print` pour indiquer que nous voulons finir l'affichage avec un espace au lieu de l'habituel retour à la ligne.

Ensuite, nous ajoutons un élément à la liste en utilisant la méthode `append` de l'objet `list`, comme indiqué précédemment. puis, nous vérifions que l'élément a bien été ajouté à la liste en affichant le contenu de la liste avec l'instruction `print` qui l'affiche proprement. Puis, nous trions la liste avec la méthode `sort` de la liste. Il est important de comprendre que cette méthode modifie la liste elle-même et ne renvoie pas une liste modifiée - c'est différent de la manière dont les chaînes de caractères fonctionnent. C'est pour cela que nous disons que les listes sont modifiables (ndlt: *mutable*) et que les chaînes de caractères sont immuables (ndlt: *immutable*).

Ensuite, quand nous avons fini d'acheter des objets sur le marché, nous voulons les enlever de la liste. Nous faisons cela avec l'instruction `del`. Ici, nous indiquons quel élément de la liste nous voulons enlever et l'instruction `del` le fait pour nous. Nous indiquons que nous voulons enlever le premier élément de la liste et donc nous utilisons `del shoplist[0]` (souvenez-vous que Python compte à partir de 0).

Si vous voulez connaître toutes les méthodes définies par l'objet `list` voyez `help(list)` pour plus de détails.

Tuple

Les tuples sont utilisés pour contenir plusieurs objets. Voyez-les comme des listes, mais sans les fonctionnalités supplémentaires que la classe `list` vous apporte. Une caractéristique majeure des tuples est qu'ils sont *immuables* comme les chaînes de caractères, vous ne pouvez pas modifier les tuples.

Les tuples sont définis en indiquant des éléments séparés par des virgules, avec éventuellement une paire de parenthèses.

Les tuples sont en général utilisés dans des cas où une instruction ou une fonction définie par l'utilisateur peut légitimement supposer que la liste de valeurs, c'est-à-dire le tuple de valeurs, ne changera pas.

Exemple (enregistrez sous `ds_using_tuple.py`):

```
zoo = ('python', 'éléphant', 'pingouin')
# souvenez-vous, les parenthèses sont optionnelles
print('Le nombre d\'animaux est', len(zoo))

new_zoo = ('singe', 'chameau', zoo)
print('le nombre de cages dans le nouveau zoo est', len(new_zoo))
print('Les animaux dans le nouveau zoo sont', new_zoo)
print('Les animaux venant de l\'ancien zoo sont', new_zoo[2])
print('Le dernier animal venant de l\'ancien zoo sont', new_zoo[2][2])
print('Le nombre d\'animaux dans le nouveau zoo sont', len(new_zoo)-1+len(new_zoo[2]))
```

Résultat:

```
$ python using_tuple.py
Le nombre d'animaux est 3
Le nombre de cages dans le nouveau zoo est 3
Les animaux dans le nouveau zoo sont ('singe', 'chameau', ('python', 'éléphant', 'pingouin'))
Les animaux venant de l'ancien zoo sont ('python', 'éléphant', 'pingouin')
Le dernier animal venant de l'ancien zoo est pingouin
Le nombre d'animaux dans le nouveau zoo est 5
```

Comment ça marche

La variable `zoo` se réfère à un tuple d'éléments. Nous voyons que la fonction `len` peut être utilisée pour obtenir la longueur du tuple. Cela indique aussi qu'un tuple est également une [séquence](#).

Nous décalons ces animaux vers un nouveau zoo, car l'ancien est maintenant fermé. En conséquence, le tuple `new_zoo` contient quelques animaux qui étaient déjà là, et les animaux de l'ancien zoo. Retour à la réalité, notez qu'un tuple à l'intérieur d'un tuple ne perd pas son identité.

Nous pouvons accéder les éléments dans le tuple en indiquant la position de l'élément à l'intérieur d'une paire de crochets comme pour les listes. On appelle cela l'opérateur *d'indexation*. Nous accédons le troisième élément dans `new_zoo` en indiquant `new_zoo[2]` et nous accédons le troisième élément à l'intérieur du troisième élément à l'intérieur du tuple `new_zoo` en indiquant `new_zoo[2][2]`. C'est très simple une fois que vous avez compris le principe.

Tuple avec 0 ou 1 élément

Un tuple vide créé par une paire de parenthèses sans rien à l'intérieur comme `myempty = ()`. Cependant, un tuple avec un seul élément n'est pas si simple. Il faut le définir avec une virgule suivie du premier (et seul) élément, afin que Python puisse différencier un tuple d'une paire de parenthèses entourant l'objet dans une expression, c'est-à-dire que vous devez déclarer `singleton = (2,)` si vous voulez un tuple contenant l'élément `2`.

Note pour les programmeurs Perl

Une liste à l'intérieur d'une liste ne perd pas son identité, c'est-à-dire que les listes ne sont pas aplatis comme en Perl. Le même principe s'applique à un tuple à l'intérieur d'un tuple, ou un tuple à l'intérieur d'une liste, ou une liste à l'intérieur d'un tuple, etc. En Python, ils sont juste des objets stockés dans un autre objet, c'est tout.

Dictionnaire

Un dictionnaire est comme un carnet d'adresses dans lequel vous pouvez trouver une adresse et des renseignements sur une personne à partir de son nom, c'est-à-dire que nous associons des *clés* (nom) avec des *valeurs* (détails). Notez que la clé doit être unique, de la même manière que vous ne pouvez pas trouver une information correcte si vous avez deux personnes avec exactement le même nom.

Notez que pour les clés d'un dictionnaire, vous pouvez seulement utiliser des objets immuables (comme des chaînes de caractères), mais vous pouvez utiliser soit des objets immuables ou mutables pour les valeurs d'un dictionnaire. Cela revient à dire que pour les clés, vous devez seulement utiliser des simples objets.

Des paires de clés et valeurs sont spécifiées dans un dictionnaire avec la notation `d = {key1: value1, key2: value2}`. Notez que les paires de valeurs de clés sont séparées par le caractère deux-points et que les paires sont elles-mêmes séparées par une virgule et que tout cela est entouré d'accolades.

Souvenez-vous que les paires de valeurs de clé dans un dictionnaire ne sont pas ordonnées. Si vous voulez qu'elles soient dans l'ordre, il vous faudra les trier vous-mêmes avant de les utiliser.

Les dictionnaires que vous utiliserez sont des instances/objets de la classe `dict`.

Exemple (enregistrez sous `ds_using_dict.py`):

```
ab = {
    'Swaroop' : 'swaroop@swaroopch.com',
    'Larry' : 'larry@wall.org',
    'Matsumoto' : 'matz@ruby-lang.org',
    'Spammer' : 'spammer@hotmail.com'
}

print("L'adresse de Swaroop est", ab['Swaroop'])

# Destruction d'une paire de valeurs
del ab['Spammer']

print("\nIl y a {} contacts dans le carnet d'adresse\n".format(len(ab)))

for name, address in ab.items():
    print("Contactez {} à l'adresse {}".format(name, address))

# Ajout d'une paire de valeurs
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nL'adresse de Guido est", ab['Guido'])
```

Résultat:

```
$ python using_dict.py
L'adresse de Swaroop est swaroop@swaroopch.com

Il y a 3 contacts dans le carnet d'adresse

Contact Swaroop at swaroop@swaroopch.com
Contact Matsumoto at matz@ruby-lang.org
Contact Larry at larry@wall.org

L'adresse de Guido est guido@python.org
```

Comment ça marche

Nous créons le dictionnaire `ab` en utilisant la notation déjà vue. Puis nous accédons les paires de valeurs de clés en indiquant la clé avec l'opérateur d'indexation comme vu dans le contexte de listes et tuples. Observez la simplicité de la syntaxe.

Nous pouvons détruire les paires de valeurs de clés en utilisant notre vieil ami, l'instruction `del`. Nous indiquons simplement le dictionnaire et l'opérateur d'indexation pour la clé à enlever et nous la passons à l'instruction `del`. Il n'y a pas besoin de connaître la valeur correspondant à la clé pour cette opération.

Ensuite, nous accédons chaque paire de valeur de clé du dictionnaire avec la méthode `items` du dictionnaire qui retourne une liste de tuples, chaque tuple contient une paire d'éléments - la clé suivie de la valeur. Nous retrouvons cette paire et nous l'affectons aux variables `name` et `address` correspondantes pour chaque paire en utilisant la boucle `for..in` et nous affichons ces valeurs dans le bloc `for`.

Nous pouvons ajouter des nouvelles paires de valeurs de clés en utilisant juste l'opérateur d'indexation pour accéder une clé et lui affecter cette valeur, comme nous l'avons fait pour Guido dans le cas précédent.

Nous pouvons tester si une clé existe avec l'opérateur `in`.

Pour la liste des méthodes de la classe `dict`, voyez `help(dict)`.

Paramètres nommés et dictionnaires

Si vous avez utilisé des paramètres nommés dans vos fonctions, vous avez déjà utilisé des dictionnaires! Voyez-le ainsi - la paire de valeurs est indiquée par vous dans la liste des paramètres de la définition de la fonction et quand vous accédez les variables à l'intérieur de votre fonction, cela revient juste à accéder une clé dans un dictionnaire (qui est appelé la *table des symboles* dans la terminologie de conception des compilateurs).

Séquence

Les listes, tuples et chaînes de caractères sont des exemples de séquences, mais que sont les séquences et qu'il y a-t-il de tellement particulier à leur sujet ?

La principale fonctionnalité est qu'elles ont des test d'appartenance (c'est-à-dire les expressions `in` et `not in`) et les opérations d'indexage. L'opération d'*indexage* nous permet de rechercher directement un élément particulier dans la séquence.

Les trois types de séquences mentionnées au-dessus - listes, tuples et chaînes de caractères, ont aussi une opération de *tranchage* (ndlt: *slicing*) qui nous permet de retrouver une partie de la séquence.

Exemple (enregistrez sous `ds_seq.py`):

```
shoplist = ['pomme', 'mangue', 'carotte', 'banane']
name = 'swaroop'

# Indexation ou opération 'Subscription' operation
print("L'élément 0 est", shoplist[0])
print("L'élément 1 est", shoplist[1])
print("L'élément 2 est", shoplist[2])
print("L'élément 3 est", shoplist[3])
print("L'élément -1 est", shoplist[-1])
print("L'élément -2 est", shoplist[-2])
print("Le caractère 0 est", name[0])

# Slicing sur une liste
print("L'élément 1 à 3 est", shoplist[1:3])
print("L'élément 2 jusqu'à la fin est", shoplist[2:])
print("L'élément 1 à -1 est", shoplist[1:-1])
print("Les éléments du début à la fin sont", shoplist[:])

# Slicing sur une chaîne
print("Les caractères 1 to 3 sont", name[1:3])
print("Les caractères 2 à la fin sont", name[2:])
print("Les caractères 1 à -1 sont", name[1:-1])
print("Les caractères du début à la fin sont", name[:])
```

Résultat:

```

L'élément 0 est pomme
L'élément 1 est mangue
L'élément 2 est carotte
L'élément 3 est banane
L'élément -1 est banane
L'élément -2 est carotte
Le caractère 0 est s
L'élément 1 à 3 est ['mangue', 'carotte']
L'élément 2 jusqu'à la fin est ['carotte', 'banane']
L'élément 1 à -1 est ['mangue', 'carotte']
Les éléments du début à la fin sont ['pomme', 'mangue', 'carotte', 'banane']
Les caractères 1 to 3 sont wa
Les caractères 2 à la fin sont aroop
Les caractères 1 à -1 sont waroo
Les caractères du début à la fin sont swaroop

```

Comment ça marche

D'abord, nous voyons comment utiliser des index pour récupérer des éléments d'une séquence. On appelle aussi cela *operation de souscription*. Quand vous indiquez un nombre dans une séquence avec des square brackets comme montré au-dessus, Python va rechercher l'élément correspondant à la position dans cette séquence. Souvenez-vous que Python compte à partir de 0. En conséquence, `shoplist[0]` cherche le premier élément et `shoplist[3]` cherche le quatrième élément dans la séquence `shoplist`.

L'index peut être une valeur négative à partir de la fin de la séquence. Donc, `shoplist[-1]` fait référence au dernier élément de la séquence et `shoplist[-2]` récupère l'avant-dernier élément de la séquence.

L'opération de tranchage est utilisée en indiquant le nom de la séquence suivi par une paire optionnelle de nombres séparés par un caractère deux-points à l'intérieur des square brackets. Notez que ceci est très similaire à l'opération d'indexation utilisée jusque-là. Souvenez-vous que les nombres sont optionnels mais pas le caractère deux-points.

Le premier nombre (avant le deux-points) dans l'opération de tranchage fait référence à la position où démarre la tranche et le deuxième nombre (après le caractère deux-points) indique où s'arrête la tranche. Si le premier nombre n'est pas indiqué, Python va commencer au début de la séquence. Si le deuxième nombre est absent, Python va arrêter à la fin de la séquence. Notez que la tranche renvoyé *démarre* à ma position de début et va se terminer juste avant la position *fin* c'est-à-dire que la position de début est incluse dans la tranche et la position de fin est exclue de la séquence de tranchage.

Ainsi, `shoplist[1:3]` renvoie une tranche de la séquence en partant de la position 1, inclut la position 2 mais s'arrête à la position 3 et donc une *tranche* de deux éléments est retourné. De la même manière, `shoplist[:]` renvoie une copie de la séquence complète.

Vous pouvez aussi faire du tranchage avec des positions négatives. Les nombres négatifs sont utilisés pour des positions à partir de la fin de la séquence. Par exemple, `shoplist[:-1]` va renvoyer une tranche de la séquence qui exclut le dernier élément de la séquence mais contient tout le reste.

Vous pouvez aussi fournir un troisième argument pour la tranche, qui est le *pas* du tranchage (par défaut, le pas est de 1):

```

>>> shoplist = ['pomme', 'mangue', 'carotte', 'banane']
>>> shoplist[::1]
['pomme', 'mangue', 'carotte', 'banane']
>>> shoplist[::2]
['pomme', 'carotte']
>>> shoplist[::3]
['pomme', 'banane']
>>> shoplist[:::-1]
['banane', 'carotte', 'mangue', 'pomme']

```

Notez que quand le pas est de 2, nous obtenons les éléments en positions 0, 2, ... Quand le pas est de 3, nous obtenons les éléments en position 0, 3, etc.

Essayez diverses combinaisons de tranchage en utilisant l'interpréteur interactif Python, c'est-à-dire l'invite de commandes, afin de voir immédiatement les résultats. Ce qui est chouette avec les séquences, c'est que vous pouvez manipuler des tuples, des listes et des chaînes de caractères de la même manière !

Ensemble

Les ensembles (ndlt: `set` en anglais) sont des collections d'objets *sans ordre*. Ils sont utilisés quand l'existence d'un objet dans une collection est plus importante que l'ordre ou le nombre de fois qu'il y est.

En utilisant un ensemble, vous pouvez tester si un objet y est déjà présent, si c'est un sous-ensemble d'un autre ensemble, trouver son intersection avec un autre ensemble, et ainsi de suite.

```
>>> bri = set(['brésil', 'russie', 'inde'])
>>> 'inde' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('chine')
>>> bric.issuperset(bri)
True
>>> bri.remove('russie')
>>> bri & bric # OU bri.intersection(bric)
{'brésil', 'inde'}
```

Comment ça marche

Si vous vous souvenez des bases de la théorie des ensembles mathématiques, cet exemple s'explique par lui-même. Dans le cas contraire, cherchez « théorie des ensembles » et « diagramme de Venn » sur google pour mieux comprendre l'utilisation des ensembles en Python.

Références

Lorsque vous créez un objet et l'affectez à une variable, celle-ci ne fait que *référence* à l'objet et ne représente pas l'objet lui-même ! C'est-à-dire que le nom de la variable pointe vers la partie de la mémoire de votre ordinateur où l'objet est stocké. Cela s'appelle *lier* le nom à l'objet.

Généralement, vous n'avez pas à vous inquiéter à ce sujet, mais il existe un effet subtil dû aux références que vous devez connaître:

Exemple (enregistrez sous `ds_reference.py`):

```
print('Affectation Simple')
shoplist = ['pomme', 'mangue', 'carotte', 'banane']
# mylist est juste un autre nom pointant sur le même objet!
mylist = shoplist

# J'ai acheté le premier élément, donc je l'enlève de la liste
del shoplist[0]

print('La liste de commissions contient', shoplist)
print('maliste contient', mylist)
# notez que shoplist et maliste affichent
# la même liste la 'pomme', ce qui confirme
# qu'ils pointent sur le même objet

print('Copie en faisant une tranche complète')
# Copie en réalisant une tranche complète
mylist = shoplist[:]
# Retire le premier élément
del mylist[0]

print('La liste de commissions contient', shoplist)
print('maliste contient', mylist)
# Notez que maintenant les deux listes sont différentes
```

Résultat:

```
$ python reference.py
Affectation Simple
La liste de commissions contient ['mangue', 'carotte', 'banane']
maliste contient ['mangue', 'carotte', 'banane']
Copie en faisant un slice complet
La liste de commissions contient ['mangue', 'carotte', 'banane']
maliste contient ['carotte', 'banane']
```

Comment ça marche

L'essentiel des explications est disponible dans les commentaires.

