# [Démarrez votre projet avec Python](https://openclassrooms.com/fr/courses/4262331-demarrez-votre-projet-avec-python)

Table des matières

[Démarrez votre projet avec Python 1](#_Toc57216445)

[**Pourquoi apprendre Python ?** 4](#_Toc57216446)

[Pourquoi apprendre Python ? 4](#_Toc57216447)

[Comment tirer parti de ce cours ? 5](#_Toc57216448)

[Installez Python ! 8](#_Toc57216449)

[To Python 2, or to Python 3-- that is the question. 8](#_Toc57216450)

[Installer Python 9](#_Toc57216451)

[Premiers pas avec la console 9](#_Toc57216452)

[L'interpréteur Python 10](#_Toc57216453)

[Commencer à coder en Python 12](#_Toc57216454)

[Découvrez le vocabulaire de Python 13](#_Toc57216455)

[Variables 13](#_Toc57216456)

[Les messages d'erreur 14](#_Toc57216457)

[Types d'objets 15](#_Toc57216458)

[A vous de jouer ! 19](#_Toc57216459)

[Découvrez les bases de Python 20](#_Toc57216460)

[Compétences évaluées 20](#_Toc57216461)

[ Question 1 20](#_Toc57216462)

[ Question 2 20](#_Toc57216463)

[ Question 3 20](#_Toc57216464)

[ Question 4 21](#_Toc57216465)

[ Question 5 21](#_Toc57216466)

[ Question 6 21](#_Toc57216467)

[ Question 7 22](#_Toc57216468)

[ Question 8 22](#_Toc57216469)

[ Question 9 22](#_Toc57216470)

[ Question 10 22](#_Toc57216471)

[Créez votre premier script 24](#_Toc57216472)

[Créer un fichier externe 24](#_Toc57216473)

[Avant de continuer... 27](#_Toc57216474)

[Code du chapitre 28](#_Toc57216475)

[Comparez des valeurs avec les opérateurs 29](#_Toc57216476)

[Opérateurs de comparaison 29](#_Toc57216477)

[Les opérateurs mathématiques 30](#_Toc57216478)

[Ajoutez un peu de logique avec les conditions 32](#_Toc57216479)

[La condition "Si" 32](#_Toc57216480)

[Le cas échéant... 33](#_Toc57216481)

[Intégrer une condition à la suite d'une autre condition 33](#_Toc57216482)

[Code du chapitre 34](#_Toc57216483)

[Structurez votre programme en utilisant les fonctions 35](#_Toc57216484)

[Créez une nouvelle fonction 35](#_Toc57216485)

[Paramètres 36](#_Toc57216486)

[Utilisez les fonctions standards de Python 36](#_Toc57216487)

[Renvoyez une valeur 37](#_Toc57216488)

[Code du chapitre 37](#_Toc57216489)

[Répétez une action grâce aux boucles 39](#_Toc57216490)

[La boucle while 39](#_Toc57216491)

[La boucle for 41](#_Toc57216492)

[Code du chapitre 41](#_Toc57216493)

[Les fondations 43](#_Toc57216494)

[Compétences évaluées 43](#_Toc57216495)

[ Question 1 43](#_Toc57216496)

[ Question 2 43](#_Toc57216497)

[ Question 3 43](#_Toc57216498)

[ Question 4 44](#_Toc57216499)

[ Question 5 44](#_Toc57216500)

[ Question 6 44](#_Toc57216501)

[ Question 7 45](#_Toc57216502)

[ Question 8 45](#_Toc57216503)

[ Question 9 46](#_Toc57216504)

[ Question 10 46](#_Toc57216505)

[Modifiez des chaînes de caractères 47](#_Toc57216506)

[A vous de jouer ! 48](#_Toc57216507)

[Modifiez des nombres 49](#_Toc57216508)

[Rangez des données dans des listes 50](#_Toc57216509)

[A vous de jouer ! 51](#_Toc57216510)

[Organisez vos données dans un dictionnaire 52](#_Toc57216511)

[A vous de jouer ! 52](#_Toc57216512)

[Retour à notre programme 52](#_Toc57216513)

[Code du chapitre 53](#_Toc57216514)

[Manipulez des objets 54](#_Toc57216515)

[Compétences évaluées 54](#_Toc57216516)

[ Question 1 54](#_Toc57216517)

[ Question 2 54](#_Toc57216518)

[ Question 3 54](#_Toc57216519)

[ Question 4 55](#_Toc57216520)

[ Question 5 55](#_Toc57216521)

[ Question 6 56](#_Toc57216522)

[ Question 7 56](#_Toc57216523)

[ Question 8 56](#_Toc57216524)

[ Question 9 56](#_Toc57216525)

[ Question 10 57](#_Toc57216526)

[Installez votre premier module 58](#_Toc57216527)

[Importer un module 58](#_Toc57216528)

[Utiliser les fonctions d’un module 59](#_Toc57216529)

[Utiliser le module random 59](#_Toc57216530)

[Code de ce chapitre 60](#_Toc57216531)

[Aller plus loin 60](#_Toc57216532)

[[BONUS] Stockez des citations dans un fichier avec le module json 61](#_Toc57216533)

[Code du chapitre 64](#_Toc57216534)

[[BONUS] Collectez des citations automatiquement avec Scrapy 65](#_Toc57216535)

[Le fichier requirements.txt 65](#_Toc57216536)

[Utiliser Scrapy 69](#_Toc57216537)

[Code de ce chapitre 72](#_Toc57216538)

**DEMARRER VOTRE PROJET AVEC PYTHON**

Apprendre à coder est devenu indispensable dans beaucoup de métiers. Que vous travailliez dans le marketing, la communication ou la finance, connaître les bases d’un langage vous ouvrira des portes.