Souvenez-vous que si vous voulez faire une copie d'une liste ou autre séquence ou d'objets complexes (pas des *objets* simples comme des entiers), alors vous devez utiliser l'opération de tranchage pour faire une copie. Si affectez juste une séquence à une nouvelle variable, les deux font faire *référence* au même objet, et cela peut poser problème si vous n'y êtes pas attentifs.

Note pour les programmeurs Perl

N'oubliez pas qu'une instruction d'affectation pour une liste ne crée **pas** une copie. Vous devez utiliser l'opération de tranchage pour copier la séquence.

Complément sur les chaînes de caractères

Nous avons déjà étudié les chaînes de caractères en détail plus tôt. Que pouvons-nous apprendre de plus ? Et bien, savez-vous que les chaînes de caractères sont aussi des objets et ont des méthodes qui font tout, depuis la vérification d'une partie d'une chaîne jusqu'à enlever des espaces ! En fait, vous utilisez déjà une méthode de chaîne ... la méthode `format` !

Les chaînes de caractères que vous utilisez dans les programmes sont toutes des objets de la classe `str`. Quelques méthodes de cette classe sont mises en évidence dans l'exemple suivant. Pour une liste complète de ces méthodes, voyez `help(str)`.

Exemple (enregistrez sous `ds_str_methods.py`):

```
nom = 'Swaroop' # Ceci est un objet de type chaîne

if nom.startswith('Sw'):
    print('Oui, la chaîne commence par "Sw"')

if 'a' in nom:
    print('Oui, elle contient la chaîne "a"')

if nom.find('ar') != -1:
    print('Oui, elle contient la chaîne "ar"')

delimiteur = '_*_'
maliste = ['Brésil', 'Russie', 'Inde', 'Chine']
print(delimiteur.join(maliste))
```

Résultat:

```
$ python str_methods.py
Oui, la chaîne commence par "Sw"
Oui, elle contient la chaîne "a"
Oui, elle contient la chaîne "ar"
Brésil_*_Russie_*_Inde_*_Chine
```

Comment ça marche

Ici, nous voyons en action de nombreuses méthodes des chaînes de caractères. La méthode `startswith` est utilisée pour trouver si une chaîne de caractères commence avec la chaîne indiquée. L'opérateur `in` est utilisé pour vérifier si une chaîne donnée fait partie de la chaîne de caractères.

La méthode `find` est utilisée pour trouver la position d'une chaîne donnée dans la chaîne, ou retourne -1 si elle ne trouve pas sous-chaîne. La classe `str` a aussi une méthode pour `join` (joindre) les items d'une séquence, avec la chaîne jouant le rôle de délimiteur entre chaque item de la séquence et retournant une chaîne plus longue.

Récapitulatif

Nous avons étudié les différentes structures de données disponibles dans Python en détail. Ces structures de données seront essentielles pour écrire des programmes de taille raisonnable.

Maintenant que nous possédons l'essentiel des bases de Python, nous allons concevoir et écrire un vrai programme Python.

Nous avons étudié diverses parties du langage Python et nous allons maintenant voir comment ces parties s'assemblent ensemble, en concevant et en écrivant un programme qui *fait* quelque chose d'utile. L'idée est d'apprendre à écrire un script Python.

Le problème

Le problème que nous voulons résoudre est :

je veux un programme qui fasse une sauvegarde de tous mes fichiers importants.

Bien que cela soit un problème simple, il n'y a pas assez d'information pour nous permettre de travailler à une solution. Nous avons besoin d'un peu plus d'*analyse*. Par exemple, comment choisir *quel*s fichiers vont être sauvegardés ? *Comment* les stocker ? *Où* les stocker ?

Après une analyse du problème, nous *concevons* notre programme. Nous écrivons une liste de choses que notre programme doit faire. Dans ce cas, j'ai créé la liste suivante sur la manière dont *je veux* qu'il fonctionne. Si vous concevez ce programme, vous fairez sans doute une analyse différente, car chaque personne a sa manière de faire, et ce sera tout aussi juste.

- Les fichiers et répertoires à sauvegarder seront dans une liste.
- La sauvegarde doit être stocké dans un répertoire contenant toutes les sauvegardes.
- Les fichiers seront sauvegardés dans une archive *zip*.
- Le nom de l'archive zip sera la date et l'heure courante.
- Nous utilisons la commande standard `zip` disponible par défaut dans toute distribution GNU/Linux ou Unix standard. Notez que vous pouvez utiliser n'importe quelle commande d'archivage à condition qu'elle dispose d'une interface en ligne de commande.

Pour les utilisateurs Windows

Les utilisateurs Windows peuvent [installer](#) la commande `zip` à partir de [la page projet de GnuWin32](#) et ajouter `C:\Program Files\GnuWin32\bin` à la variable d'environnement `PATH` de leur système, semblable à [ce que nous avons fait pour la commande python elle-même](#).

La solution

Comme la conception de notre programme est maintenant suffisamment stable, nous pouvons écrire le code qui est une *implémentation* de notre solution.

Enregistrez sous `backup_ver1.py` :

```

import os
import time

# 1. Les fichiers et répertoires à sauvegarder sont indiqués dans une liste.
# Notez que nous devons utiliser des guillemets doubles dans une chaîne
# pour les noms avec des espaces.
# Par exemple sous Windows:
# source = ['"C:\\My Documents"']
# Nous aurions pu aussi utiliser une chaîne brute en écrivant [r'"C:\\My Documents"].
# Par exemple sous Mac OS X et Linux:
source = ['/Users/swa/notes']

# 2. La sauvegarde est stockée dans un répertoire
# Par exemple sous Windows:
# target_dir = 'E:\\Backup'
# Par exemple sous Mac OS X et Linux:
target_dir = '/Users/swa/backup'
# N'oubliez pas de changer le dossier que vous voudrez utiliser

# 3. Les fichiers sauvegardés sont mis dans une archive zip.
# 4. Le nom de l'archive zip contient la date et l'heure courante
target = target_dir + os.sep + \
         time.strftime('%Y%m%d%H%M%S') + '.zip'

# Crée le répertoire de destination, s'il n'existe pas
if not os.path.exists(target_dir):
    os.mkdir(target_dir) # Creation répertoire

# 5. Nous utilisons la commande zip pour mettre les fichiers dans une archive zip
zip_command = 'zip -r {0} {1}'.format(target,
                                         ' '.join(source))

# Lancement de la sauvegarde
print('La commande zip est :')
print(zip_command)
print('En cours :')
if os.system(zip_command) == 0:
    print('Sauvegarde réussie dans', target)
else:
    print('Sauvegarde ÉCHOUÉE')

```

Résultat:

```

$ python backup_ver1.py
La commande zip est :
zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes
En cours :
    adding: Users/swa/notes/ (stored 0%)
    adding: Users/swa/notes/blah1.txt (stored 0%)
    adding: Users/swa/notes/blah2.txt (stored 0%)
    adding: Users/swa/notes/blah3.txt (stored 0%)
Sauvegarde réussie dans /Users/swa/backup/20140328084844.zip

```

Maintenant, nous sommes dans la phase de *test*, où nous testons notre programme. S'il ne se comporte pas comme prévu, nous allons le *debuguer* c'est-à-dire enlever les *bugs* (erreurs) dans le programme.

Si le programme ci-dessus ne fonctionne pas pour vous, copiez la ligne imprimée après la ligne `La commande zip est :` dans la sortie, collez-la dans le shell (sous GNU/Linux et Mac OS X) / `cmd` (sous Windows), voyez quelle est l'erreur et essayez de la réparer. Consultez également le manuel de commande `zip` sur ce qui pourrait ne pas être correct. Si cette commande réussit, le problème peut provenir du programme Python lui-même. Vérifiez donc s'il correspond exactement au programme écrit ci-dessus.

Comment ça marche

Vous noterez comment nous avons converti notre *conception en code* en l'écrivant pas à pas.

Nous utilisons les modules `os` et `time` en commençant par les importer. Puis, nous indiquons les fichiers et répertoires à sauvegarder dans la liste `source`. Le répertoire de destination est l'endroit où nous stockons tous les fichiers sauvegardés, et ceci est indiqué dans la variable `target_dir`. Le nom de l'archive zip que nous allons créer est la date et l'heure courante, que nous générerons avec la fonction `time.strftime()`. L'archive aura une extension `.zip` et sera stockée dans le répertoire `target_dir`.

Notez l'utilisation de la variable `os.sep` - cela donne le nom du séparateur de répertoire en fonction du système d'exploitation, c'est-à-dire que cela sera `'/'` avec Linux et Unix, `'\\'` avec Windows et `':'` avec Mac OS. Le fait d'utiliser directement `os.sep` au lieu de ces caractères rend notre programme portable et il fonctionnera sous ces trois systèmes d'exploitation.

La fonction `time.strftime()` prend des arguments comme ceux utilisés dans le programme ci-dessus. La spécification `%Y` sera remplacée par l'année dans le siècle. La spécification `%m` sera remplacée par le mois en tant que nombre compris entre `01` et `12` et ainsi de suite. La liste complète de ces spécifications peut être trouvée dans le [manuel de référence Python](#).

Nous créons le nom du fichier zip cible en utilisant l'opérateur d'addition qui *concatène* les chaînes de caractère, c'est-à-dire qu'il assemble les deux chaînes ensemble et en renvoie une nouvelle. Ensuite, nous créons une chaîne `zip_command` qui contient la commande que nous allons exécuter. Vous pouvez vérifier si votre commande fonctionne en la lançant dans le shell (terminal Linux ou invite de commandes DOS).

La commande `zip` que nous utilisons a quelques options disponibles, et l'une de ces options est `-r`. L'option `-r` indique que la commande zip doit fonctionner de manière **récursive** sur les répertoires, c'est-à-dire qu'elle doit inclure tous les sous-répertoires et fichiers. Les options sont suivies du nom de l'archive zip à créer, puis de la liste des fichiers et répertoires à sauvegarder. Nous convertissons la liste `source` en une chaîne avec la méthode `join` que nous avons déjà vue.

Ensuite, nous *exécutons* la commande en utilisant la fonction `os.system` qui lance la commande comme si elle était lancée à partir du *système* c'est-à-dire dans une invite de commandes - il retourne `0` si la commande s'est exécutée avec succès, sinon il renvoie un numéro d'erreur.

En fonction du résultat de la commande, nous affichons le message approprié indiquant que la sauvegarde a échouée ou réussie.

Ca y est, nous avons créé un script pour faire une sauvegarde de nos fichiers importants !

Note pour les utilisateurs Windows

À la place d'échapper les backslash (`\`), vous pouvez utiliser des chaînes brutes. Par exemple, utilisez `'C:\\Documents'` ou `r'C:\Documents'`. Cependant, n'utilisez pas `'C:\Documents'` car vous vous retrouveriez avec un caractère d'échappement inconnu `\D`.

Maintenant que nous avons un script de sauvegarde qui fonctionne, nous pouvons l'utiliser quand nous voulons une sauvegarde de nos fichiers. Cela est appelé la phase d'*opération* ou la phase de *déploiement* du logiciel.

Le programme ci-dessus fonctionne correctement, mais (en général) la première version d'un programme ne fonctionne pas comme prévu. Par exemple, il peut y avoir des problèmes si vous n'avez pas réfléchi correctement à votre programme ou si vous avez fait une faute de frappe en rentrant le code, etc. Dans ce cas, il vous faudra revenir à la phase de conception ou déboguer votre programme.

Deuxième version

La première version de notre script fonctionne. Cependant, nous pouvons faire quelques améliorations, afin qu'il soit plus adapté pour une utilisation journalière. Ceci est appelé la phase de *maintenance* du logiciel.

Une amélioration que je trouvais utile était un meilleur nommage des fichiers - utiliser l'*heure* en tant que nom de fichier dans un répertoire nommé en fonction de la *date* dans le répertoire principal contenant les sauvegardes. Le premier avantage est que vos sauvegardes sont stockés de façon hiérarchique et donc plus faciles à gérer. Le deuxième avantage est que les noms de fichiers sont plus courts. Le troisième avantage est que des répertoires séparés vous aideront à vérifier si vous avez fait des sauvegardes pour chaque jour, vu que le répertoire ne sera créé que si vous avez réalisé une sauvegarde pour cette journée.

Enregistrez sous `backup_ver2.py` :

```

import os
import time

# 1. Les fichiers et répertoires à sauvegarder sont indiqués dans une liste.
# Notez que nous devons utiliser des guillemets doubles dans une chaîne
# pour les noms avec des espaces.
# Par exemple sous Windows:
# source = ['"C:\\My Documents"']
# Nous aurions pu aussi utiliser une chaîne brute en écrivant [r'"C:\\My Documents"].
# Par exemple sous Mac OS X et Linux:
source = ['/Users/swa/notes']

# 2. La sauvegarde est stockée dans un répertoire
# Par exemple sous Windows:
# target_dir = 'E:\\Backup'
# Par exemple sous Mac OS X et Linux:
target_dir = '/Users/swa/backup'
# N'oubliez pas de changer le dossier que vous voudrez utiliser

# Crée le répertoire de destination, s'il n'existe pas
if not os.path.exists(target_dir):
    os.mkdir(target_dir) # Creation répertoire

# 3. Les fichiers sont placés dans une archive zip.
# 4. Le jour courant est le nom du sous-répertoire dans le répertoire principal
today = target_dir + os.sep + time.strftime('%Y%m%d')
# L'heure courante est le nom de l'archive zip
now = time.strftime('%H%M%S')

# Le nom du fichier zip
target = today + os.sep + now + '.zip'

# Créer le sous-répertoire s'il n'existe pas
if not os.path.exists(today):
    os.mkdir(today)
    print('Création réussie du répertoire', today)

# 5. Nous utilisons la commande zip pour créer une archive
zip_command = 'zip -r {0} {1}'.format(target,
                                         ' '.join(source))

# Lancement de la sauvegarde
print('La commande zip est :')
print(zip_command)
print('En cours :')
if os.system(zip_command) == 0:
    print('Sauvegarde réussie dans', target)
else:
    print('Sauvegarde ÉCHOUÉE')

```

Résultat:

```

$ python backup_ver2.py
Création réussie du répertoire /Users/swa/backup/20140329
La commande zip est :
zip -r /Users/swa/backup/20140329/073201.zip /Users/swa/notes
En cours :
    adding: Users/swa/notes/ (stored 0%)
    adding: Users/swa/notes/blah1.txt (stored 0%)
    adding: Users/swa/notes/blah2.txt (stored 0%)
    adding: Users/swa/notes/blah3.txt (stored 0%)
Sauvegarde réussie dans /Users/swa/backup/20140329/073201.zip

```

Comment ça marche

L'essentiel du programme reste le même. Les changements sont que nous vérifions si un répertoire avec le jour courant dans le répertoire principal des sauvegardes existe, en utilisant la fonction `os.path.exists`. S'il n'existe pas, nous le créons avec la fonction `os.mkdir`.

Troisième version

La deuxième version fonctionne bien quand je veux de nombreux backups, mais dans ce cas, j'ai du mal à savoir pourquoi j'ai fait ces sauvegardes ! Par exemple, si j'ai fait des modifications importantes dans un programme ou une présentation, je veux pouvoir associer ces changements avec le nom de l'archive. Cela peut être facilement obtenu en ajoutant un commentaire de l'utilisateur au nom de l'archive zip.

ATTENTION: Ce programme ne fonctionne pas, mais ne vous inquiétez pas, continuez, il y a une leçon à en tirer.

Enregistrez sous `backup_ver3.py` :

```
import os
import time

# 1. Les fichiers et répertoires à sauvegarder sont indiqués dans une liste.
# Notez que nous devons utiliser des guillemets doubles dans une chaîne
# pour les noms avec des espaces.
# Par exemple sous Windows:
# source = ['"C:\\My Documents"']
# Nous aurions pu aussi utiliser une chaîne brute en écrivant [r'"C:\\My Documents"].
# Par exemple sous Mac OS X et Linux:
source = ['/Users/swa/notes']

# 2. La sauvegarde est stockée dans un répertoire
# Par exemple sous Windows:
# target_dir = 'E:\\\\Backup'
# Par exemple sous Mac OS X et Linux:
target_dir = '/Users/swa/backup'
# N'oubliez pas de changer le dossier que vous voudrez utiliser

# Crée le répertoire de destination, s'il n'existe pas
if not os.path.exists(target_dir):
    os.mkdir(target_dir) # Creation répertoire

# 3. Les fichiers sont placés dans une archive zip.
# 4. Le jour courant est le nom du sous-répertoire dans le répertoire principal
today = target_dir + os.sep + time.strftime('%Y%m%d')
# L'heure courante est le nom de l'archive zip
now = time.strftime('%H%M%S')

# Un commentaire de l'utilisateur est ajouté au nom du fichier zip
comment = input('Saisissez un commentaire --> ')
# Vérifie si un commentaire a été saisi
if len(comment) == 0:
    target = today + os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' +
        comment.replace(' ', '_') + '.zip'

# Crée le sous-répertoire s'il n'existe pas déjà.
if not os.path.exists(today):
    os.mkdir(today)
    print('Successfully created directory', today)

# 5. Nous utilisons la commande zip pour créer une archive
zip_command = 'zip -r {0} {1}'.format(target,
                                         ' '.join(source))

# Lancement de la sauvegarde
print('La commande zip est :')
print(zip_command)
print('En cours :')
if os.system(zip_command) == 0:
    print('Sauvegarde réussie dans', target)
else:
    print('Sauvegarde ÉCHOUÉE')
```

Résultat:

```
$ python backup_ver3.py
  File "backup_ver3.py", line 39
    target = today + os.sep + now + '_' +
              ^
SyntaxError: invalid syntax
```

Comment ça (ne) marche (pas)

Ce programme ne fonctionne pas ! Python dit qu'il y a une erreur de syntaxe, ce qui signifie que le script ne respecte pas les règles que Python s'attend à trouver. Quand nous regardons l'erreur indiquée par Python, il nous indique l'endroit où a été trouvée l'erreur. Donc nous commençons le *débogage* de notre programme à partir de cette ligne.

En regardant attentivement, nous voyons que la seule ligne logique a été coupée en deux lignes physiques, mais nous n'avons pas indiqué que les deux lignes physiques vont ensemble. Fondamentalement, Python a trouvé l'opérateur d'addition (+) sans aucune opérande dans cette ligne logique et en conséquence ne sait pas comment continuer. Souvenez-vous que nous indiquons que la ligne logique continue sur la prochaine ligne physique en utilisant un backslash à la fin de la ligne physique. Donc, nous faisons cette correction à notre programme. Cette correction du programme quand nous trouvons des erreurs est appelée *correction de bugs*.

Quatrième version

Enregistrez sous `backup_ver4.py` :

```

import os
import time

# 1. Les fichiers et répertoires à sauvegarder sont indiqués dans une liste.
# Notez que nous devons utiliser des guillemets doubles dans une chaîne
# pour les noms avec des espaces.
# Par exemple sous Windows:
# Source = ['"C:\\My Documents"']
# Nous aurions pu aussi utiliser une chaîne brute en écrivant [r'"C:\\My Documents"].
# Par exemple sous Mac OS X et Linux:
source = ['/Users/swa/notes']

# 2. La sauvegarde est stockée dans un répertoire
# Par exemple sous Windows:
# target_dir = 'E:\\Backup'
# Par exemple sous Mac OS X et Linux:
target_dir = '/Users/swa/backup'
# N'oubliez pas de changer le dossier que vous voudrez utiliser

# Crée le répertoire de destination, s'il n'existe pas
if not os.path.exists(target_dir):
    os.mkdir(target_dir) # Creation répertoire

# 3. Les fichiers sont placés dans une archive zip.
# 4. Le jour courant est le nom du sous-répertoire dans le répertoire principal
today = target_dir + os.sep + time.strftime('%Y%m%d')
# L'heure courante est le nom de l'archive zip
now = time.strftime('%H%M%S')

# Un commentaire de l'utilisateur est ajouté au nom du fichier zip
comment = input('Saisissez un commentaire --> ')
# Vérifie si un commentaire a été saisi
if len(comment) == 0:
    target = today + os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' + \
        comment.replace(' ', '_') + '.zip'

# Crée le sous-répertoire s'il n'existe pas déjà.
if not os.path.exists(today):
    os.mkdir(today)
    print('Successfully created directory', today)

# 5. Nous utilisons la commande zip pour créer une archive
zip_command = 'zip -r {0} {1}'.format(target,
                                         ' '.join(source))

# Lancement de la sauvegarde
print('La commande zip est :')
print(zip_command)
print('En cours :')
if os.system(zip_command) == 0:
    print('Sauvegarde réussie dans', target)
else:
    print('Sauvegarde ÉCHOUÉE')

```

Résultat:

```

$ python backup_ver4.py
Saisissez un commentaire --> ajout de nouveaux exemples
La commande zip est :
zip -r /Users/swa/backup/20140329/074122_ajout_de_nouveaux_exemples.zip /Users/swa/notes
En cours :
    adding: Users/swa/notes/ (stored 0%)
    adding: Users/swa/notes/blah1.txt (stored 0%)
    adding: Users/swa/notes/blah2.txt (stored 0%)
    adding: Users/swa/notes/blah3.txt (stored 0%)
Sauvegarde réussie dans /Users/swa/backup/20140329/074122_ajout_de_nouveaux_exemples.zip

```

Comment ça marche

Maintenant le programme fonctionne ! Regardons les améliorations que nous avons ajoutées dans la version 3. Nous saisissons les commentaires de l'utilisateur avec la fonction `input` et nous vérifions si quelque chose a été saisi en trouvant la longueur de l'entrée avec la fonction `len`. Si l'utilisateur a juste tapé `enter` sans rien saisir (c'était peut-être une sauvegarde quelconque ou sans modification), alors nous faisons comme avant.

Cependant, si un commentaire a été saisi, alors il est ajouté au nom de l'archive zip juste avant l'extension `.zip`. Notez que nous remplaçons les espaces dans le commentaire par des underscores - parce qu'il est plus facile de gérer des noms de fichiers sans des espaces.

Possibilités d'améliorations

La quatrième version est un script qui fonctionne de manière satisfaisante pour la plupart des utilisateurs, mais il y a toujours place à amélioration. Par exemple, vous pouvez inclure un niveau *verbosité* pour la commande zip en spécifiant l'option `-v` pour que votre programme devienne plus bavard ou l'option `-q` pour le rendre *silencieux* (ndlt: `quiet`).

Une autre amélioration possible serait de permettre de passer au script en ligne de commande des fichiers et répertoires en plus. Nous pouvons récupérer ces noms à partir de la liste `sys.argv` et nous pouvons les ajouter à notre liste `source` en utilisant la méthode `extend` fournie par la classe `list`.

L'amélioration la plus importante serait de ne pas utiliser la méthode `os.system` pour créer les archives, mais plutôt les modules intégrés `zipfile` ou `tarfile`. Ils font partie de la bibliothèque standard et sont déjà disponibles pour que vous puissiez utiliser le script sans la dépendance externe envers le programme zip installé sur votre ordinateur.

J'ai utilisé la méthode `os.system` pour créer une sauvegarde dans les exemples ci-dessus à des fins purement pédagogiques, de sorte que cet exemple soit suffisamment simple pour être compris par tout le monde, mais suffisamment réel pour être utile.

Pouvez-vous essayer d'écrire la cinquième version qui utilise le module [zipfile] (<http://docs.python.org/3/library/zipfile.html>) à la place de l'appel `os.system` ?

Le cycle de développement

Nous avons parcouru les différentes *phases* de l'écriture d'un logiciel. Ces phases peuvent être résumées comme suit :

- Quoi (Analyse)
- Comment (Conception)
- Le faire (Implémentation)
- Test (Test et Débogage)
- Utilisation (Opération ou Déploiement)
- Maintenance (Amélioration)

Une manière recommandée d'écrire un programme est la procédure que nous avons suivie en créant le script backup : faire l'analyse et la conception. Commencer à implémenter une version simple. Tester et déboguer. L'utiliser pour s'assurer qu'il fonctionne comme prévu. Maintenant, ajouter les fonctionnalités que vous voulez et continuer le cycle Créer-Tester-Utiliser autant de fois que nécessaire.

Souvenez-vous:

Un logiciel ne se fabrique pas, il se cultive. -- Bill de hÓra

Récapitulatif

Nous avons vu comment créer notre programme/script Python et les différentes étapes dans l'écriture d'un tel programme. Vous trouverez utile de créer vos programmes comme nous l'avons fait dans ce chapitre, ainsi vous serez plus à l'aise avec Python comme dans la résolution de problèmes.

Ensuite, nous allons parler de programmation orientée objet.

Programmation orientée objet

Dans tous les programmes que nous avons écrits jusque-là, nous avons conçu notre programme autour des fonctions, c'est-à-dire des blocs d'instructions qui manipulent des données. C'est ce qu'on appelle la programmation *orientée procédure*. Il y a une autre façon d'organiser votre programme, qui est de combiner les données et les fonctionnalités et de tout emballer à l'intérieur de quelque chose qu'on appelle un objet. C'est ce qu'on appelle le paradigme de la programmation *orientée objet*. La plupart du temps vous pouvez utiliser la programmation procédurale, mais quand vous écrivez des programmes de taille importante, ou avez un problème qui se résoud mieux avec cette méthode, vous pouvez utiliser les techniques de la programmation orientée objet.

Les classes et les objets sont les deux principaux aspects de la programmation orientée objet. Une **classe** crée un nouveau *type* où les **objets** sont des **instances** de la classe. Une analogie est que vous pouvez avoir des variables de type `int` ce qui revient à dire que les variables qui stockent des entiers sont des variables qui sont des instances (des objets) de la classe `int`.

Note pour les programmeurs en langages statiques Notez que même les entiers sont traités en tant qu'objets (de la classe `int`). Cela est le contraire de C++ et Java (avant la version 1.5) où les entiers sont des types primitifs natifs.

Voyez `help(int)` pour plus de détails sur cette classe.

Les programmeurs C# et Java 1.5 trouveront cela similaire au concept de *boxing* et *unboxing*.

Les objets peuvent stocker des données en utilisant des variables ordinaires qui *appartiennent* à l'objet. Les variables qui appartiennent à un objet ou à une classe sont appelés des **champs**. Les objets peuvent aussi avoir des fonctionnalités en utilisant des fonctions qui *appartiennent* à une classe. De telles fonctions sont appelées les **méthodes** de la classe. Cette terminologie est importante parce qu'elle nous aide à différencier les fonctions et variables qui sont indépendantes et celles qui appartiennent à une classe ou un objet. Collectivement, on fait référence aux champs et méthodes en tant qu'**attributs** de cette classe.

Les champs sont de deux types, ils peuvent appartenir à chaque instance/objet de la classe ou ils peuvent appartenir à la classe elle-même. On les appelle respectivement les **variables d'instance** et les **variables de classe**.

Une classe est créée en utilisant le mot-clé `class`. Les champs et méthodes de la classe sont listés dans un bloc indenté.

Le paramètre `self`

Les méthodes d'une classe ont une seule différence avec les fonctions ordinaires - elles ont un nom en plus qui doit être ajouté au début de la liste des paramètres, mais vous ne devez **pas** donner une valeur à ce paramètre quand vous appelez la méthode, Python le fournira. Cette variable particulière fait référence à l'objet *lui-même*, et par convention on lui donne le nom de `self`.

Vous pouvez donner n'importe quel nom à ce paramètre, mais il est *fortement recommandé* d'utiliser le nom `self`, tout autre nom est mal vu. Il y a de nombreux avantages à utiliser un nom standard: n'importe quelle personne lisant votre programme le reconnaîtra immédiatement et même des EDIs spécialisés (Environnement de Développement Intégré) vous aideront si vous utilisez `self`.

Note pour les programmeurs C++/Java/C#

Le `self` en Python est équivalent au pointeur `this` en C++ et à la référence `this` en Java et C#.

Vous vous demandez comment Python donne une valeur à `self` et pourquoi vous n'avez pas besoin de lui en fournir une. Un exemple va clarifier cela. Disons que vous avez une classe appelée `MyClass` et une instance de cette classe appelée `myobject`. Quand vousappelez une méthode de cet objet en tant que `myobject.method(arg1, arg2)`, cela est automatiquement converti par Python en `MyClass.method(myobject, arg1, arg2)`, c'est tout ce qu'il y a à dire sur le `self`.

Cela signifie aussi que si vous avez une méthode qui ne prend pas d'argument, alors elle a quand même un argument: `self`.

Classes

La classe la plus simple possible est montrée dans l'exemple suivant (enregistrez sous `oop_simplestclass.py`).

```
class Person:
    pass # Un bloc vide

p = Person()
print(p)
```

Résultat:

```
$ python simplestclass.py
<__main__.Person object at 0x019F85F0>
```

Comment ça marche

Nous créons une nouvelle classe avec l'instruction `class` et le nom de la classe. Suit un bloc indenté d'instructions qui forment le corps de la classe. Dans ce cas, nous avons un bloc vide qui est indiqué par l'instruction `pass`.

Ensuite, nous créons un objet/instance de cette classe en utilisant le nom de la classe suivi d'une paire de parenthèses (nous en apprendrons [plus sur les instantiations](#) dans la prochaine section). Pour vérifier, nous confirmons le type de la variable en l'affichant. Cela nous dit que nous avons une instance de la classe `Person` dans le module `__main__`.

Notez que l'adresse de la mémoire de l'ordinateur où l'objet est stocké est affichée. Cette adresse aura une autre valeur sur votre ordinateur car Python va stocker l'objet n'importe où, là où il trouvera de la place.

Méthodes

Nous avons déjà vu que les classes/objets peuvent avoir des méthodes comme les fonctions, avec la différence que nous avons une variable `self` en plus. Voyons avec un exemple (enregistrez sous `oop_method.py`).

```
class Person:
    def say_hi(self):
        print('Bonjour, ça va ?')

p = Person()
p.say_hi()

# Ce court exemple peut aussi s'écrire
# Person().say_hi()
```

Résultat:

```
$ python method.py
Bonjour, ça va ?
```

Comment ça marche

Nous voyons ici `self` en action. Notez que la méthode `say_hi` ne prend pas de paramètres, mais possède `self` dans la définition de la fonction.

La méthode `__init__`

De nombreuses méthodes ont une signification particulière pour les classes Python. Nous allons voir la signification de la méthode `__init__` maintenant.

La méthode `__init__` est exécutée dès qu'un objet d'une classe est instancié. Cette méthode est utile pour exécuter n'importe quelle *initialisation* que vous voulez exécuter pour votre objet. Notez les double underscores à la fois au début et à la fin du nom.

Exemple (enregistrez sous `oop_init.py`):

```

class Person:
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print("Bonjour, je m'appelle", self.name)

p = Person('Swaroop')
p.say_hi()
# Ce court exemple peut aussi s'écrire
# Person().say_hi()

```

Résultat:

```

$ python class_init.py
Bonjour, je m'appelle Swaroop

```

Comment ça marche

Ici, nous définissons la méthode `__init__` comme prenant un paramètre `name` (avec l'habuel `self`). Puis, nous créons un nouveau champ également appelé `name`. Notez que ce sont deux variables différentes même si elles sont toutes les deux appelées « `name` ». Grâce à la notation pointée `self.name`, il n'y a pas de problème, il y a quelque chose appelé "name" qui fait partie de l'objet appelé "self" et l'autre `name` est une variable locale. Comme nous indiquons explicitement à quel `name` nous faisons référence, il n'y a pas de confusion possible.

Quand nous créons une nouvelle instance `p` de la classe `Person`, nous utilisons le nom de la classe, suivi des arguments entre parenthèses: `p = Person('Swaroop')`

Nous n'appelons pas explicitement la méthode `__init__`. C'est la signification spéciale de cette méthode.

Maintenant, nous pouvons utiliser le champ `self.name` dans nos méthodes, ce qui est démontré dans la méthode `say_hi`.

Classe et variable d'objets

Nous avons déjà vu la partie fonctionnalité des classes et objets (c'est-à-dire les méthodes), apprenons maintenant la partie données. La partie données, c'est-à-dire les champs, n'est rien d'autre que des variables ordinaires qui sont *liées à l'espace de noms* des classes et objets. Cela veut dire que ces noms sont valides seulement à l'intérieur du contexte de ces classes et objets. Voilà pourquoi on les appelle des *espaces de noms*.

Il y a deux types de *champs*, les variables de classe et les variables objets qui sont classées en fonction de la classe ou de l'objet qui les possèdent respectivement.

Les variables de classe sont partagées, elles peuvent être accédées par toutes les instances de cette classe. Il n'y a qu'une seule copie de la variable de classe et quand n'importe quel objet modifie une variable de classe, ce changement est vu par toutes les autres instances.

Les variables d'objets appartiennent à chaque objet/instance individuel de la classe. Dans ce cas, chaque objet a sa propre copie du champ, c'est-à-dire que ces copies ne sont pas partagées et n'ont aucun rapport avec le champ portant le même nom dans un instance différente. Un exemple va nous aider à comprendre cela (enregistrez sous `oop_objvar.py`):

```

class Robot:
    """Représente un robot, avec un nom."""

    # Une variable de classe, qui compte le nombre de robots
    population = 0

    def __init__(self, name):
        """Initialise les données."""
        self.name = name
        print('Initialisation {}'.format(self.name))

    # Quand ce robot est créé, il est ajouté à la population
    Robot.population += 1

    def die(self):
        """Je meurs."""
        print('{} est détruit !'.format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print('{} était le dernier.'.format(self.name))
        else:
            print('Il y a encore {} robots au travail.'.format(Robot.population))

    def say_hi(self):
        """Bonjour du robot.

        Oui, ils peuvent faire cela."""
        print('Bonjour, mes maîtres m\'appellent {}.'.format(self.name))

    @classmethod
    def how_many(cls):
        """Affiche la population actuelle."""
        print('Nous avons {} robots.'.format(cls.population))

droid1 = Robot('R2-D2')
droid1.say_hi()
Robot.how_many()

droid2 = Robot('C-3PO')
droid2.say_hi()
Robot.how_many()

print("\nLes robots peuvent faire un travail ici.\n")

print("Les robots ont terminé leur travail. Donc détruisons-les.")
droid1.die()
droid2.die()

Robot.how_many()

```

Résultat:

```

(Initialisation R2-D2)
Bonjour, mes maîtres m'appellent R2-D2.
Nous avons 1 robots.
(Initialisation C-3PO)
Bonjour, mes maîtres m'appellent C-3PO.
Nous avons 2 robots.

Les robots peuvent faire un travail ici.

Les robots ont terminé leur travail. Donc détruisons-les.
R2-D2 est détruit !
Il y a encore 1 robots au travail.
C-3PO est détruit !
C-3PO était le dernier.
Nous avons 0 robots.