Dans ce cours vous découvrirez toutes les bases de Python, un langage simple et facile à apprendre qui est utilisé dans la robotique, le big data et le web de manière plus générale.

Nous réaliserons ensemble un programme qui fera dire à des personnages de dessin animé des citations de [San Antonio](https://fr.wikipedia.org/wiki/San-Antonio_(s%C3%A9rie)). Et je vous promets que ça va être fun !

Prêt ? Inscrivez-vous !

**Objectifs pédagogiques :**

* Utiliser le vocabulaire de base de Python
* Poser les fondations d’un programme Python
* Manipuler des objets en Python

Prérequis : aucun, venez comme vous êtes !

Ce cours a été réalisé avec la collaboration inestimable de [Régis Behmo](https://openclassrooms.com/membres/regisb-1). Merci !

**Pourquoi apprendre Python ?**

Bienvenue dans ce cours à la découverte du monde fabuleux de la programmation ! Nous allons ensemble apprendre Python, un langage bien connu des scientifiques, des startups et des amateurs d’un certain groupe d’humoristes britanniques.

Oui, vous avez bien lu ! Pour expliquer l’origine du langage, revenons un peu en arrière. En 1989, par une froide nuit néerlandaise, un développeur du plat pays nommé [**Guido van Rossum**](https://en.wikipedia.org/wiki/Guido_van_Rossum) s’ennuie. Il cherche un moyen de s’occuper pendant la période qui précède Noël car les bureaux de son entreprise sont fermés. Quand certains auraient dévoré l’intégrale d’Urgences ou décoré un sapin, lui se lance dans l’invention d’un langage. Etant un grand fan des [**Monty Python**](https://en.wikipedia.org/wiki/Monty_Python) et d’humeur irrévérencieuse, il l’appelle Python. Voilà pourquoi les développeurs Python [**ont de l’humour**](https://www.python.org/doc/humor/) et s’amusent à glisser des petites blagues dans leur code !

### Pourquoi apprendre Python ?

Avant tout car **c’est fun** ! Oui oui, croyez-moi, nous allons nous amuser à créer rapidement et facilement des programmes qui rendront jaloux tous vos amis ! Grâce à ce langage, vous pourrez créer des programmes pour votre ordinateur, des sites web et même des jeux ! Pinterest, Instagram et le site du New York Times ont été développés en Python !



Civilization IV utilise Python pour la majorité de ses tâches

**Débuter facilement.** Python est un langage parfait pour débuter. Il fonctionne sur tous les systèmes d’exploitation (Windows, Mac et Linux) et vous n’avez pas à utiliser un logiciel spécifique pour voir le résultat de votre code. Vous avez juste besoin d’un ordinateur. Et de votre tête. Alouette.

**Gagner des sous.** Le salaire moyen d’un développeur Python **junior** en France se situe entre 35 000 et 40 000 [selon Hired](http://www.journaldunet.com/solutions/reseau-social-d-entreprise/1188855-le-salaire-des-developpeurs-python-en-graphiques/) et est en croissance selon [Urban Linker](https://www.urbanlinker.com/le-webzine/etude-des-salaires-des-metiers-tech-2016-en-idf-52), un cabinet de recrutement spécialisé dans les Startups.

**Apprendre un langage reconnu.** Python est le 4e langage le plus populaire selon l’[index TIOBE](http://www.tiobe.com/tiobe-index/python/) et son usage est resté stable depuis une dizaine d’années. Vous avez la garantie d’utiliser longtemps ce que vous apprendrez dans ce cours !  
Fan de statistiques ou de big data ? Python est un des langages principaux utilisés en data analysis (analyse de données) et en machine learning (apprentissage par la machine).  
Envie de créer un robot qui vous servirait le café le matin ? Python est également le langage de référence pour apprendre la robotique. [L’ordinateur le moins cher du monde](https://fr.wikipedia.org/wiki/Raspberry_Pi) ($25) a d’ailleurs été conçu dans cet objectif : rendre l’informatique abordable et ludique.

**Rejoindre une communauté mondiale.** Active et drôle, la communauté Python saura vous accueillir. Avec 27 000 membres sur [Meetup](https://www.meetup.com/fr-FR/topics/python/" \t "_blank) et plus de 40 groupes, il s’agit d’une des plus grandes communautés de France. En cherchant un peu vous trouverez certainement des rendez-vous dans la ville la plus proche pour aller boire un verre, parler code et débugger votre dernier programme. Vous rencontrez un souci ? Vous trouverez toujours une réponse sur Stack Overflow.  
Guido van Rossum étant un fervent contributeur à des projets Open Source, il n’est pas étonnant que le langage soit très utilisé dans de nombreux projets libres. Il s’agit d’une des forces de la communauté.

### Comment tirer parti de ce cours ?

Une de mes convictions pédagogiques les plus profondes est qu’**on apprend en faisant**, et uniquement en faisant. C’est pourquoi ce cours est organisé autour d’un projet que je développerai au fur et à mesure.