```

Comment ça marche

Cet exemple est long, mais nous aide à démontrer la nature des variables et objets de classe. Ici, `population` appartient à la classe `Robot` et est donc une variable de classe. La variable `name` appartient à l'objet (il est assigné en utilisant le `self`) et est donc une variable de l'objet.

Ensuite, nous faisons référence à la variable de classe `population` en tant que `Robot.population` et pas en tant que `self.population`. Nous faisons référence à la variable objet `name` avec la notation `self.name` dans les méthodes de cet objet. Souvenez-vous de cette simple différence entre variable de classe et variable objet. Notez aussi qu'une variable objet avec le même nom qu'une variable de classe va cacher la variable de classe !

À la place d'écrire `Robot.population`, nous aurions aussi pu utiliser `self.__class__.population` car tout objet peut référer à sa classe à travers l'attribut `self.__class__`.

`how_many` est en fait une méthode qui appartient à la classe et pas à l'objet. Cela veut dire que nous pouvons le définir soit en tant que `classmethod` ou `staticmethod` selon que nous avons besoin de savoir de quelle classe nous faisons partie. Comme nous faisons référence à une variable de classe, utilisons `classmethod`.

Nous avons annoté la méthode `how_many` en tant que méthode de classe utilisant un [décorateur](#).

On peut imaginer que les décorateurs sont un raccourci pour appeler une fonction enveloppante (c.-à-d. Une fonction qui en « enveloppe » une autre fonction afin qu'elle puisse faire quelque chose avant ou après la fonction interne), appliquer le décorateur `@classmethod` est donc identique à appeler :

```
how_many = classmethod(how_many)
```

Notez que la méthode `__init__` est utilisée pour initialiser l'instance `Robot` avec un nom. Dans cette méthode, nous augmentons le compteur `population` de 1, vu que nous avons ajouté un robot. Notez aussi que la valeur de `self.name` est spécifique à chaque objet de par sa nature de variable d'objet.

Souvenez-vous, vous devez vous référer aux variables et méthodes du même objet en utilisant *uniquement* `self`. Cela s'appelle une *référence d'attribut*.

Dans ce programme, nous voyons aussi l'utilisation des *docstrings* pour les classes et les méthodes. Nous pouvons accéder la docstring de la classe à l'exécution en utilisant `Robot.__doc__` et à celles des méthodes avec `Robot.sayHi.__doc__`.

Dans la méthode `die`, nous nous contentons de décrémenter `Robot.population` de 1.

Tous les attributs de classe sont publics. Une exception: si vous nommez un attribut avec le préfixe *double underscore* tel que `__privatevar`, Python utilise le « charcutage de nom » pour la rendre privée dans la pratique.

Ainsi, la convention suivie est que toute variable devant être utilisée uniquement dans la classe ou l'objet doit commencer par un underscore et que tous les autres noms sont publics et peuvent être utilisés par d'autres classes/objets. Rappelez-vous qu'il ne s'agit que d'une convention et que Python ne bloque pas l'accès (à l'exception du préfixe du double underscore).

Note pour les programmeurs C++/Java/C#

Tous les membres de classe (en incluant les variables) sont *publics* et toutes les méthodes sont *virtuelles* en Python.

Héritage

Un des avantages majeurs de la programmation orientée objet est la **re-utilisation** de code et une des manières d'y arriver est par le mécanisme d'**héritage**. On peut voir l'héritage comme le fait d'implémenter une relation **type et sous-type** entre classes.

Supposons que vous voulez écrire un programme qui mémorise les professeurs et les élèves d'une école. Ils ont des caractéristiques en commun, comme le nom, l'âge et l'adresse. Ils ont aussi des caractéristiques spécifiques comme le salaire, les cours et les congés pour les professeurs, et les notes et les bourses pour les élèves.

Vous pouvez créer deux classes indépendantes pour chaque type, et les traiter à part, mais ajouter une nouvelle caractéristique commune impliquera de l'ajouter à chacune de ces classes. Cela devient vite lourd.

Une meilleure approche consiste à créer une classe commune appelée `SchoolMember` et que le professeur et l'élève *héritent* de cette classe, c'est-à-dire qu'ils deviennent un sous-type de ce type (cette classe), et nous pouvons ajouter des caractéristiques spécifiques à ces sous-types.

Cette approche présente de nombreux avantages. Si nous changeons n'importe quelle fonctionnalité dans `SchoolMember`, cela est automatiquement répercuté dans les sous-types. Par exemple, vous pouvez ajouter une nouveau champ carte d'identité pour les professeurs et les élèves en l'ajoutant à la classe `SchoolMember`. Cependant, les changements dans les sous-types ne modifient pas les autres sous-types. Un autre avantage est que vous pouvez faire référence à l'objet professeur ou élève, ce qui peut être utile, par exemple si vous voulez compter le nombre de personnes dans cette école. Cela est appelé le *polymorphisme*, où un sous-type peut être substitué dans n'importe quelle situation dans laquelle un type parent est attendu, c'est-à-dire que l'objet peut être traité comme une instance de la classe parent.

Notez aussi que nous re-utilisons le code de la classe parent et nous n'avons pas besoin de la répéter dans les différentes classes, comme il aurait fallu le faire si nous avions utilisé des classes indépendantes.

La classe `SchoolMember` dans ce cas est vue comme la **classe parente** ou la *superclasse*. Les classes `Teacher` et `Student` sont appelées les **classes dérivées** ou **sous-classes**.

Voyons cela avec un programme (enregistrez sous `oop_subclass.py`):

```
class SchoolMember:
    """Représente n'\importe quel personne de l'\école."""
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('Personne de l\'école initialisée : {}'.format(self.name))

    def tell(self):
        """Donnez-moi des détails."""
        print('Nom :{} Age:{}{}'.format(self.name, self.age), end=" ")

class Teacher(SchoolMember):
    """Représente un professeur."""
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print('Professeur initialisé : {}'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Salaire: {}'.format(self.salary))

class Student(SchoolMember):
    """Représente un étudiant."""
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print('Etudiant initialisé : {}'.format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print('Note : {}'.format(self.marks))

t = Teacher('Mrs. Shrividya', 40, 30000)
s = Student('Swaroop', 25, 75)

print() # imprime une ligne vide

members = [t, s]
for member in members:
    member.tell() # marche à la fois pour les étudiants et les professeurs.
```

Résultat:

```
$ python inherit.py
(Personne de l'école initialisée : Mrs. Shrividya)
(Professeur initialisé : Mrs. Shrividya)
(Personne de l'école initialisée : Swaroop)
(Etudiant initialisé : Swaroop)

Nom :"Mrs. Shrividya" Age:"40" Salaire: "30000"
Nom :"Swaroop" Age:"25" Note : "75"
```

Comment ça marche

Pour utiliser l'héritage, nous spécifions les noms des classes parentes dans un tuple qui suit le nom de classe dans sa définition (par exemple, `class Teacher (SchoolMember)`). Ensuite, notons que la méthode `__init__` de la classe parente est appelée explicitement à l'aide de la variable `self` afin que nous puissions initialiser la partie de la classe parente d'une instance de la sous-classe. Ceci est très important à retenir. Comme nous définissons une méthode `__init__` dans les sous-classes `Teacher` et `Student`, Python n'appelle pas automatiquement le constructeur de la classe parente `SchoolMember`, vous devez l'appeler explicitement vous-même.

En revanche, si nous n'avons pas défini de méthode `__init__` dans une sous-classe, Python appellera automatiquement le constructeur de la classe parente.

Bien que nous puissions traiter les instances de `Teacher` ou `Student` comme une instance de `SchoolMember` et accéder à la méthode `tell` de `SchoolMember` en tapant simplement `Teacher.tell` ou `Student.tell`, nous définissons une autre méthode `tell` dans chaque sous-classe (en utilisant la méthode `tell` de `SchoolMember` pour une partie de celle-ci) afin de l'adapter à cette sous-classe. Comme nous avons fait ça, lorsque nous écrivons `Teacher.tell`, Python utilise la méthode `tell` de cette sous-classe au lieu de celle de la classe parente. Cependant, si nous n'avions pas de méthode `tell` dans la sous-classe, Python utiliserait la méthode `tell` dans la classe parente. Python commence toujours par rechercher d'abord les méthodes du type de sous-classe réel, et s'il ne trouve rien, il examine les méthodes des classes parentes de la sous-classe, une par une, dans l'ordre dans lequel elles sont spécifiées dans le tuple dans la définition de classe (ici nous n'avons qu'une seule classe parent, mais il est possible d'en avoir plusieurs).

Une note de terminologie: si plus d'une classe est présente dans le tuple d'héritage, alors cela s'appelle **l'héritage multiple**.

Le paramètre `end` est utilisé dans la fonction `print` de la méthode `tell()` de la classe parente pour imprimer une ligne et permettre à l'impression suivante de continuer sur la même ligne. C'est une astuce pour que `print` n'imprime pas un symbole `\n` (nouvelle ligne) à la fin de l'impression.

Récapitulatif

Nous avons maintenant exploré les différents aspects des classes et objets, et aussi les différentes terminologies associées. Nous avons également vu les bénéfices et les écueils de la programmation orientée objet. Python est extrêmement orienté objet et comprendre complètement ces concepts vous aidera beaucoup à long terme.

Ensuite, nous apprendrons à gérer des entrées/sorties et à accéder des fichiers en Python.

Entrées Sorties

Il y a des cas où votre programme va interagir avec l'utilisateur. Par exemple, vous voulez faire saisir des valeurs à l'utilisateur et afficher ensuite des résultats. Nous pouvons faire cela en utilisant les fonctions `input()` et `print()` respectivement.

Pour l'affichage, nous pouvons utiliser les différentes méthodes de la classe `str` (chaîne de caractère). Par exemple, vous pouvez utiliser la méthode `rjust` pour justifier à droite une chaîne de caractère avec une largeur donnée. Voyez `help(str)` pour plus de détails.

Une autre type d'entrée/sortie est de traiter des fichiers. La possibilité de créer, lire et écrire des fichiers est essentielle dans de nombreux programmes et nous verrons cela dans ce chapitre.

Entrée de l'utilisateur

Enregistrez sous `io_input.py` :

```
def reverse(text):
    return text[::-1]

def is_palindrome(text):
    return text == reverse(text)

something = input('Entrez votre texte : ')
if (is_palindrome(something)):
    print("Oui, c'est un palindrome")
else:
    print("Non, ce n'est pas un palindrome")
```

Résultat:

```
$ python user_input.py
Entrez votre texte : monsieur
Non, ce n'est pas un palindrome

$ python user_input.py
Entrez votre texte : kayak
Oui, c'est un palindrome

$ python user_input.py
Entrez votre texte : ressasser
Oui, c'est un palindrome
```

Comment ça marche

Nous utilisons le tranchage pour renverser le texte. Nous avons déjà vu comment trancher des séquences en utilisant le code `seq[a:b]` qui va de la position `a` à la position `b`. Nous pouvons aussi fournir un troisième argument qui détermine le *pas* de tranchage. Le pas par défaut est de `1` ainsi il renvoie une partie continue du texte. Donner un pas négatif, c'est-à-dire `-1` va inverser le texte.

La fonction `input()` prend une chaîne de caractères en tant qu'argument et l'affiche à l'utilisateur. Ensuite elle attend une saisie de l'utilisateur terminée par la touche entrée. Quand l'utilisateur a fait cela, la fonction `input()` renvoie le texte saisi.

Nous prenons un texte et le renversons. Si le texte renversé est le même que l'original, alors ce texte est un [palindrome](#).

Devoir maison

Vérifier si un texte est un palindrome devrait ignorer la ponctuation, les espaces et la casse. Par exemple, "Karine alla en Irak" est un palindrome mais la version actuelle de notre programme dit le contraire. Pouvez-vous améliorer le programme ci-dessus pour qu'il reconnaissse ce palindrome?

Si vous avez besoin d'un indice, vous pouvez utiliser un tuple contenant tous les caractères interdits (vous trouverez la liste de *tous* les [caractères de ponctuation ici](#)), puis utilisez le test d'appartenance pour déterminer si un caractère doit être supprimé ou non. Par exemple: `forbidden = ('!', '?', '.', ...)`.

Fichiers

Vous pouvez ouvrir et utiliser des fichiers en lecture ou en écriture en créant un objet de la classe `file` et en utilisant ses méthodes `read`, `readline` ou `write` de manière appropriée pour lire ou écrire dans un fichier. La capacité de lire ou d'écrire dans un fichier dépend de la manière dont il a été ouvert. Enfin, quand vous avez fini de manipuler un fichier, vous pouvez utiliser la méthode `close` pour dire à Python que vous avez fini.

Exemple (enregistrez sous `io_using_file.py`):

```
poem = '''\
La programmation est drôle
Quand le travail est fait
si vous voulez rendre votre travail drôle:
    utiliser Python sera parfait!
'''

# Ouvre le fichier en écriture ('w'riting)
f = open('poem.txt', 'w')
# Écrit du texte dans le fichier
f.write(poem)
# Ferme le fichier
f.close()

# Si le mode n'est pas indiqué,
# le mode lecture est utilisé par défaut ('r'ead)
f = open('poem.txt')
while True:
    line = f.readline()
    # Une longueur de zéro indique fin de fichier
    if len(line) == 0:
        break
    # `line` contient déjà un caractère de
    # retour à la ligne à la fin de chaque ligne
    # car nous lisons un fichier.
    print(line, end=' ')
# Ferme le fichier
f.close()
```

Résultat:

```
$ python using_file.py
La programmation est drôle
Quand le travail est fait
si vous voulez rendre votre travail drôle:
    utiliser Python sera parfait!
```

Comment ça marche

Notez que nous pouvons créer un nouvel objet `file` en utilisant simplement la méthode `open`. Nous ouvrons ce fichier (ou le créons s'il n'existe pas déjà) en utilisant la fonction intégrée `open` et en spécifiant le nom du fichier et le mode dans lequel nous voulons ouvrir le fichier. Le mode peut être un mode de lecture (`r`ead), un mode écriture (`w`rite) ou un mode ajout (`a`ppend). Nous pouvons également spécifier si nous lisons, écrivons ou ajoutons en mode texte (`t`) ou en mode binaire (`b`). Il existe en fait beaucoup plus de modes disponibles et `help(open)` vous donnera les détails. Par défaut, `open()` considère le fichier comme un fichier texte et l'ouvre en mode lecture.

Dans notre exemple, nous ouvrons/créons en premier lieu le fichier en mode texte et écriture, puis nous utilisons la méthode `write` de l'objet `f` de type `file` pour écrire le contenu de notre variable `poem` dans le fichier, puis nous le fermons avec sa méthode `close`.

Ensuite, nous rouvrons le même fichier pour le lire. Nous n'avons pas besoin de spécifier un mode car « lire un fichier texte » est le mode par défaut. Nous lisons chaque ligne du fichier en utilisant la méthode `readline` dans une boucle. Cette méthode retourne une ligne complète incluant le caractère de retour à la ligne à la fin de la ligne. Quand une chaîne *vide* nous est renvoyée, cela signifie que nous avons atteint la fin du fichier et nous « sortons » de la boucle avec l'instruction `break`.

Enfin, nous fermons le fichier avec sa méthode `close`.

Nous pouvons voir dans le résultat, que ce programme a bien créé le fichier `poem.txt`, y a écrit le poème, puis l'a réouvert et lu à nouveau.

Pickle

Python fournit un module standard appelé `pickle` qui permet d'enregistrer *n'importe quel* objet Python dans un fichier pour y accéder plus tard. On appelle cela enregistrer l'objet *de manière persistante*.

Exemple (enregistrez sous `io_pickle.py`):

```
import pickle

# Le nom du fichier dans lequel nous stockerons l'objet
shoplistfile = 'shoplist.data'
# Notre liste de courses
shoplist = ['pomme', 'mangue', 'carotte']

# Ouvre le fichier en écriture
f = open(shoplistfile, 'wb')
# Écrit l'objet dans le fichier
pickle.dump(shoplist, f)
f.close()

# Détruit la variable shoplist
del shoplist

# Ouvre le fichier
f = open(shoplistfile, 'rb')
# Charge l'objet à partir du fichier
storedlist = pickle.load(f)
print(storedlist)
f.close()
```

Résultat:

```
$ python pickling.py
['pomme', 'mangue', 'carotte']
```

Comment ça marche

Pour enregistrer un objet dans un fichier, nous devons d'abord ouvrir (`open`) le fichier avec le mode écriture binaire (`wb` pour *write binary*) et ensuite appeler la fonction `dump` du module `pickle`. Cette procédure est appelé *pickling*.

Ensuite, nous récupérons l'objet en utilisant la fonction `load` du module `pickle` qui renvoie l'objet. Ce procédé est appelé *unpickling*.

Unicode

Jusqu'à présent, lorsque nous écrivions et utilisions des chaînes de caractères, ou lisions et écrivions dans un fichier, nous n'utilisions que des caractères non accentués de l'alphabet latin. Les caractères accentués, non accentués ou d'autres alphabets peuvent être représentés en Unicode (voir les articles à la fin de cette section pour plus d'informations), que Python 3 utilise par défaut pour les chaînes de caractères (tout le texte que nous avons écrit en utilisant simple, double ou triple guillemets).

NOTE: Si vous utilisez Python 2 et que vous voulez pouvoir lire et écrire des langues non anglaises, vous devez utiliser le type `unicode`, en préfixant vos chaînes par le caractère `u`. Par exemple: `u"hello world"`

```
>>> "hello world"
'hello world'
>>> type("hello world")
<class 'str'>
>>> u"hello world"
'hello world'
>>> type(u"hello world")
<class 'str'>
```

Lorsque des données sont envoyées sur Internet, nous devons les envoyer en octets... pour que votre ordinateur puisse comprendre facilement. Les règles de conversion d'Unicode (utilisé par Python lorsqu'il stocke une chaîne) en octets sont appelées encodage. Un encodage populaire à utiliser est UTF-8. Nous pouvons lire et écrire en UTF-8 en utilisant un simple paramètre nommé dans notre fonction `open`.

```
# encoding=utf-8
import io

f = io.open("abc.txt", "wt", encoding="utf-8")
f.write(u"こんにちは世界")
f.close()

text = io.open("abc.txt", encoding="utf-8").read()
print(text)
```

Comment ça marche

Nous utilisons `io.open` puis le paramètre `encoding` dans le premier `open` pour encoder le message, puis de nouveau dans le deuxième lors du décodage du message. Notez que le paramètre `encoding` ne fait sens que lorsque nous manipulons des fichiers en mode texte.

Chaque fois que nous écrivons un programme qui utilise des constantes littérales Unicode (en plaçant un "u" avant la chaîne de caractères) comme nous l'avons fait ci-dessus, nous devons nous assurer que Python lui-même est informé que notre programme est en UTF-8, et nous devons mettre le commentaire `#encoding=utf-8` en haut de notre programme.

Vous devriez en apprendre plus sur ce sujet en lisant:

- [Le minimum absolu que tout programmeur doit absolument connaître sur Unicode et les jeux de caractères](#)
- [Guide sur Unicode avec Python](#)
- [Discussion pragmatique sur Unicode par Nat Batchelder](#)

Récapitulatif

Nous avons vu différents types d'entrées sorties, comment gérer des fichiers, l'utilisation du module pickle et mentionné Unicode.

Nous allons maintenant travailler sur le concept d'exception.

Exceptions

Les exceptions se produisent quand une situation *exceptionnelle* arrive dans votre programme. Par exemple, que se passe-t-il quand vous voulez lire un fichier et que ce fichier n'existe pas ? Ou quand vous le détruissez par erreur pendant l'exécution du programme ? De telles situations sont gérées en utilisant des **exceptions**.

De la même manière, que se passe-t-il si votre programme contient des instructions incorrectes ? Cela est géré par Python qui **lève** la main et vous dit qu'il y a une **erreur**.

Erreurs

Considérez un simple appel à la fonction `print`. Que se passe-t-il si nous faisons une faute de frappe et écrivons `Print` à la place de `print` ? Notez la lettre majuscule. Dans ce cas, Python *lève* une erreur de syntaxe.

```
>>> Print("Hello World")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print("Hello World")
Hello World
```

Observez qu'une `NameError` est levée et que l'emplacement où l'erreur a été détectée est affiché. C'est le travail du **gestionnaire d'erreur** de cette erreur.

Exceptions

Nous allons **essayer** de lire la saisie de l'utilisateur. Entrez la première ligne ci-dessous et appuyez sur la touche `Entrée`. Lorsque votre ordinateur vous invite à saisir, faites à la place `[ctrl-d]` sur Mac ou `[ctrl-z]` sous Windows pour voir ce qui se passe. (Si vous utilisez Windows et qu'aucune des deux options ne fonctionne, essayez `[ctrl-c]` dans l'invite de commande pour générer une erreur `KeyboardInterrupt` à la place).

```
>>> s = input('Saisissez quelque chose --> ')
Saisissez quelque chose --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python lève une erreur appelée `EOFError` qui signifie en gros qu'il a trouvé un symbole de fin de fichier (*End Of File*) (représenté par `[ctrl-d]`) alors qu'il ne s'attendait pas à le voir.

Gérer les exceptions

We can handle exceptions using the `try..except` statement. We basically put our usual statements within the try-block and put all our error handlers in the except-block.

Exemple (enregistrez sous `exceptions_handle.py`):

```

try:
    text = input('Saisissez quelque chose --> ')
except EOFError:
    print('Pourquoi m\'avez-vous envoyé une fin de fichier ?')
except KeyboardInterrupt:
    print('Vous avez annulé l\'opération.')
else:
    print('Vous avez saisi {}'.format(text))

```

Résultat:

```

# Tapez ctrl + d
$ python exceptions_handle.py
Saisissez quelque chose -->
Pourquoi m'avez-vous envoyé une fin de fichier ?

# Tapez ctrl + c
$ python exceptions_handle.py
Saisissez quelque chose --> ^CVous avez annulé l'opération.

$ python exceptions_handle.py
Saisissez quelque chose --> Pas d'exception
Vous avez saisi Pas d'exception

```

Comment ça marche

Nous mettons toutes les instructions qui peuvent lever des exceptions/erreurs à l'intérieur du bloc `try` et gérons les erreurs/exceptions prévues dans la clause (ou bloc) `except`. La clause `except` peut gérer une seule erreur ou exception spécifiée, ou une liste d'erreurs ou d'exceptions entre parenthèses. Si le nom des erreurs ou exceptions à gérer n'est pas spécifié, le bloc gérera *toutes* les erreurs et exceptions.

Notez qu'il faut avoir au moins une clause `except` associée à chaque clause `try`. Sinon, quel serait l'intérêt d'avoir un bloc `try` ?

Si une erreur ou exception n'est pas gérée, alors le gestionnaire par default de Python est appelé, il arrête l'exécution du programme et affiche un message d'erreur. Nous avons déjà testé cela plus haut.

Vous pouvez aussi avoir une clause `else` associée à un bloc `try..except`. La clause `else` est exécutée lorsqu'on ne rencontre pas d'exception.

Dans l'exemple suivant, nous verrons comment récupérer l'objet d'exception afin d'obtenir des informations supplémentaires.

Lever des exceptions

Vous pouvez *lever* des exceptions avec l'instruction `raise` en fournissant le nom de l'erreur/exception et l'objet `exception` qui sera lancé.

L'erreur ou exception que vous levez doit être une classe qui doit être dérivée directement ou indirectement de la classe `Exception`.

Exemple (enregistrez sous `exceptions_raise.py`):

```

class ShortInputException(Exception):
    '''Une exception définie par l'utilisateur.'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast

    try:
        text = input('Saisissez quelque chose --> ')
        if len(text) < 3:
            raise ShortInputException(len(text), 3)
        # On pourrait continuer de manière normale ici
    except EOFError:
        print('Pourquoi m\'avez-vous envoyé une fin de fichier ?')
    except ShortInputException as ex:
        print(('ShortInputException: La saisie avait une longueur de ' +
              '{0}, et on attendait au moins {1}').format(ex.length, ex.atleast))
    else:
        print('Pas d\'exception levée.')

```

Résultat:

```

$ python exceptions_raise.py
Saisissez quelque chose --> a
ShortInputException: La saisie avait une longueur de 1, et on attendait au moins 3

$ python exceptions_raise.py
Saisissez quelque chose --> abc
Pas d'exception levée.

```

Comment ça marche

Nous créons ici notre propre type d'exception. Ce nouveau type d'exception est appelé `ShortInputException`. Il possède deux champs: `length` qui est la longueur saisie, et `atleast` qui est la longueur minimum attendue par le programme.

Dans la clause `except`, nous indiquons la classe de l'erreur qui sera stockée en tant que `ex` (`as ex`), le nom de variable qui va contenir l'objet d'erreur/exception correspondant. Cela est comparable aux paramètres et arguments dans un appel de fonction. A l'intérieur de cet clause `except`, nous utilisons les champs `length` et `atleast` de l'objet exception pour afficher le message approprié à l'utilisateur.

Try ... Finally

Supposons, vous lisez un fichier dans votre programme. Comment être certain que l'objet fichier est fermé proprement, indépendamment du fait qu'une exception soit levée ou non dans le code qui le manipule ? Cela se fait à l'aide d'un bloc `finally`.

Enregistrez ce programme sous `exceptions_finally.py` :

```

import sys
import time

f = None
try:
    f = open("poem.txt")
    # Notre idiomme habituel pour lire un fichier
    while True:
        line = f.readline()
        if len(line) == 0:
            break
        print(line, end='')
        sys.stdout.flush()
        print("Tapez ctrl+c maintenant")
        # Pour être certain que cela dure un peu
        time.sleep(2)
except IOError:
    print("Le fichier poem.txt n'existe pas")
except KeyboardInterrupt:
    print("!! Vous avez annulé la lecture du fichier")
finally:
    if f:
        f.close()
    print("(Nettoyage : fermeture du fichier)")

```

Résultat:

```

$ python exceptions_finally.py
Programming is fun
Press ctrl+c now
^C!! You cancelled the reading from the file.
(Cleaning up: Closed the file)

```

Comment ça marche

Nous faisons notre lecture de fichier habituelle, mais nous avons introduit de manière arbitraire une pause de 2 secondes après l'impression de chaque ligne à l'aide de la fonction `time.sleep` afin que le programme s'exécute lentement (Python est très rapide par nature). Pendant que le programme est en cours d'exécution, appuyez sur `ctrl + c` pour l'interrompre.

Notez que l'exception `KeyboardInterrupt` est levée et que le programme se ferme. Cependant, avant sa fermeture, la clause `finally` est exécutée et l'objet fichier est fermé.

Notez également qu'une variable affectée d'une valeur de 0 ou `None` ou d'une variable qui contient séquence ou une collection vide est considérée comme `False` par Python. C'est pourquoi nous pouvons utiliser `if: f` dans le code ci-dessus.

Notez également que nous utilisons `sys.stdout.flush()` après `print` pour nous garantir que le message soit immédiatement affiché à l'écran.

L'instruction `with`

Acquérir une ressource dans le bloc `try` et la relâcher dans le bloc `finally` arrive très fréquemment. Par conséquent, il y a aussi une instruction `with` qui permet de faire cela de manière plus lisible :

Enregistrez sous `exceptions_using_with.py` :

```

with open("poem.txt") as f:
    for line in f:
        print(line, end='')

```

Comment ça marche

Le résultat devrait être le même que dans l'exemple précédent. La différence est que nous utilisons la fonction `open` avec l'instruction `with` et nous laissons la fermeture du fichier se faire automatiquement avec `with open`.

Ce qui se passe en arrière-plan est qu'un protocole est utilisé par l'instruction `with`. Il recherche l'objet retourné par l'instruction `open`, appelons-le `thefile` dans ce cas.

Il appelle *toujours* la fonction `thefile.__enter__` avant de commencer le bloc de code en dessous et appelle *toujours* `thefile.__exit__` après avoir terminé le bloc de code.

Donc la méthode `__exit__` prendra en charge le code que nous aurions écrit dans un bloc `finally`. C'est ce qui nous aide à éviter d'utiliser des instructions `try..finally` de manière répétitive.

Une discussion plus approfondie sur ce sujet est au-delà de l'objectif de ce livre, consultez [PEP 343](#) pour une explication détaillée.

Récapitulatif

Nous avons vu comment utiliser les instructions `try..except` et `try..finally`. Nous avons vu comment créer nos propres types d'exceptions et comment lever des exceptions.

Ensuite, nous allons explorer la bibliothèque standard de Python.

Bibliothèque standard

La bibliothèque standard de Python contient de très nombreux modules et fait partie de l'installation standard de Python. Il est important de se familiariser avec la bibliothèque standard de Python, car de nombreux problèmes peuvent être résolu rapidement si vous savez ce qu'elle propose.

Nous étudierons quelques modules couramment utilisés dans cette librairie. Vous pouvez trouver tous les détails de tous les modules de la bibliothèque standard dans la section [bibliothèque standard](#) de la documentation fournie par votre installation de Python.

Explorons quelques modules utiles.

ATTENTION: Si vous trouvez les sujets de ce chapitre trop difficiles, je vous conseille de passer au suivant. Cependant, je vous recommande de revenir ensuite à ce chapitre quand vous serez plus à l'aise en programmation Python.

Le module sys

Le module `sys` contient des fonctionnalités spécifiques au système utilisé. Nous avons déjà vu que la liste `sys.argv` contient les arguments passés en ligne de commande.

Supposons que nous voulions tester la version de Python, pour vérifier que nous utilisons une version supérieure ou égale à la version 3. Le module `sys` nous offre cette fonctionnalité.

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=6, micro=0, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

Comment ça marche

Le module `sys` possède un tuple `version_info` qui nous informe sur la version. La première entrée est la version majeure. Nous pouvons accéder à cette information et l'utiliser.

Le module logging

Et si vous voulez stocker quelque part des messages de débogage ou d'autres messages importants, afin de vérifier que votre programme s'est comporté comme vous l'espérez ? Comment « stocker quelque part » ces messages ? Cela peut être réalisé avec le module `logging`.

Enregistrez sous `stdlib_logging.py` :

```

import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Écriture du journal dans", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
    filemode='w',
)

logging.debug("Début du programme")
logging.info("En train de faire quelque chose")
logging.warning("Terminé")

```

Résultat:

```

$ python stdlib_logging.py
Écriture du journal dans /Users/swa/test.log

$ cat /Users/swa/test.log
2014-03-29 09:27:36,660 : DEBUG : Début du programme
2014-03-29 09:27:36,660 : INFO : En train de faire quelque chose
2014-03-29 09:27:36,660 : WARNING : Terminé

```

La commande `cat` est utilisée en la ligne de commande pour lire le fichier `'test.log'`. Si la commande `cat` n'est pas disponible, vous pouvez ouvrir le fichier `test.log` dans un éditeur de texte.

Comment ça marche

Nous utilisons trois modules de la librairie standard: le module `os` pour interagir avec le système d'exploitation, le module `platform` pour accéder à des informations sur la plate-forme, c'est-à-dire le système d'exploitation et le module `logging` pour écrire des informations dans un journal.

D'abord nous vérifions quel système d'exploitation nous utilisons en regardant la chaîne de caractères renvoyée par `platform.platform()` (pour plus d'informations, voyez `import platform; help(platform)`). Si c'est Windows, nous en déduisons le disque et le répertoire par défaut et le nom du fichier où nous stockons l'information. En mettant bout à bout ces trois informations, nous obtenons l'emplacement complet du fichier. Pour les autres plate-formes, nous avons juste besoin de connaître le répertoire par défaut de l'utilisateur et nous avons l'emplacement complet du fichier.

Nous utilisons la fonction `os.path.join()` pour concaténer ces trois parties de l'emplacement. Nous utilisons une fonction spéciale plutôt qu'un simple ajout de chaînes de caractères, parce que cette fonction va s'assurer que l'emplacement complet correspond au format attendu par le système d'exploitation. Note: la méthode `join()` que nous utilisons ici fait partie du module `os`, ce n'est pas la même que de la méthode `join()` disponible sur les chaînes de caractères que nous avons utilisé ailleurs dans ce livre.

Nous configurons le module `logging` pour écrire nos messages avec un format particulier dans le fichier que nous spécifions.

Enfin, nous écrivons les messages prévus pour le débogage, l'information, les avertissements ou les messages critiques. Après l'exécution du programme, nous pouvons regarder ce fichier et voir ce qu'il s'est passé, bien qu'aucune information n'ait été affichée à l'utilisateur pendant l'exécution du programme.

Série « Le module de la semaine »

Il reste beaucoup de choses à découvrir dans la bibliothèque standard, comme le [débogage](#), la [la gestion des options en ligne de commande](#), les [expressions régulières](#) et bien plus.

La meilleure façon de continuer à explorer la bibliothèque standard est de lire l'excelente série de Doug Hellmann [Le module Python de la semaine](#) (également disponible en [format papier](#)) et de lire la [documentation de Python](#).

Récapitulatif

Nous avons exploré quelques fonctionnalités des nombreux modules de la Librairie Standard Python. Il est fortement recommandé de naviguer dans la [documentation de la bibliothèque standard de Python](#) pour se faire une idée de tous les modules disponibles.

Ensuite, nous allons voir différents aspects de Python afin que notre voyage dans Python soit plus *complet*.

Pour aller plus loin

Nous avons couvert les principaux aspects de Python que vous utiliserez. Dans ce chapitre, nous verrons quelques autres points qui complèteront notre connaissance de Python.

Retourner des tuples

Vous avez déjà voulu retourner deux valeurs depuis une fonction ? C'est possible. Il vous suffit d'utiliser un tuple.