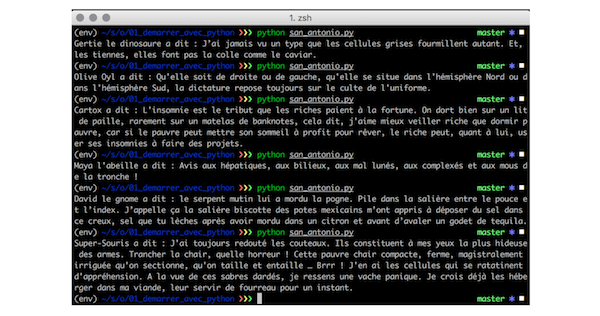
Chaque chapitre sera ponctué de petits exercices pratiques que vous pourrez réaliser directement dans notre console interactive. Entrez votre réponse et cliquez sur Run Code. Votre exercice est corrigé instantanément !

C'est le moment de vous entrainer !

[Cliquez sur ce lien.](https://www.codevolve.com/api/v1/publishable_key/2A9CAA3419124E3E8C3F5AFCE5306292?content_id=174cb64c-2aa1-42a3-8cdc-45a9260c789f)

Vous pouvez lire le cours sans pratiquer… mais ce serait un peu comme si vous appreniez à faire du vélo en lisant un manuel. Vous avez beau connaître la théorie, vous ne serez pas plus avancé·e.

Le projet fil-rouge de ce cours est un programme qui ira chercher des citations de [San Antonio](https://fr.wikipedia.org/wiki/San-Antonio_(s%C3%A9rie)) sur Internet et les fera dire par un personnage de dessin animé au hasard. Pour celles et ceux qui ne connaissent pas San Antonio, vous découvrirez bien vite pourquoi cela m’a fait tant rire de développer ce programme !



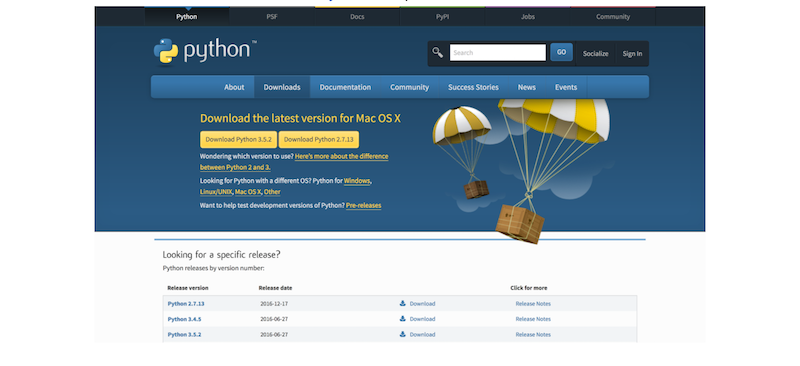
Dans ce cours, je vais vous accompagner dans la réalisation des étapes suivantes :

* Installer Python et faire connaissance avec la Console 👋
* Trouver comment “enregistrer” une citation et la retrouver plus tard.
* Créer des phrases sous la forme ‘<personnage> a dit : “<citation>”’ et la modifier automatiquement.
* Créer des listes pour stocker plusieurs citations et plusieurs personnages.
* Créer des “dictionnaires” pour attribuer plusieurs citations à un même personnage.
* Enregistrer votre programme dans un fichier externe car il commence à faire plusieurs lignes !
* Interagir avec notre utilisateur : quand il tape “entrée”, le programme doit afficher une nouvelle citation. Quand il tape “B”, le programme s’arrête. ☠
* Afficher une citation au hasard quand on lance le programme.
* BONUS : Stocker nos citations et nos personnages dans un fichier externe.
* BONUS : Coder un petit robot qui va parcourir le Web à la recherche de citations et de personnages puis les stocker dans un fichier sur votre ordinateur.

Allez, maintenant que je vous ai donné envie, il est temps de s’y mettre !

## Installez Python !

Rendez-vous sur [le site officiel de Python](https://www.python.org/) et cliquez sur Downloads.

[](https://www.python.org/)

### To Python 2, or to Python 3-- that is the question.

Le site vous offre la possibilité de télécharger deux versions : Python 2 ou Python 3.

Quelle différence me direz-vous ?

Tout logiciel est amené à être mis à jour régulièrement pour améliorer ses fonctionnalités ou réparer des bugs. Vous avez certainement déjà vu un petit message “Une nouvelle version de votre logiciel est prête à être installée !”. Souvent, ces mises à jour sont rétrocompatibles, c’est-à-dire que vous pouvez toujours ouvrir vos anciens fichiers avec la nouvelle version.

Par exemple, vous pouvez ouvrir un document Microsoft Word .doc créé sous Word 1998 avec la version de Word 2016 mais vous ne pourrez pas faire l’inverse !

À la fin des années 2000, la Python Software Foundation a travaillé à la création d’une toute nouvelle version du langage qui règlerait de nombreux soucis. Malheureusement, il était impossible d’assurer la rétrocompatibilité. Lorsque la version 3 de Python est sortie en février 2009, la communauté a hurlé car utiliser Python 3 supposait réécrire tous les projets qui utilisaient alors la version 2. Pourtant, les avancées étaient vraiment séduisantes. Vous comprenez un peu le dilemme ?!

Le débat Python 2 versus Python 3 continue de faire rage, même si de plus en plus de développeurs se rangent vers la dernière version. Cela va faire bientôt 10 ans qu’elle est sortie et les modules les plus utilisés ont déjà migré à Python 3. C’est pourquoi j’utiliserai cette version dans ce cours.

[En savoir plus sur les différences entre Python 2 et Python 3](https://wiki.python.org/moin/Python2orPython3)🇬🇧

### Installer Python

Cliquez sur Download Python 3.X.X (X étant les dernières versions en date)

#### Sous Mac

Décompressez le dossier que vous avez téléchargé en double-cliquant dessus. Puis cliquez de nouveau sur le document afin de lancer l’installation et laissez-vous guider !

Vous utilisez déjà votre console et vous souhaitez utiliser Brew ? Super ! Tapez simplement  brew install python3  et c’est fini !

#### Sous Windows

Enregistrez le dossier à télécharger puis suivez les instructions. Quand l’installation est terminée, vous pouvez vous rendre dans Démarrer > Tous les Programmes. Vous devriez y voir un nouveau dossier Python contenant, notamment, Python et IDLE.

#### Sous Linux

Python est pré-installé dans la plupart des distributions Linux mais sa version est certainement obsolète. Utilisez APT-GET pour l'installer ou mettre à jour Python.

Tapez  sudo apt-get install python3  dans votre console.

### Premiers pas avec la console

#### Sous Mac et Linux

La console est un petit programme qui vous permet d’interagir directement avec votre ordinateur en parlant son langage. Lorsque vous utilisez un logiciel, n’importe lequel, chaque action que vous effectuez avec votre souris est traduite en langage informatique puis exécutée par votre ordinateur. La console vous permet d’aller plus vite puisque vous n’utilisez plus d’interface graphique pour traduire vos commandes.

Au début, on a du mal à s’y faire. On se demande “Mais pourquoi ? C’est si simple avec une souris ! Là je dois retenir plein de commandes…”. C’est plus simple, certes, mais infiniment plus long ! Prenons un exemple basique : supprimer un fichier définitivement.

Normalement, vous allez effectuer un clic droit puis cliquer sur “supprimer” et votre fichier va être déplacé dans la corbeille. Vous allez ensuite ouvrir la corbeille et cliquer sur “Supprimer définitivement”. Le système d’exploitation va alors vous demander si vous êtes sûr·e de vouloir supprimer ce fichier pour toujours. Vous cliquez “oui”. Une bonne minute a dû s’écouler.

En utilisant votre console, une seule commande suffit.  rm -rf nomdudossier . Et voilà. C’est tout, fini, done, acabado, finito, 完成. Moins d’une seconde !

Cette console vous permet de vous déplacer dans l’ordinateur, d’en manipuler les fichiers et même de créer de petits programmes qui automatiseront certaines tâches. Lorsque vous interagissez avec la console, vous parlez en langage bash (sur Linux et Mac) ou en DOS (Windows). bash et DOS n’ont pas les mêmes super-pouvoirs que Python. Leur fonction est simplement de vous permettre d’interagir avec votre ordinateur ! Tandis que Python vous permet de créer des programmes, des sites, des jeux, ...

L'autre nom de la console est "terminal".

Ce cours n’a pas vocation à vous apprendre à utiliser la console et vous pouvez vous en sortir sans elle… pour l’instant. Mais vous serez très vite limité. C’est pourquoi je vous propose de lire, dès maintenant, l’excellent chapitre [**La Console, ça se mange ?**](https://openclassrooms.com/courses/reprenez-le-controle-a-l-aide-de-linux/la-console-ca-se-mange) du cours [**Reprenez le contrôle à l’aide de Linux !**](https://openclassrooms.com/courses/reprenez-le-controle-a-l-aide-de-linux)(prévoyez un quart d’heure environ).

Retrouvez-moi juste après !

#### Sous Windows

Windows met à votre disposition un outil qui s'appelle "Invite de commande" et est très similaire à la console dont je viens de parler. Il vous permet de vous déplacer dans votre ordinateur et de lui demander d'exécuter des actions pour vous. De manière schématique, vous pouvez réaliser grâce à cet outil la majorité des actions que vous effectuez avec la souris : ouvrir un fichier, supprimer un document, exécuter un programme, ...

Néanmoins, il existe de nombreuses différences qui le rendent difficile à manier pour un débutant (et désagréable pour les plus expérimentés).

Je vous explique, juste en-dessous, comment utiliser Python avec Windows !

### L'interpréteur Python

L'interpréteur Python est un programme qui va comprendre les instructions écrites en Python et les “transformer” en langage binaire. Vous n’avez pas envie d’écrire une suite de 0 et de 1 pour que votre programme fonctionne, n’est-ce pas ?!

La première étape est donc de lancer de programme afin d'entrer dans le monde fabuleux de Python ! Mais comment faire ? Encore une fois, le chemin est légèrement différent selon que vous utilisiez Windows ou Linux / Mac. Mais ne vous inquiétez pas : nous nous retrouverons au même endroit à la fin !

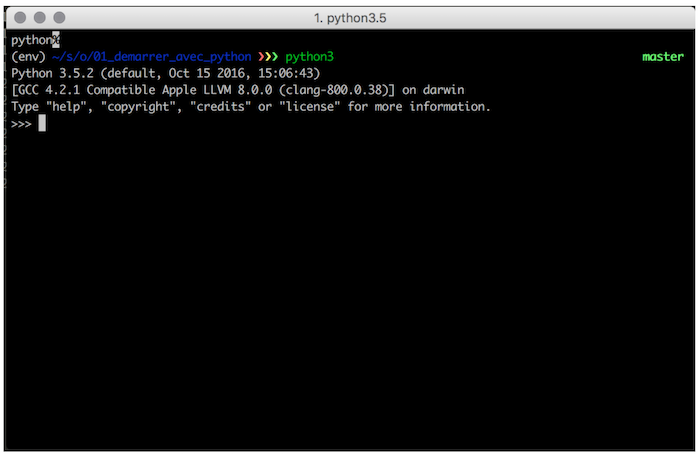
#### Lancer l'interpréteur via la console (Mac et Linux)

Ouvrez la console et entrez une commande magique :  python3 .