```
>>> def get_error_details():
...     return (2, 'details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'details'
```

Notez que l'usage de `a, b = <une expression>` interprète le résultat de l'expression en tant que tuple avec deux valeurs.

De cette manière, la méthode la plus rapide pour échanger deux variables en Python est :

```
>>> a = 5; b = 8
>>> a, b
(5, 8)
>>> a, b = b, a
>>> a, b
(8, 5)
```

Méthodes spéciales

Certaines méthodes comme `__init__` et `__del__` ont une signification spéciale dans les classes.

Les méthodes spéciales sont utilisées pour imiter certains comportements des types intégrés. Par exemple, si vous voulez utiliser l'opération d'indexation `x[key]` pour votre classe (comme vous le faites avec les listes et les tuples), alors il vous suffit d'implémenter la méthode `__getitem__()` et c'est fait. Si vous y réfléchissez, c'est ce que fait Python pour la classe `list` elle-même !

Quelques méthodes spéciales utiles sont listées dans le tableau suivant. Si vous voulez tout savoir sur les méthodes spéciales, [référez-vous à la documentation](#).

- `__init__(self, ...)`
 - Cette méthode est appelée juste avant que l'objet nouvellement créé ne soit retourné pour être utilisé.
- `__del__(self)`
 - Appelée juste avant que l'objet ne soit détruit par Python (ce moment est imprévisible, donc évitez de vous en servir)
- `__str__(self)`
 - Appelée quand nous appelons la fonction `print` ou quand `str()` est utilisé.
- `__lt__(self, other)`
 - Appelée quand l'opérateur *inférieur à* (`<`) est utilisé. De la même manière, il y a des méthodes spéciales pour tous les opérateurs (`+`, `>`, etc...)
- `__getitem__(self, key)`
 - Appelée quand l'opération d'indexation `x[clef]` est utilisée.
- `__len__(self)`

- Appelée quand la fonction intégrée `len()` est utilisée sur un objet séquence.

Blocs d'instructions à une ligne

Nous avons vu que chaque bloc d'instructions se démarque des autres par son niveau d'indentation. Il y a une mise en garde. Si votre bloc d'instructions ne contient qu'une instruction, vous pouvez l'écrire sur la même ligne que, disons, une instruction de condition ou de boucle. L'exemple suivant devrait clarifier cela :

```
>>> flag = True
>>> if flag: print('Oui')
...
Oui
```

Notez que l'unique instruction est utilisée sur place et pas en tant que bloc séparé. Bien qu'il soit possible d'utiliser cela pour *raccourcir* votre programme, je recommande fortement d'éviter cette méthode, à part pour rechercher des erreurs, principalement parce qu'il sera beaucoup plus facile d'ajouter une instruction supplémentaire si vous utilisez une indentation correcte.

Fonctions lambda

Une instruction `lambda` est utilisée pour créer de nouveaux objets de type fonction. En résumé, une `lambda` prend un paramètre suivi d'une seule expression. La valeur de cette expression est retournée par la nouvelle fonction.

Exemple (enregistrez sous `more_lambda.py`):

```
points = [{"x": 2, "y": 3},
          {"x": 4, "y": 1}]
points.sort(key=lambda i: i['y'])
print(points)
```

Résultat:

```
$ python more_lambda.py
[{"y": 1, "x": 4}, {"y": 3, "x": 2}]
```

Comment ça marche

Notez que la méthode `sort` d'une liste peut prendre un paramètre `key` qui détermine le mode de tri de la liste (en général, nous ne trions que par ordre croissant ou décroissant). Dans notre cas, nous voulons faire un tri personnalisé, pour cela nous devons écrire une fonction. Au lieu d'écrire un bloc `def` distinct pour une fonction qui ne sera utilisée qu'à cet endroit, nous utilisons une expression `lambda` pour créer une nouvelle fonction.

Compréhension de liste

Les compréhension de liste sont utilisées pour créer une nouvelle liste à partir d'une liste existante. Supposons, vous avez une liste de nombres et vous voulez obtenir la liste correspondante dont chaque valeur est multipliée par 2, mais uniquement nombre supérieurs à 2. Les compréhension de liste sont idéales pour ces situations.

Exemple (enregistrez sous `more_list_comprehension.py`):

```
listone = [2, 3, 4]
listtwo = [2 * i for i in listone if i > 2]
print(listtwo)
```

Résultat:

```
$ python more_list_comprehension.py  
[6, 8]
```

Comment ça marche

Ici, nous créons une nouvelle liste en indiquant la manipulation à effectuer (`2 * i`) et la condition à satisfaire (`if i > 2`). Notez que la liste originale n'est pas modifiée.

L'avantage des list comprehensions est que cela réduit la quantité de code passe-partout nécessaire quand on utilise une boucle pour traiter chaque élément d'une liste et le stocker dans une nouvelle liste.

Recevoir des tuples et des dictionnaires dans des fonctions

Il existe une façon spéciale de recevoir des paramètres pour une fonction en tant que tuple ou dictionnaire en utilisant respectivement les préfixes `*` ou `**`. Cela est utile pour créer des fonctions prenant un nombre variable de paramètres.

```
>>> def powersum(power, *args):  
...     '''Renvoie la somme de chaque paramètre élevé à la puissance indiquée.'''  
...     total = 0  
...     for i in args:  
...         total += pow(i, power)  
...     return total  
...  
>>> powersum(2, 3, 4)  
25  
>>> powersum(2, 10)  
100
```

Comme nous avons un préfixe `*` sur la variable `args`, tous les arguments supplémentaires passés à la fonction sont stockés dans le tuple `args`. Si un préfixe `**` avait été utilisé, les paramètres supplémentaires auraient été vus comme des paires clé/valeur d'un dictionnaire.

L'instruction assert

L'instruction `assert` est utilisée pour vérifier que quelque chose est vrai. Par exemple, si vous êtes persuadé qu'une liste contient au moins un élément, et que vous voulez lever une erreur dans le cas contraire, alors l'instruction `assert` est idéale. Quand une vérification avec `assert` échoue, une `AssertionError` est levée.

La méthode `pop()` retire et retourne le dernier élément d'une liste.

```
>>> mylist = ['item']  
>>> assert len(mylist) >= 1  
>>> mylist.pop()  
'item'  
>>> assert len(mylist) >= 1  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AssertionError
```

L'instruction `assert` doit être utilisée à bon escient. La plupart du temps, il vaut mieux gérer des exceptions puis, soit gérer le problème, soit afficher un message d'erreur à l'utilisateur et quitter le programme.

Décorateurs

Les décorateurs sont un raccourci pour appliquer des fonctions autour d'une autre fonction. Ceci est utile pour "envelopper" une fonctionnalité avec le même code, encore et encore. Par exemple, j'ai créé un décorateur `retry` que je ne peux appliquer à n'importe quelle fonction. Si une exception est levée au cours d'une exécution, l'appel à la fonction sera tentée jusqu'à 5 fois avec un délai entre chaque nouvelle tentative. Ceci est particulièrement utile lorsque vous essayez de réaliser un appel réseau à un ordinateur distant:

```
from time import sleep
from functools import wraps
import logging
logging.basicConfig()
log = logging.getLogger("retry")

def retry(f):
    @wraps(f)
    def wrapper_function(*args, **kwargs):
        MAX_ATTEMPTS = 5
        for attempt in range(1, MAX_ATTEMPTS + 1):
            try:
                return f(*args, **kwargs)
            except Exception:
                log.exception("La tentative %s/%s a échoué : %s",
                              attempt,
                              MAX_ATTEMPTS,
                              (args, kwargs))
                sleep(10 * attempt)
        log.critical("Les %s tentatives ont échoué : %s",
                     MAX_ATTEMPTS,
                     (args, kwargs))
    return wrapper_function

counter = 0

@retry
def save_to_database(arg):
    print("Écrit dans une base de données, réalise un appel réseau ou autres...")
    print("Si une exception est levé, une nouvelle tentative sera automatiquement lancé.")
    global counter
    counter += 1
    # Ceci lancera une exception au premier appel
    # mais marchera au deuxième (lors d'une nouvelle tentative)
    if counter < 2:
        raise ValueError(arg)

if __name__ == '__main__':
    save_to_database("Une mauvaise valeur")
```

Résultat:

```
$ python more_decorator.py
Écrit dans une base de données, réalise un appel réseau ou autres...
Si une exception est levé, une nouvelle tentative sera automatiquement lancé.
ERROR:retry:La tentative 1/5 a échoué : (('Une mauvaise valeur',), {})
Traceback (most recent call last):
  File "more_decorator.py", line 14, in wrapper_function
    return f(*args, **kwargs)
  File "more_decorator.py", line 39, in save_to_database
    raise ValueError(arg)
ValueError: Une mauvaise valeur
Écrit dans une base de données, réalise un appel réseau ou autres...
Si une exception est levé, une nouvelle tentative sera automatiquement lancé.
```

Comment ça marche

Consultez:

- [Vidéo: Comprendre facilement les décorateurs Python](#)
- <http://www.ibm.com/developerworks/linux/library/l-cpdecor.html>

- <http://toumorokoshi.github.io/dry-principles-through-python-decorators.html>

Differences entre Python 2 et Python 3

Consultez:

- [Bibliothèque « Six »](#)
- [Porter du code en Python 3 Redux par Armin](#)
- [Expérience avec Python 3 par PyDanny](#)
- [Guide officiel de Django pour porter du code en Python 3](#)
- [Discussion sur: quels sont les avantages de python 3.x?](#)

Récapitulatif

Nous avons vu quelques autres fonctionnalités de Python dans ce chapitre, et pourtant, nous n'avons pas encore couvert toutes les fonctionnalités de Python. Cependant, nous avons vu maintenant l'essentiel de ce que vous utiliserez en pratique. C'est suffisant pour vous permettre de démarrer avec les programmes que vous allez créer.

Il nous reste à discuter de comment vous y prendre pour approfondir vos connaissances en Python.

Et ensuite

Si vous avez lu ce livre attentivement jusqu'ici et écrit de nombreux programmes, alors vous êtes à l'aise avec Python. Vous avez sans doute créé quelques programmes Python pour faire des tentatives et tester vos capacités en Python. Si vous ne l'avez pas encore fait, vous devriez. La question à se poser maintenant est « Et ensuite ? ».

Je vous suggère de vous attaquer à ce problème :

Créer votre propre *carnet d'adresses* en ligne de commande avec lequel vous pourrez consulter, ajouter, modifier, détruire ou rechercher parmi vos contacts comme vos amis, les membres de votre famille et vos collègues, et retrouver des informations comme l'email et/ou le téléphone de chacun. Les détails doivent être enregistrés entre chaque utilisation du programme.

Cela est assez facile si vous pensez à tout ce que l'on a vu précédemment. Si vous voulez des indications sur la manière de faire, voici un indice: Créez une classe qui représente les informations relatives à une personne. Utilisez un dictionnaire pour enregistrer les objets d'une personne avec le nom en tant que clé. Utilisez le module pickle pour enregistrer les objets de manière persistente sur votre disque dur. Utilisez les méthodes fournies par le dictionnaire pour ajouter, détruire et modifier les personnes.

Quand vous serez capable de faire cela, vous pourrez dire que vous êtes un programmeur Python. Maintenant, tout de suite [envoyez-moi un email](#) pour me remercier pour ce super livre ;). Cette étape est bien-sûr optionnelle, mais recommandée. Aussi, pensez à [acheter une copie physique du livre](#) pour contribuer à son développement continu.

Si vous avez trouvé ce programme facile, en voici un autre :

Implémentez la [commande remplacer](#). Cette commande remplacera une chaîne de caractères par une autre dans la liste de fichiers fournie.

La commande remplacer peut être simple ou compliquée comme vous le voulez, de la simple substitution de chaîne de caractères à la recherche de motifs (expressions régulières).

Projets suivants

Si vous avez trouvé les programmes ci-dessus faciles à créer, consultez cette liste complète de projets et essayez d'écrire vos propres programmes: <https://github.com/thekarangoel/Projects#numbers> (la même liste est également disponible à l'adresse [Mega liste de projets par Martyr2](#)).

Voyez également:

- [Exercices pour programmeurs: 57 défis pour développer vos compétences en programmation](#)
- [Projets Python niveau intermédiaire.](#)

Example Code

La meilleure façon d'apprendre un langage de programmation consiste à écrire et à lire beaucoup de code:

- Le [livre de recettes Python](#) est une collection extrêmement précieuse de recettes ou d'astuces sur la façon de résoudre certains types de problèmes avec Python. C'est une lecture incontournable pour tous les utilisateurs Python.
- La série [Le module Python de la semaine](#) est un autre excellent guide à lire absolument sur la [bibliothèque standard](#).

Conseils

- [The Hitchhiker's Guide to Python!](#)
- [The Elements of Python Style](#)
- [Python Big Picture](#)
- [« Writing Idiomatic Python » ebook \(paid\)](#)

Vidéos

- [Full Stack Web Development with Flask](#)
- [PyVideo](#)

Questions et réponses

- [Official Python Dos and Don'ts](#)
- [Official Python FAQ](#)
- [Norvig's list of Infrequently Asked Questions](#)
- [Python Interview Q & A](#)
- [StackOverflow questions tagged with python](#)

Tutoriels

- [Hidden features of Python](#)
- [What's the one code snippet/python trick/etc did you wish you knew when you learned python?](#)
- [Awaretek's comprehensive list of Python tutorials](#)

Discussion

Si vous êtes coincé avec un problème Python et que vous ne savez pas à qui demander, la [liste de tuteurs python](#) est le meilleur endroit pour vous renseigner.

Assurez-vous de bien faire vos devoirs en essayant d'abord de résoudre le problème vous-même et [posez des questions intelligentes](#).

Actualités

Si vous voulez être au courant des dernières nouveautés dans le monde Python, alors suivez le blog officiel [Python Planet](#).

Installation de bibliothèques

Il y a de très nombreuses bibliothèques open source dans le [Python Package Index](#) que vous pouvez utiliser dans vos programmes.

Pour les installer et les utiliser, utilisez [pip](#).

Création de sites web

Apprenez à utiliser [Flask](#) pour créer vos propres sites web. Quelques resources pour démarrer :

- [Flask Official Quickstart](#)
- [The Flask Mega-Tutorial](#)
- [Example Flask Projects](#)

Logiciels avec interface graphique

Supposons que vous vouliez créer votre propre programme avec une interface graphique en utilisant Python. Cela peut être fait avec une bibliothèque graphique et les `bindings` Python. Des `bindings` permettent d'écrire des programmes en Python qui utilisent des bibliothèques écrites en C, C++ ou d'autres langages.

Il existe un large choix de bibliothèques graphiques avec Python:

- Kivy
 - <http://kivy.org>
- PyGTK
 - Il s'agit des bindings Python pour le toolkit GTK+ toolkit qui est la fondation sur laquelle GNOME est construit. GTK+ a de nombreuses bizarries, mais une fois que vous y êtes habitué, vous pouvez créer rapidement des applications graphiques. Le Glade Graphical Interface Designer est indispensable. La documentation est améliorable. GTK+ fonctionne bien sous Linux mais son portage sous Windows est incomplet. Vous pouvez créer des logiciels libres ou propriétaires avec GTK+. Pour commencer, lisez le [tutoriel PyGTK](#).
- PyQt
 - Il s'agit des bindings Python pour le toolkit Qt, qui est la fondation sur laquelle KDE est construit. Qt est très facile à utiliser et très puissant, en particulier grâce à Qt Designer et l'excellente documentation Qt. PyQt est gratuit si vous voulez créer un programme open source (sous licence GPL) et vous devez payer si vous voulez créer un programme propriétaire dont le code est fermé. A partir de Qt 4.5 vous pouvez aussi créer du code non-GPL. Pour commencer, renseignez vous sur [PySide](#).
- wxPython
 - Il s'agit des bindings Python pour le toolkit wxWidgets. wxPython a une courbe d'apprentissage associée. Cependant, il est très portable, et fonctionne sous Linux, Windows, Mac et même des plate-formes embarquées. Il y a de nombreux IDEs disponibles pour wxPython, dont des GUI designers comme [SPE \(Stani's Python Editor\)](#) et [wxGlade](#) GUI builder. Vous pouvez créer des logiciels libres ou propriétaires avec wxPython. Pour commencer, lisez le [tutoriel wxPython](#).

Récapitulatif sur les outils sur les interfaces graphiques

Pour plus de choix, voyez [la page GUI Programming sur le site officiel Python](#).

Il n'y a, hélas, pas d'outil graphique standard pour Python. Je vous suggère de choisir l'un des outils pré-cités en fonction de vos besoins. Le premier critère est si vous êtes d'accord pour payer pour l'un de ces outils. Le deuxième critère est si vous voulez programmer sous Windows ou sous Mac et Linux ou sous tous. Le troisième critère, si vous choisissez Linux, est votre préférence utilisateur envers KDE ou GNOME.

Pour une analyse plus détaillée, voyez la page 26 de [« The Python Papers, Volume 3, Issue 1 » \(PDF\)](#).

Autres implémentations

Il y a en général deux parties dans un langage de programmation - le langage et le logiciel. Un langage est *comment* vous écrivez quelque chose. Le logiciel est *ce qui* fait réellement tourner notre programme.

Nous avons utilisé le logiciel *CPython* pour exécuter nos programmes. On l'appelle CPython parce qu'il est écrit en langage C et que c'est l'interpréteur classique de Python.

Il existe également d'autres logiciels pour exécuter vos programmes Python :

- [Jython](#)
 - Une implémentation de Python qui tourne sur la plate-forme Java. Cela signifie que vous pouvez utiliser des classes et bibliothèques Java à partir du langage Python et vice-versa.
- [IronPython](#)
 - Une implémentation de Python qui tourne sur la plate-forme .NET. Cela signifie que vous pouvez utiliser les bibliothèques et classes .NET à partir du langage Python et vice-versa.
- [PyPy](#)
 - Une implémentation Python écrite en Python! C'est un projet de recherche pour améliorer l'interpréteur et le rendre plus rapide, dans ce cas l'interpréteur lui-même est écrit dans un langage dynamique (au contraire de langages statiques comme C, Java ou C# dans les trois implémentations au-dessus)

Il en existe d'autres comme [CLPython](#), une implémentation Python écrite en Common Lisp et [Brython](#) qui est une implémentation en Javascript, ce qui veut dire que vous pourriez utiliser Python (au lieu de JavaScript) pour écrire vos logiciels pour navigateur.

Chacune des ces implémentations est utile dans le domaine dans lequel elle est spécialisée.

Programmation fonctionnelle (pour les lecteurs avancés)

Lorsque vous commencez à écrire des programmes plus volumineux, vous devez absolument en apprendre davantage sur une approche fonctionnelle de la programmation, par opposition à l'approche de la programmation basée sur les classes que nous avons apprise dans le [chapitre sur la programmation orientée objet](#) :

- [Functional Programming Howto by A.M. Kuchling](#)
- [Functional programming chapter in 'Dive Into Python' book](#)
- [Functional Programming with Python presentation](#)
- [Fancy library](#)
- [PyToolz library](#)

Récapitatif

Vous êtes maintenant arrivé à la fin de ce livre, mais comme on dit, c'est *le début de la fin!*. Vous êtes maintenant un utilisateur Python avide, prêt à résoudre de nombreux problèmes avec Python. Vous pouvez commencer à automatiser des tâches sur votre ordinateur ou faire de nombreuses choses auparavant inimaginables ou écrire vos propres jeux et bien plus. Donc allez-y !

Annexe: FLOSS

NOTE: Veuillez noter que cette section a été écrite en 2003, elle vous semblera donc pittoresque :-)

« Le logiciel libre et à code source ouvert », ou [FLOSS](#) est basé sur le concept de communauté, qui lui-même basé sur le concept de partage, et en particulier partage des connaissances. Les logiciels libres sont gratuits à l'utilisation, mais aussi pour leur modification et redistribution.

Si vous avez déjà lu ce livre, alors vous êtes déjà familier avec le logiciel libre car vous avez utilisé *Python* pendant la lecture et Python est un logiciel open source!

Voici quelques exemples de logiciels libre pour donner une idée du genre de choses que le partage et la construction de communauté peuvent créer:

Linux: Il s'agit du noyau du système d'exploitation GNU/Linux. Linux, le noyau, a été créé par Linus Torvalds quand il était étudiant. Android est basé sur Linux. Tous les sites Web que vous utilisez ces jours-ci fonctionnent principalement sous Linux.

Ubuntu: Il s'agit d'une distribution axée sur la communauté, sponsorisée par Canonical. Il s'agit de la distribution GNU/Linux la plus populaire à l'heure actuelle. Elle vous permet d'installer une pléthore de logiciels libres et tout cela d'une manière facile à utiliser et à installer. Mieux encore, vous pouvez simplement redémarrer votre ordinateur et exécuter GNU/Linux à partir d'un CD! Cela vous permet d'essayer complètement ce nouveau système d'exploitation avant de l'installer sur votre ordinateur. Cependant, Ubuntu n'est pas entièrement un logiciel libre; il contient des pilotes, micrologiciels et des applications propriétaires.

LibreOffice: Il s'agit d'une excellente suite bureautique développée et développée par la communauté avec, entre autres, un logiciel de traitement de texte, de diaporama, un tableur et des composants de dessin. Il peut même ouvrir et éditer facilement des fichiers MS Word et MS PowerPoint. Il fonctionne sur presque toutes les plateformes et constitue un logiciel entièrement libre et open source.

Mozilla Firefox: C'est le meilleur navigateur Web. Il est extrêmement rapide et a été salué par la critique pour ses fonctionnalités sensées et impressionnantes. Le concept d'extensions permet d'utiliser n'importe quel type de plugin.

Mono: Ceci est une implémentation open source de la plate-forme [Microsoft .NET](#). Il permet aux applications .NET d'être créées et exécutées sous GNU/Linux, Windows, FreeBSD, Mac OS et de nombreuses autres plates-formes.

Apache web server: C'est le, très populaire, serveur web open source. En fait, c'est le serveur web le plus populaire de la planète! Il gère presque plus de la moitié des sites Web. Et oui - Apache gère plus de sites Web que tous ses concurrents (y compris Microsoft IIS) réunis.

VLC Player: Ceci est un lecteur vidéo capable de lire tout ce qui va du DivX au MP3 en passant par le format Ogg, les VCD, les DVD... qui a dit que l'open source ne pouvait pas être amusant? ;-)

Cette liste a simplement pour but de vous donner une idée - il existe de nombreux autres excellents logiciels libres, tels que le langage Perl, le langage PHP, le système de gestion de contenu Drupal, le serveur de base de données PostgreSQL, le jeu de courses TORCS, l'éditeur KDevelop, le lecteur de film Xine, l'éditeur VIM, l'éditeur Quanta+, le lecteur audio Banshee, le programme d'édition d'images GIMP, ... Cette liste pourrait continuer sans fin.

Pour être au courant des derniers bruits du monde du logiciel libre, consultez les sites suivants :

- [OMG! Ubuntu!](#)
- [Web Upd8](#)
- [DistroWatch](#)
- [Planet Debian](#)

Visitez les sites suivants pour plus d'information sur le logiciel libre:

- [GitHub Explore](#)
- [Code Triage](#)
- [SourceForge](#)
- [FreshMeat](#)

Donc, allez-y et explorez le vaste, libre et gratuit monde du logiciel libre!

Annexe: Colophon

Presque tous les logiciels que j'ai utilisés dans la création de ce livre sont [libres](#).

Naissance du livre

Dans la première révision de ce livre, j'avais utilisé Linux Red Hat 9.0 puis Linux Fedora Core 3 à partir de la sixième révision.

Au départ, j'utilisais KWord pour écrire le livre (comme expliqué dans la [leçon d'histoire](#)).

Adolescence

Plus tard, je suis passé à DocBook XML en utilisant Kate, mais j'ai trouvé ça trop fastidieux. Je suis donc passé à OpenOffice, qui était parfais de part le niveau de contrôle qu'il offrait pour le formatage ainsi que pour la génération de PDF. Il produisait cependant un code HTML de mauvaise qualité.

Enfin, j'ai découvert XEmacs et j'ai réécrit le livre de zéro au format DocBook XML (encore une fois) après avoir décidé que ce format était la solution à long terme.

Pour la sixième révision, j'ai décidé d'utiliser Quanta+ pour effectuer toutes les modifications. J'utilisais les feuilles de style XSL standard fournies avec Linux Fedora Core 3. Cependant, j'avais écrit un document CSS pour styler et colorer le rendu HTML. J'avais également écrit un analyseur lexical simple, en Python bien sûr, qui fournissait automatiquement la coloration syntaxique pour les exemples.

Pour la septième révision, j'ai utilisé [MediaWiki](#) comme base de ma configuration. J'avais l'habitude de tout éditer en ligne et les lecteurs pouvaient lire / éditer / discuter directement sur le site wiki, mais au final, je passais plus de temps à lutter contre le spam qu'à écrire.

Pour la huitième révision, j'ai utilisé [Vim](#), [Pandoc](#) et Mac OS X.

Pour la neuvième révision, je suis passé au format [AsciiDoc](#) et j'ai utilisé [Emacs 24.3](#), le thème tomorrow, la police Fira Mono et le mode adoc pour écrire.

De nos jours

2016: Je me suis fatigué de plusieurs problèmes de rendu mineurs avec AsciiDoctor, comme le `++` de `c/c++` qui disparaissait et il m'était difficile de ne pas oublier d'échapper ces petits détails. De plus, j'étais devenu réticent à éditer le texte à cause du format complexe d'Asciidoc.

Pour la dixième révision, je suis passé à l'écriture au format Markdown + [GitBook](#), à l'aide de l'[éditeur Spacemacs](#).

À propos de l'auteur

Voir <https://www.swaroopch.com/about/>

Leçon d'histoire

J'ai commencé avec Python quand j'ai eu besoin d'écrire un installeur pour un logiciel que j'avais écrit appelé « Diamant » afin de faciliter l'installation. Je devais choisir entre les bindings Python et Perl pour la librairie Qt. J'ai cherché sur le web et je suis tombé sur [un article d'Eric S. Raymond](#), le hacker célèbre et respecté, qui expliquait comment Python était devenu son langage favori. J'ai également trouvé que les bindings PyQt étaient plus matures que Perl-Qt. Donc, je décidai de choisir Python.

Puis, je cherchais un bon livre sur Python. Je n'ai pu en trouver aucun ! J'ai trouvé des livres chez O'Reilly, mais ils étaient soit trop chers, soit plus un manuel de référence qu'un guide. Donc je consultais la documentation livrée avec Python. Cependant c'était trop succinct et petit. Cela m'a donné une bonne idée à propos de Python, mais ce n'était pas complet. Je me débrouillai avec car j'avais déjà une expérience de la programmation, mais ce n'était pas adapté aux débutants.

Six mois après ce premier contact avec Python, j'installais le (à ce moment-là) dernier Red Hat 9.0 Linux et je jouais avec KWord. J'étais enthousiaste à ce sujet, et j'ai eu d'un coup l'idée d'écrire un truc sur Python. Je commençais à écrire quelques pages, mais rapidement cela devint 30 pages. Puis, je décidai de faire quelque chose d'utile sous la forme d'un livre. Après *beaucoup* de corrections, cela a atteint une étape où c'est devenu un guide utile pour apprendre le langage Python. Je considère ce livre comme ma contribution et mon hommage à la communauté open-source.

Ce livre a commencé comme des notes personnelles sur Python et je le considère toujours comme cela, cependant j'ai fait de nombreux efforts pour le rendre lisible au plus grand nombre :)

Dans le véritable esprit de l'open source, j'ai reçu beaucoup de suggestions constructives, critiques et [feedback](#) des lecteurs enthousiastes ont aidé à beaucoup améliorer le livre.

Statut du livre

Le livre a besoin de l'aide de ses lecteurs tels que vous pour indiquer les parties du livre qui ne sont pas bonnes, qui ne sont pas compréhensibles ou qui sont simplement fausses. Veuillez [écrire à l'auteur principal](#) ou aux [traducteurs](#) respectifs avec vos commentaires et suggestions.

Historique des révisions

- 4.0
 - 19 janvier 2016
 - Retour au Python 3
 - Retour au Markdown, en utilisant [GitBook](#) et [Spacemacs](#)
- 3.0
 - 31 mars 2014
 - Re-écriture pour Python 2 en utilisant [AsciiDoc](#) et [adoc-mode](#).
- 2.1
 - 03 août 2013
 - Re-écriture en Markdown avec le [Mode Markdown de Jason Blevins](#)
- 2.0
 - 20 octobre 2012
 - Re-écriture au [format Pandoc](#), grâce à ma femme qui a réalisé la majorité de la conversion à partir du format Mediawiki
 - Simplification du texte en supprimant des sections non essentielles telles que les variables `nonlocal` et les `metaclass`
- 1.90
 - 04 septembre 2008, toujours en cours
 - Réveil après une interruption de 3 ans et demi!

- Re-écriture pour Python 3.0
 - Conversion au format <http://www.mediawiki.org>[MediaWiki] (à nouveau)
- 1.20
 - 13 janvier 2005
 - Re-écriture complète au format [Quanta+](#) sur [Fedora](#) Core 3 avec beaucoup de corrections et de mises à jour. Beaucoup de nouveaux exemples. J'ai réécrit ma configuration DocBook de zéro.
 - 1.15
 - 28 mars 2004
 - Révisions mineures
 - 1.12
 - 16 mars 2004
 - Ajouts et corrections
 - 1.10
 - 09 mars 2004
 - De nouvelles corrections de fautes de frappe, grâce à de nombreux retours de lecteurs enthousiastes et aidants.
 - 1.00
 - 08 mars 2004
 - Après de nombreux commentaires et suggestions de la part de lecteurs, j'ai apporté des modifications importantes au contenu, ainsi que des corrections de fautes de frappe.
 - 0.99
 - 22 février 2004
 - Ajout d'un nouveau chapitre sur les modules. Ajout de détails sur les fonctions à nombre variable d'arguments.
 - 0.98
 - 16 février 2004
 - Ecriture d'un script Python et une feuille de style CSS pour améliorer la sortie XHTML, y compris un analyseur lexical, simple mais fonctionnel, pour la coloration automatique de la syntaxe (comme VIM).