Appuyez sur la touche “entrée” après avoir tapé la commande !

Désormais vous ne pouvez plus vous déplacer dans votre ordinateur. Vous êtes dans un univers parallèle, celui de Python, avec un nouveau langage et de nouvelles possibilités.

Vous devriez voir apparaître l’écran suivant :



Les couleurs d’affichage varient selon les consoles. Ne vous affolez donc pas si la vôtre est différente de la mienne !

#### Lancer l'interpréteur sous Windows

Lorsque vous installez Python, vous installez également un autre programme qui s'appelle [IDLE](https://fr.wikipedia.org/wiki/IDLE_(Python)). Il s'agit d'un logiciel qui vous permettra d'écrire du code Python et de l'exécuter. [Découvrez-en plus](https://fr.wikipedia.org/wiki/IDLE_(Python)).

Cliquez sur le menu Démarrer, puis sur Python et Idle. Une fenêtre s'ouvre et le programme vous invite à écrire des commandes. Nous y sommes !

Idle

### Commencer à coder en Python

Bravo ! Vous pouvez maintenant coder en Python ! Pour commencer, tapez “7 + 7”. L'intepréteur vous répond : 14. Super ! (Oui, je sais, je suis très enthousiaste…)

Notez d’ailleurs que la console affiche  >>>  quand c’est à vous de parler, et rien du tout quand elle vous répond.

Vous êtes prêt·e pour la suite ! Voyons ensemble comment “enregistrer” dans la console une première citation.

Pour sortir de l’interpréteur, entrez simplement :  exit()  (attention aux parenthèses !) puis appuyez sur la touche entrée.

## Découvrez le vocabulaire de Python

C’est parti pour enregistrer notre première citation dans l’interpréteur ! Commençons par nous amuser un peu avec ce dernier. Vous avez déjà vu que vous pouviez vous en servir comme calculatrice. Mais saviez-vous que vous pouviez également lui demander de conserver des informations en mémoire ? Comme à un ami à qui vous diriez : “Peux-tu retenir que le numéro de téléphone de Laurène est le 07XXXXXXXX ? J’en aurai besoin tout à l’heure pour l’appeler.”.

Comment faire ?

Il suffit d’enregistrer le numéro de téléphone dans une **variable**.

### Variables

Chaque information que vous souhaitez réutiliser plus tard s’appelle une variable. Il s’agit d’un concept basique en programmation et que vous retrouverez dans tous les langages.

Nous connaissons tous les variables, sans le savoir, car nous avons tous fait (ou essayé de faire !) des maths.

Une variable garde en mémoire une information tant que l’interpréteur Python est ouvert. Si vous quittez le programme puis le redémarrez (en tapant  exit()  puis  python3  par exemple), toutes vos variables seront effacées. Python a la mémoire courte !

Nous pouvons voir une variable comme une étiquette que vous colleriez sur un objet pour vous souvenir de son nom. Afin de mieux comprendre ce concept, revenons à notre tendre enfance et souvenons-nous de la manière dont nous avons appris à parler. Une personne vous désignait les choses et vous disait son nom. "Ceci est du brocoli. Ceci est une table. Ceci est du feu". Cela vous a permis, par la suite, de faire référence à ces objets par ces mêmes termes : "Z'aime pas les brocolis. Veux monter sur la table. Le feu ça fait mal.".

Nous tenons ce concept de Bergson, philosophe du langage, qui s'est profondément intéressé aux mots que nous utilisons pour représenter les choses. Selon lui, "Nous ne voyons pas les choses mêmes; nous nous bornons, le plus souvent, à lire des étiquettes collées sur elles.". [**En savoir plus**](http://www.philosophie-spiritualite.com/textes_1/Bergson24.htm)

Il en est de même en programmation. Vous pouvez indiquer à l'ordinateur que vous souhaitez associer tel terme à tel objet.

Pour définir une variable, nous allons taper son nom, un signe égal puis sa valeur. Par exemple :

>>> laurene\_phone\_number = "07XXXXXXXX"

Pour afficher le contenu d’une variable, vous écrivez simplement son nom.

>>> laurene\_phone\_number

0759039203

Comment la modifier ? En la redéfinissant, tout simplement !

>>> laurene\_phone\_number = "07XXXXXXXX"

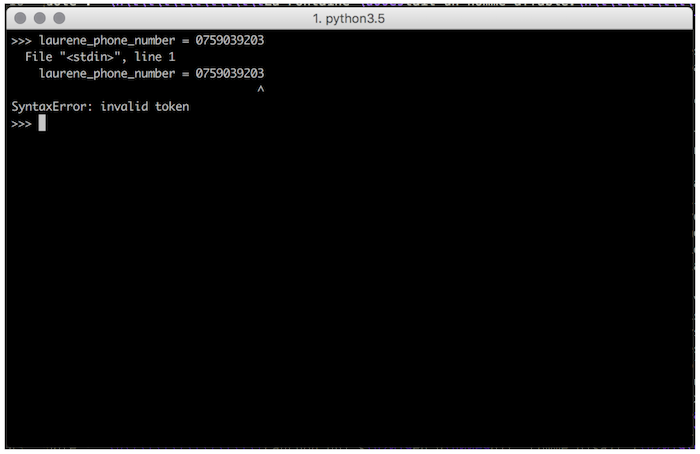
Et comment la supprimer ? En utilisant le mot-clé  del  (nous en parlerons plus tard):

>>> del(laurene\_phone\_number)

Nous allons définir une variable correspondant à une citation. À vous de jouer !

[Cliquez sur ce lien.](https://www.codevolve.com/api/v1/publishable_key/2A9CAA3419124E3E8C3F5AFCE5306292?content_id=6bd4e510-796a-47c3-87f1-b1f00e420a49)

Vous aurez certainement remarqué que j’ai mis des guillemets autour de la valeur associée. Si nous essayons sans, nous avons un beau message. Oh, la jolie SyntaxError !



### Les messages d'erreur

Python est apprécié des débutants notamment pour ses messages d’erreurs. D’autres programmes auraient affiché Error, et basta, mais Python aime ses développeurs et leur donne un peu plus de détails ! Attardons-nous sur la ligne la plus importante : la dernière.

Le message nous indique  SyntaxError . Bien, je suppose que j’ai dû faire une erreur de syntaxe quelque part… oublier des guillemets par exemple !

Prenez le temps de bien lire les messages d'erreur. Ils sont souvent très parlants !

### Types d'objets

En fait, je vous ai un peu menti (oui, je sais, ce n’est pas bien !). Ou plutôt je ne vous ai pas dit toute la vérité. Vous n’avez pas toujours à mettre de guillemets. C’est une révélation, je sais…

Seules les “chaînes de caractère” sont entourées de guillemets. Comme des citations ! D‘autres types d’informations vont être, eux, entourés de parenthèses ou de crochets.

D’ailleurs, en parlant d’information, rectifions un peu ce terme. Toute information est appelée “objet” en Python. Pourquoi ? Car nous pouvons manipuler les informations de la même manière que des objets : nous pouvons définir une variable, la modifier, la supprimer…

Pour l’instant ce concept peut sembler obscur, mais ne vous inquiétez pas : nous reviendrons dessus dans un autre cours ! Pour l’instant, rappelez-vous simplement que tout est objet en Python et que chacun de ces objets a des propriétés bien particulières.

Si vous désirez en savoir plus, je vous conseille de suivre le cours[**Découvrez la programmation orientée objet en Python**](https://openclassrooms.com/courses/decouvrez-la-programmation-orientee-objet-avec-python).

Alors, me direz-vous certainement, **pourquoi ne pas avoir le même type d’information partout** ? La réponse est assez simple : vous voudrez effectuer des actions différentes selon le type de votre objet. Vous êtes d'accord avec moi si je vous dis qu'on ne fait pas la même chose avec poignée de porte et un cure-dents ? Eh bien c'est parce que ce sont des objets de types différents.

Prenons un autre exemple.

Que ferez-vous avec un nombre ? Vous voudrez certainement l’additionner avec un autre nombre. Si vous tapez  4 + 4 , vous vous attendez à ce que l’interpréteur vous retourne  8 .

Et si vous voulez ajouter deux mots les uns à la suite des autres ? Vous utiliserez certainement le même opérateur, le + :  "tire-" + "bouchon"  et la console vous retournera…  "tire-bouchon" .

Vous ne voulez pas les additionner, comme avec un nombre, mais bien les **concaténer**. Python vous a compris. Voilà pourquoi vous avez des types d’objets différents !

Maintenant que vous avez créé votre premier objet, la variable ‘quote’ contenant une chaîne de caractères, faisons un tour d’horizon des autres types d’objets que l’on peut créer avec Python.

#### Nombres (integers and floats)

Il existe deux grands types de nombres en Python : les entiers (integers) et les décimaux (floats). Tous deux n’ont besoin ni de guillemets, de parenthèses ou de crochets. Ils se baladent tous nus, hu hu !

>>> my\_integer = 1

>>> my\_float = 1.3

Pas de virgules ! Les décimales se situent après un point.

#### Chaînes de caractères (strings)

Nous les avons déjà vues ! Il s’agit d’un ensemble de signes (lettres, chiffres, ponctuation, caractères spéciaux, …) délimités par des guillemets ouvrants et des guillemets fermants.

Vous pouvez utiliser des guillemets doubles ou simples, mais pensez à utiliser les mêmes au début et à la fin !

Vous pouvez également insérer des guillemets à l’intérieur des guillemets… à condition de les “échapper” ou de ne pas utiliser les mêmes ! Voici un exemple :

>>> quote = "San Antonio a dit : 'Ecoutez-moi, Monsieur Shakespeare: Nous avons beau être ou ne pas être, nous sommes !'"

Échapper un caractère : utilisation d’un antislash (“\”) avant un caractère pour qu’il soit interprété comme du texte :

>>> quote = "San Antonio a dit : \"Ecoutez-moi, Monsieur Shakespeare: Nous avons beau être ou ne pas être, nous sommes !\""

Python vous permet également d’écrire un texte sur plusieurs lignes en utilisant des triple guillemets :

>>> """Ma ligne commence ici...

... … et se continue ici

... """

#### Booléens (booleans)

Un booléen est une information Vraie ou Fausse. Ce type nous sera très utile un peu plus tard lorsque nous commencerons à comparer des valeurs entre elles. Sans surprise, vrai est  True  et faux est  False  (pensez aux majuscules !).

C’est un peu comparable à un quiz auquel on répondrait par Vrai ou par Faux. Exemple :

Un est inférieur à deux

>>> 1 < 2

True

Un est supérieur à 2

>>> 1 > 2

False

#### Listes

Une liste contient une succession de plusieurs objets. Comme une todo list !