 - 0.97
 - 13 février 2004
 - Un autre brouillon complètement réécrit, au format DocBook XML (à nouveau). Le livre s'est beaucoup amélioré - il est plus cohérent et lisible.
 - 0.93
 - 25 janvier 2004
 - Ajout de contenu sur IDLE et plus de choses spécifiques à Windows
 - 0.92
 - 05 janvier 2004
 - Modifications de quelques exemples.
 - 0.91
 - 30 Dec 2003
 - Correction de fautes de frappe. Amélioration de nombreux sujets.
 - 0.90
 - 18 Dec 2003
 - Ajout de 2 chapitres supplémentaires. Format [OpenOffice](#) avec révisions.
 - 0.60
 - 21 Nov 2003
 - Entièrement réécrit et étendu.
 - 0.20

- 20 Nov 2003
- Correction de fautes de frappe et d'erreurs.
- 0.15
 - 20 Nov 2003
 - Conversion au format [DocBook XML](#) avec XEmacs.
- 0.10
 - 14 Nov 2003
 - Brouillon initial avec [KWord](#).

Traductions

De nombreuses traductions du livre sont disponibles dans différentes langues, grâce à de nombreux bénévoles!

Si vous souhaitez aider avec ces traductions, veuillez consulter la liste des volontaires et des langues ci-dessous et décidez si vous souhaitez commencer une nouvelle traduction ou aider un projet de traduction existant.

Si vous envisagez de démarrer une nouvelle traduction, veuillez lire le [Guide de traduction](#).

Allemand

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) et Christoph Zworschke (cito@online.de) se sont portés volontaires pour traduire le livre en allemand.

La traduction peut être trouvée à l'adresse http://cito.github.io/byte_of_python/

Lutz Horn dit:

J'ai 32 ans et suis diplômé en mathématiques de l'Université de Heidelberg, en Allemagne. Je travaille actuellement en tant qu'ingénieur logiciel sur un projet financé par des fonds publics visant à créer un portail Web pour tout ce qui a trait à l'informatique en Allemagne. Le langage principal que j'utilise en tant que professionnel est Java, mais j'essaie de faire le maximum avec Python dans les coulisses. En particulier, l'analyse et la conversion de texte sont très faciles avec Python. Je ne connais pas très bien les boîtes à outils d'interface graphique, car la majeure partie de ma programmation concerne des applications Web, l'interface utilisateur étant construite à l'aide de frameworks Java tels que Struts. Actuellement, j'essaie de tirer davantage parti des fonctionnalités de programmation fonctionnelles de Python et des générateurs. Après avoir jeté un coup d'œil à Ruby, j'ai été très impressionné par l'utilisation des blocs dans ce langage. Généralement, j'aime les langages dynamiques comme Python et Ruby, car cela me permet de faire des choses impossibles dans des langages plus statiques comme Java. J'ai recherché une sorte d'initiation à la programmation, appropriée pour enseigner à un débutant. J'ai trouvé le livre « Penser comme un informaticien: apprendre avec Python » et « Plonger dans Python ». Le premier est bon pour les débutants mais trop long à traduire. Le deuxième ne convient pas aux débutants. Je pense que « A Byte of Python » se situe bien entre ceux-ci, car il n'est pas trop long, écrit dans le bon ordre, et en même temps assez prolix pour enseigner à un débutant. De plus, j'aime bien la structure simple de DocBook, qui fait que traduire le texte et générer les divers formats de sortie à été un plaisir.

Bernd Hengelein dit:

Lutz et moi allons faire la traduction allemande ensemble. Nous venons juste de commencer l'intro et la préface, mais nous vous tiendrons au courant des progrès réalisés. Ok, maintenant quelques choses personnelles. J'ai 34 ans et je joue avec des ordinateurs depuis les années 1980, lorsque le « Commodore C64 » était le maître incontesté. Après des études en informatique, j'ai commencé à travailler comme ingénieur en logiciel. Actuellement, je travaille dans le domaine de l'imagerie médicale pour une grande entreprise allemande. Bien que C++ soit mon langage principal que j'utilise (obligatoirement) pour mon travail quotidien, je suis constamment à la recherche de nouvelles choses à apprendre. L'année dernière, je suis tombée amoureuse de Python, un langage merveilleuse, tant par ses possibilités que par sa beauté. J'ai lu quelque part sur le réseau un type qui disait qu'il aime le Python, parce que le code est si beau. À mon avis, il a absolument raison. A l'époque où j'ai décidé d'apprendre le Python, j'ai remarqué qu'il y avait très peu de bonne documentation disponible en allemand. Lorsque je suis tombé sur votre livre, l'idée spontanée d'une traduction en allemand m'a traversé l'esprit. Heureusement, Lutz a eu la même idée et nous pouvons maintenant nous diviser le travail. Je suis impatient de notre coopération!

Arabe

Ci-dessous le lien pour la version arabe. Merci à Ashraf Ali Khalaf pour la traduction du livre, vous pouvez lire le livre en entier à l'adresse <http://www.khaledhosny.org/byte-of-python/index.html> ou le télécharger à partir de sourceforge.net pour plus d'informations, voir http://itwadi.com/byteteofpython_arabi.

Azéri

Jahangir Shabiyev (c.shabiev@gmail.com) s'est porté volontaire pour traduire le livre en azéri. La traduction est en cours à l'adresse <https://www.gitbook.com/book/jahangir-sh/piton-sancmasi>

Catalan

Moises Gomez (moisessomezgiron@gmail.com) s'est porté volontaire pour traduire le livre en catalan. La traduction est en cours.

Moisès Gómez - Je suis développeur et enseignant en programmation (normalement pour des personnes sans expérience préalable).

Il y a quelque temps, j'avais besoin d'apprendre à programmer en Python et le travail de Swaroop était vraiment utile. Clair, concis et complet. Juste ce dont j'avais besoin.

Après cette expérience, j'ai pensé que d'autres personnes de mon pays pourraient également en profiter. Mais la langue anglaise peut être un obstacle.

Alors, pourquoi ne pas essayer de le traduire? Et je l'ai fait pour une version précédente de BoP.

Dans mon pays, il existe deux langues officielles. J'ai choisi la langue catalane en supposant que d'autres la traduiront en espagnol plus répandu.

Chinois

En 2017, soit après 11 ans, Mo Lun (i@molun.net) a retraduit le livre depuis le début sur la base de la version 4.0. Et la traduction est stockée dans GitHub et Gitbook. Il continue de suivre cette édition traduite et est prêt à la corriger s'il y a une erreur ou une erreur dans le BoP traduit.

L'édition 2017 de la traduction est disponible à l'adresse <https://bop.molun.net>.

Mo Lun dit:

Je suis un étudiant en journalisme de la CYU, à Beijing. Et en fait, j'étais un débutant absolu en programmation Python lorsque j'ai commencé à traduire ce livre. Au début, ce n'était qu'un caprice, mais lorsque j'ai effectué ce travail, j'ai réalisé qu'une décision prise par intérêt m'avait poussé à avancer autant.

Avec l'aide des traductions de mes prédécesseurs et de la grande quantité d'informations fournie par Internet, et avec l'aide de mes amis, j'ai prudemment présenté cette édition traduite. J'espère juste que mon travail de traduction aidera d'autres nouveaux arrivants à apprendre Python.

En même temps, j'attends des commentaires et suggestions et je suis prêt à changer ou à améliorer le travail superficiel réalisé sur cette traduction.

Traduction antérieure en chinois

En 2005, Shen Jieyuan a traduit la version 1.20 de ce livre en chinois et l'a publié sur Internet. Ceci est la première édition chinoise. Sur le site officiel de BoP, il s'appelait Juan Shen, avec comme adresse email orion_val@163.com. Cette édition a été largement diffusée sur le réseau mais les liens fournis sur le site officiel de BoP ne sont plus fonctionnels, et il n'a pas été possible de retrouver le document original. Vous pouvez essayer de rechercher des mots-clés tels que « 简明Python教程 沈洁元 » pour en trouver une copie.

Juan Shen dit:

Je suis un étudiant de troisième cycle à la Wireless Telecommunication Graduate School de l'Université de Technologie de Beijing, en République populaire de Chine. Mon domaine de recherche actuel porte sur la synchronisation, l'estimation de canal et la détection multi-utilisateurs du système CDMA à porteuses multiples. Python est mon principal langage de programmation pour la simulation quotidienne et mon travail de recherche, avec l'aide de numpy. J'ai appris le python il y a à peine six mois, mais c'est, comme vous pouvez le constater, très facile à comprendre, facile à utiliser et productif. Tout comme ce qui est assuré dans le livre de Swaroop, « C'est mon langage de programmation préféré maintenant ».

« A Byte of Python » est mon tutoriel pour apprendre le Python. C'est une moyen clair et efficace de vous attirer dans le monde de Python dans les plus brefs délais. Ce n'est pas trop long, mais couvre efficacement presque toutes les choses importantes. Je pense que « A Byte of Python » devrait être fortement recommandé aux débutants en tant que premier tutoriel sur Python. Je dédie ma traduction aux millions d'utilisateurs potentiels de Python en Chine.

Chinois traditionnel

Fred Lin (gasolin@gmail.com) s'est porté volontaire pour traduire le livre en chinois traditionnel.

Il est disponible à l'adresse <http://code.google.com/p/zhpy/wiki/ByteOfZhpy>.

Une caractéristique intéressante de cette traduction est qu'elle contient également les sources *executable en python chinois* côté à côté avec les sources python d'origine.

Fred Lin - Je travaille en tant qu'ingénieur de micrologiciel réseau chez Delta Network, et je contribue également au framework Web TurboGears.

En tant qu'évangéliste python (:-p), j'ai besoin de matériel pour promouvoir le langage python. J'ai trouvé que « A Byte of Python » est le bon compromis pour parler à des débutants et des programmeurs expérimentés. « A Byte of Python » élabore l'essentiel de Python pour une longueur abordable.

Les traductions sont à l'origine basées sur la version chinoise simplifiée, mais rapidement beaucoup de réécritures ont été faites pour correspondre à la version actuelle du wiki et améliorer la qualité de la lecture.

La version traditionnelle chinoise récente contient également des sources exécutables en python chinois, qui sont réalisées dans le cadre de mon nouveau projet « zhpy » (python en chinois) (lancement à partir du 07 août).

zhpy (prononcez (Z.H.? ou zippy) construit une couche au dessus de Python pour traduire ou interagir avec python en chinois (traditionnel ou simplifié). Ce projet est principalement destiné au monde de l'éducation.

Coréen

Jeongbin Park (pjb7687@gmail.com) a traduit le livre en coréen - https://github.com/pjb7687/byte_of_python

Je suis Jeongbin Park, travaillant actuellement en tant que chercheur en biophysique et bioinformatique en Corée.

Il y a un an, je cherchais un bon tutoriel/guide pour présenter Python à mes collègues, car l'utilisation de Python dans de tels domaines de recherche devient inévitable du fait que le nombre d'utilisateurs ne cesse de croître.

Mais à cette époque, seuls quelques livres en Python étaient disponibles en coréen, j'ai donc décidé de traduire votre ebook, car c'est l'un des meilleurs guides que j'ai jamais lus!

Actuellement, le livre est presque entièrement traduit en coréen, à l'exception d'une partie du texte du chapitre d'introduction et des annexes.

Merci encore d'avoir écrit un si bon guide!

Espagnol

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) et Cristian Bermudez Serna (crisbermud@hotmail.com) ont proposé de traduire le livre en espagnol.

Gustavo Echeverria dit:

Je travaille en tant qu'ingénieur logiciel en Argentine. J'utilise principalement les technologies C# et .NET au travail, mais n'utilise que Python ou en Ruby pour mes projets personnels. Je connaissais Python il y a de nombreuses années et je me suis retrouvé coincé immédiatement. Peu de temps après avoir découvert Python, j'ai découvert ce livre et cela m'a aidé à apprendre le langage. Je me suis porté volontaire pour traduire le livre en espagnol. Maintenant, après avoir reçu quelques demandes, j'ai commencé à traduire « A Byte of Python » avec l'aide de Maximiliano Soler.

Cristian Bermudez Serna a déclaré:

Je suis étudiant en ingénierie des télécommunications à l'Université d'Antioquia (Colombie). Il y a quelques mois, j'ai commencé à apprendre le python et j'ai trouvé ce merveilleux livre. Je me suis donc porté volontaire pour arriver à une traduction en espagnol.

Français

Gregory (coulix@ozforces.com.au) s'est porté volontaire pour traduire le livre en français.

Gérard Labadie (gerard.labadie@gmail.com) a terminé la traduction du livre en français.

Cette traduction a été portée au format gitbook, mise à jour, et remise en ligne par Romain Gilliotte (rgilliotte@gmail.com).

Elle est maintenant hébergée sur GitBook à l'adresse <https://rgilliotte.gitbook.io/byte-of-python/>

Grec

La communauté grecque Ubuntu [a traduit le livre en grec](#), pour l'utiliser dans nos leçons asynchrone de Python en ligne que nous proposons sur nos forums. Contactez [@savvasradevic](#) pour plus d'informations.

Indonésien

Daniel (daniel.mirror@gmail.com) traduit le livre en indonésien à l'adresse <http://python.or.id/moin.cgi/ByteofPython>.

Wisnu Priyambodo (cibermen@gmail.com) s'est également porté volontaire pour traduire le livre en indonésien.

Bagus Aji Santoso (baguzzaji@gmail.com) s'est porté volontaire.

Italien

Enrico Morelli (mr.mlucci@gmail.com) et Massimo Lucci (morelli@cerm.unifi.it) se sont portés volontaires pour traduire le livre en italien.

La traduction italienne est présente à l'adresse <http://www.gentoo.it/Programmazione/bytewofpython>.

Massimo Lucci et Enrico Morelli: nous travaillons à l'Université de Florence (Italie) au département de chimie. Massimo en tant qu'ingénieur de service et administrateur système pour les spectromètres à résonance magnétique nucléaire; Enrico en tant qu'ingénieur de service et administrateur système pour notre CED et nos systèmes parallèles / en cluster. Nous programmons sur Python depuis environ sept ans et nous travaillons avec la plateforme Linux depuis dix ans. En Italie, nous sommes responsables et administrateurs du site Web www.gentoo.it pour la distribution Gentoo/Linux et du site www.nmr.it (en construction) pour les applications de la résonance magnétique nucléaire et l'organisation et la gestion du Congrès. C'est tout! Nous sommes impressionnés par le langage intelligent utilisé dans votre livre et estimons qu'il est essentiel pour amener Python à de nouveaux utilisateurs (nous pensons à une centaine d'étudiants et de chercheurs travaillant dans nos laboratoires).

Une deuxième traduction en italien a été créée par [Calvina Bice](#) et ses collègues sur <http://besthcgdropwebsite.com/translate/a-byte-of-python/>.

Japonais

Shunro Dozono (dozono@gmail.com) est en train de traduire le livre en japonais.

Mongol

Ariunsanaa Tunjin (luftballons2010@gmail.com) s'est portée volontaire pour traduire le livre en mongol.

Mise à jour le 22 novembre 2009: Ariunsanaa est sur le point de terminer la traduction.

Norvégien (bokmål)

Eirik Vågeskar est élève du secondaire à [Sandvika videregående skole](#) en Norvège, et un [blogueur](#). Il traduit actuellement le livre en norvégien (bokmål).

Eirik Vågeskar: J'ai toujours voulu programmer, mais comme je parle une petite langue, le processus d'apprentissage a été beaucoup plus difficile. La plupart des didacticiels et des livres sont rédigés dans un anglais très technique. Par conséquent, la plupart des diplômés du secondaire n'ont pas le vocabulaire nécessaire pour comprendre le fond des didacticiels. Quand j'ai découvert ce livre, tous mes problèmes ont été résolus. « A Byte of Python » utilisait un langage simple, non technique, pour expliquer un langage de programmation tout aussi simple, ce qui rend l'apprentissage de Python amusant. Après avoir lu la moitié du livre, j'ai décidé qu'il valait la peine qu'on le traduise. J'espère que la traduction aidera les personnes qui se sont retrouvées dans la même situation que moi (en particulier les jeunes) et peut-être aussi de faire en sorte de susciter un intérêt envers ce langage chez les personnes possédant un bagage technique plus limité.

Polonais

Dominik Kozaczko (dominik@kozaczko.info) s'est porté volontaire pour traduire le livre en polonais. La traduction est en cours et sa page principale est disponible ici: [Ukás Pythona](#).

Mise à jour: La traduction est terminée et prête depuis le 2 octobre 2009. Merci à Dominik, ses deux étudiants et leur ami pour leur temps et leurs efforts!

Dominik Kozaczko - Je suis un enseignant en informatique et en technologies de l'information.

Portugais

Il existe trois traductions à différents niveaux d'achèvement et d'accessibilité. L'ancienne traduction est maintenant manquante / perdue et la nouvelle traduction est incomplète.

Samuel Dias Neto (samuel.arataca@gmail.com) a réalisé la première traduction en portugais brésilien (ancienne traduction) de ce livre lorsque Python était en version 2.3.5. Ce n'est plus accessible au public.

Artur Weber (arturweberguimaraes@gmail.com) a achevé la traduction de ce livre en portugais (le 21 février 2018) à l'adresse <https://www.homeyou.com/~edu/introducao>.

Artur Weber: Mes étudiants étudient à la faculté polytechnique de l'Université écologique de la ville de Curitiba (Brésil) et certains d'entre eux s'intéressent à différents papiers.

Comme ils écrivent des cours et des articles académiques, ils recherchent toujours des articles et des pages intéressants. Je fais aussi de mon mieux pour trouver des matériaux intéressants qui peuvent être la source de leurs travaux universitaires.

J'ai trouvé le matériel de votre site utile pour certains de mes étudiants qui écrivent des articles qui nécessitent de la programmation en Python. En fait, c'est la raison pour laquelle j'ai décidé de faire une traduction en portugais pour permettre à mes étudiants qui ne maîtrisaient pas l'anglais de lire ce livre passionnant dans leur langue maternelle (en portugais).

Rodrigo Amaral (rodrigoamaral@gmail.com) s'est porté volontaire pour traduire le livre en portugais brésilien (nouvelle traduction), qui reste à compléter.

Roumain

Paul-Sebastian Manole (brokenthorn@gmail.com) s'est porté volontaire pour traduire ce livre en roumain.

Paul-Sebastian Manole - Je suis un étudiant de deuxième année en informatique à l'université Spiru Haret, ici en Roumanie. Je suis un programmeur autodidacte et j'ai décidé d'apprendre un nouveau langage, Python. Le Web m'a dit qu'il n'y avait pas de meilleur moyen de le faire que de lire « A Byte of Python ». C'est à quel point ce livre est populaire (félicitations à l'auteur pour avoir écrit un livre aussi facile à lire). J'ai commencé à aimer Python et j'ai donc décidé d'aider à traduire la dernière version du livre de Swaroop en roumain. Bien que je puisse être le premier à prendre l'initiative, je ne suis qu'un volontaire, alors si vous pouvez aider, rejoignez-moi s'il vous plaît.

Russe

Vladimir Smolyar (v_2e@ukr.net) a terminé une traduction en russe à l'adresse <http://wombat.org.ua/AByteOfPython/>.

Serbe

« BugSpice » (amortizerka@gmail.com) a achevé la traduction en serbe:

Ce lien de téléchargement n'est plus accessible.

Plus de détails à l'adresse <http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona>.

Slovaque

Albertio Ward (albertioward@gmail.com) a traduit le livre en slovaque à l'adresse <http://www.fatcow.com/edu/python-swaroopch-sl/>:

Nous sommes une organisation à but non lucratif appelée « Traduction pour l'éducation ». Nous représentons un groupe de personnes, principalement des étudiants et des professeurs, de la *Slavonic University*. Nous avons des étudiants de différents départements: linguistique, chimie, biologie, etc... Nous essayons de trouver sur Internet des publications intéressantes qui peuvent être pertinentes pour nous et nos collègues universitaires. Parfois, nous trouvons des articles par nous-mêmes; D'autres fois, nos professeurs nous aident à choisir le matériel à traduire. Après avoir obtenu l'autorisation des auteurs, nous traduisons des articles et les publions sur notre blog, qui est disponible et accessible à nos collègues et amis. Ces publications traduites aident souvent les étudiants dans leur études.

Suédois

Mikael Jacobsson (leochingkwake@gmail.com) s'est porté volontaire pour traduire le livre en suédois.

Turc

Türker Sezer (tsezer@btturk.net) et Bugra Cakir (bugracakir@gmail.com) se sont portés volontaires pour traduire le livre en turc. « Où est la version turque? Bitse de okusak. »

Ukrainien

Averkiev Andrey (averkiyev@ukr.net) s'est porté volontaire pour traduire le livre en russe et peut-être en ukrainien (si le temps le permet).

Comment traduire ce livre

1. Les sources de ce livre sont disponibles à l'adresse <https://github.com/swaroopch/byte-of-python>.
2. [Forkez le dépôt](#).
3. Ensuite, récupérez le dépôt sur votre ordinateur. Vous devez savoir comment utiliser [Git](#) pour le faire.
4. Lisez la [documentation GitBook](#), en particulier la [section Markdown](#).
5. Commencez à éditer les fichiers `.md` pour les traduire dans votre langue.
6. [Inscrivez-vous sur GitBook.com](#), créez un livre et vous pourrez voir un site Web magnifiquement rendu, avec des liens pour télécharger les fichiers PDF, EPUB, etc.

Feedback

Le livre a besoin de l'aide de ses lecteurs tels que vous pour indiquer les parties du livre qui ne sont pas bonnes, qui ne sont pas compréhensibles ou qui sont simplement fausses. Veuillez [écrire à l'auteur principal](#) ou aux traducteurs respectifs avec vos commentaires et suggestions.