On les reconnaît aisément car elles sont entourées de crochets et comprennent des valeurs séparées par des virgules. Comme ceci :

>>> characters = ["Alvin et les Chipmunks", "Babar", "Betty Boop", "Calimero", "Casper", "Le chat potté", "Kirikou"]

Pour créer une nouvelle liste, ouvrez des crochets, séparez les objets par des virgules et fermez les crochets.  
Vous pouvez y stocker des chiffres, des chaînes de caractère, d’autres listes… Un peu tout en fait !

quote = "Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !"

quote\_2 = "On doit pouvoir choisir entre s'écouter parler et se faire entendre."

all\_quotes = [quote, quote\_2]

Pour accéder à un élément stocké, vous écrivez le nom de la liste puis sa position entre crochets. Par exemple, si nous souhaitons accéder au premier élément de la liste characters :  characters[0] . Comme dans un tableur !

La position du premier élément d’une liste est le 0 !

Nous verrons plus tard, dans le chapitre consacré à cet effet, comment accéder plus précisément aux différents objets, les modifier, changer leur ordre ou les supprimer. Nous verrons aussi d’autres fonctionnalités assez sympathiques : mélanger les valeurs, mettre le premier mot d’une phrase en majuscules ou faire une division un peu spéciale… Mais je m’égare. Continuons !

#### Tuples

Les tuples servent à créer des structures dont le nombre d'éléments ne bouge pas. On dit qu’ils sont immuables car leur structure ne peut pas changer. On ne peut ajouter ou supprimer un élément.

Elles commencent par une parenthèse ouvrante, un ensemble d’objets séparés par des virgules et une parenthèse fermante.  
Exemple :

>>> ("Ecoutez-moi", "Monsieur Shakespeare", "nous avons beau être ou ne pas être", "nous sommes !")

Mais à quoi ça sert ?

À stocker des données dont la structure est importante et ne doit pas changer. Un tuple peut, par exemple, représenter les coordonnées géographiques d’une ville :

>>> paris = (48.856578, 2.351828)

#### Dictionnaires

Un dictionnaire est un ensemble de valeurs auxquelles vous pouvez accéder grâce d’autres objets (des chaînes de caractère par exemple !). Reprenons l’excellente définition de [Wikipedia](https://fr.wikipedia.org/wiki/Table_de_hachage" \t "_blank) :

Un dictionnaire est une structure de données qui permet d'associer des objets deux à deux -- exactement comme dans un dictionnaire en papier ! Les entrées du dictionnaire sont appelées des "clés", et à chaque clé est associée une valeur.

À la manière d’un vrai dictionnaire, vous accédez aux différentes définitions en regardant le mot qui vous intéresse. Si vous utilisez un dictionnaire de traduction anglais / français, vous vous attendez à ce que le mot "house" soit associé au mot "maison". En Python, vous le représenteriez ainsi :

>>> english\_french\_dict = {

... "un": "one",

... "deux": "two",

... "trois": "three"

... }

Nous pouvons utiliser un dictionnaire pour stocker toutes nos citations et tous nos personnages, par exemple !

Définir un dictionnaire commence par écrire une accolade ouvrante. Puis nous indiquons la première clé, deux points et la valeur associée, et enfin nous ajoutons une accolade fermante.

>>> my\_dictionary = {'key': 'value'}

Vous pouvez également stocker des éléments listes, comme ceci :

>>> program = {"quotes": ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."], "characters": ["alvin et les Chipmunks", "Babar", "betty boop", "calimero", "casper", "le chat potté", "Kirikou"]}

Accéder à une valeur se fait de la même manière que pour une liste mais au lieu d’indiquer sa position (0, 1, 2, …) vous utiliserez sa clé.

>>> my\_dictionary['key']

>>> program['quotes']

['Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !', 'On doit pouvoir choisir entre s'écouter parler et se faire entendre.']

Il n’y a pas d’ordre dans un dictionnaire. Vous ne pouvez pas accéder à la première valeur ou à la dernière en utilisant sa position. Dans ce cas il est préférable de créer une liste !

Comment connaitre le type d'un objet ? Je dois tout apprendre par coeur ?

Ces types reviennent très fréquemment et vous les manipulerez au quotidien donc vous les apprendrez bien vite, pas d'inquiétude ! Néanmoins il peut être intéressant, quand vous avez un doute, de demander à Python le type d'un objet. Vous utiliserez pour cela  type()  :

>>> type("hello!")

<class 'str'>

>>> type(34)

<class 'int'>

>>> type(1.3)

<class 'float'>

>>> type({"key": "value"}

<class 'dict'>

>>> type(["salut"])

<class 'list'>

### A vous de jouer !

[Cliquez sur ce lien.](https://www.codevolve.com/api/v1/publishable_key/2A9CAA3419124E3E8C3F5AFCE5306292?content_id=7e45bbe5-6550-48d0-89f2-4272391713a4)

Maintenant que vous avez une bonne vue globale des différents éléments que vous utiliserez dans la suite de ce cours, je suis sûre que vous rêvez de les manipuler tous ensemble, et de générer tout un tas de citations par exemple ! Hé bien, cher lecteur, cela tombe bien ! C’est justement ce que nous verrons dans le prochain chapitre !

# Découvrez les bases de Python

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Utiliser le vocabulaire de base de Python

### Question 1

**Pourquoi l'inventeur de Python a-t-il choisi ce nom ?**

* + 

Il aimait beaucoup les serpents.

* + 

Il aimait beaucoup le groupe d'humoristes Monty Python.

### Question 2

**Python est très utilisé dans les domaines suivants :**

*Attention, plusieurs réponses sont possibles.*

* + 

Big data

* + 

Application mobile pour iPhone (iOS)

* + 

Web

* + 

Robotique

### Question 3

**Pour installer Python, vous pouvez :**

*Attention, plusieurs réponses sont possibles.*

* + 

Télécharger un dossier sur [www.python.org](http://www.python.org/) puis suivre les étapes.

* + 

Il n'y a rien à faire, la dernière version de Python est installée par défaut sur tous les ordinateurs.

* + 

Ecrire une ligne de commande dans la console (Mac ou Linux).

### Question 4

**Python 3 est rétrocompatible, c'est à dire que vous pouvez travailler sur des fichiers utilisant la version 2 de Python si vous avez installé la version 3.**

* + 

Vrai

* + 

Faux

### Question 5

**La console (pas l'interpréteur) sert à...**

*Attention, plusieurs réponses sont possibles.*

* + 

Naviguer dans un ordinateur en ligne de commande.

* + 

Lancer des programmes en ligne de commande.

* + 

Coder en Python (vous pouvez écrire  7 + 7  et la console affichera  14 )

### Question 6

**L'interpréteur Python (pas la console) sert à...**

* + 

Ecrire du code Python.

* + 

Se déplacer dans l'ordinateur en langage Bash ou DOS (je peux écrire  ls ).

### Question 7

**Une chaine de caractère est :**

* + 

Un ensemble de chiffres et de lettres encadrés par des guillemets simples ou doubles. Exemple :  "Ceci est une chaîne"

* + 

Un ensemble de caractères spéciaux sans guillemets. Exemple :  &()

* + 

Un ensemble de chiffres et de lettres à la suite, sans majuscule, espace ou caractère spécial. Exemple :  ceci\_est\_une\_chaine

### Question 8

**Une variable est :**

* + 

Un ensemble de chiffres et de lettres encadrés par des guillemets simples ou doubles. Exemple : "Ceci est une variable"

* + 

Un ensemble de chiffres et de lettres à la suite, sans majuscule, espace ou caractère spécial, qui sert à sauvegarder une valeur pour la réutiliser ultérieurement. Exemple :  ma\_variable = "Une valeur"

### Question 9

**Une liste organise un ensemble de valeurs. On peut retrouver ces valeurs grâce à leur position (leur index) dans la liste. Exemple :  ma\_liste = ["chien", "chat"]**

* + 

Vrai

* + 

Faux

### Question 10

**Un dictionnaire organise un ensemble de valeurs. On peut les retrouver grâce à leur clé. Exemple :  mon\_dict = {"one": "un", "two": "deux"}**

* + 

Vrai

* + 

Faux

## Créez votre premier script

Avant de continuer notre programme, j’ai bien envie de ne plus utiliser l’interpréteur.

🙀 Attends, tu nous as fait tout un pensum sur l’utilisation de l’interpréteur et maintenant tu veux l’abandonner ?

😀

Cher lecteur, vous avez raison ! Je ne vais pas l’abandonner : je vais plutôt l’utiliser en complément d’un éditeur de texte.

Vous avez certainement remarqué qu’il devient de plus en plus compliqué de travailler avec l’interpréteur. Quand vous le quittez, tout s’efface. Vous devez vous souvenir du code créé. De plus, vous ne pouvez pas modifier facilement une variable.

Peut-être avez-vous déjà commencé à créer un fichier externe pour prendre des notes et conserver une trace des différentes commandes ? C’est exactement ce que nous allons faire : créer un fichier qui conservera nos commandes, une par une, puis lancer le fichier avec python. Au lieu d’entrer, à la main, chaque commande, vous lancerez le programme. C’est plus rapide, non ?!

### Créer un fichier externe

#### Sous Linux et Mac

Travailler avec un fichier externe se fait en deux étapes : vous créez le document puis vous l'exécutez.

##### Utilisation d'un document externe

Commencez par créer un dossier concernant ce cours (évitez les noms à rallonge, même si  supercourstropinteressantsurpython.py  me flattera beaucoup !). A l’intérieur, créez un nouveau fichier  san\_antonio.py .

C'est bien, vous avez votre premier script ! A présent, comment l'ouvrir et le modifier ? En utilisant un éditeur de texte, un petit logiciel conçu pour écrire du code. Il en existe à foison mais je vous conseille [Sublime Text](https://www.sublimetext.com/3).

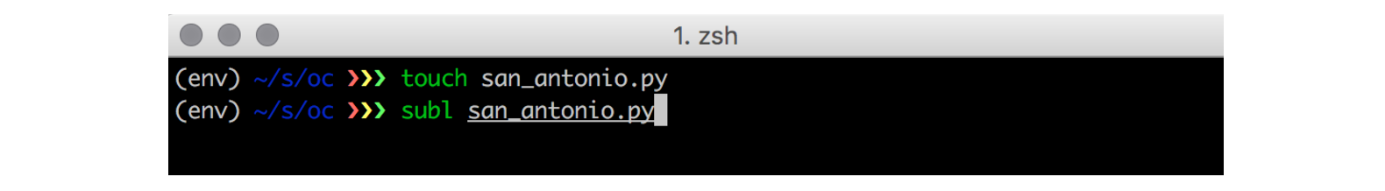
Vous pouvez réaliser ces actions en utilisant votre souris mais pourquoi ne pas le faire en ligne de commande ? En tant que véritable développeur·se en herbe, je suis sûre que vous avez très envie de savoir comment !

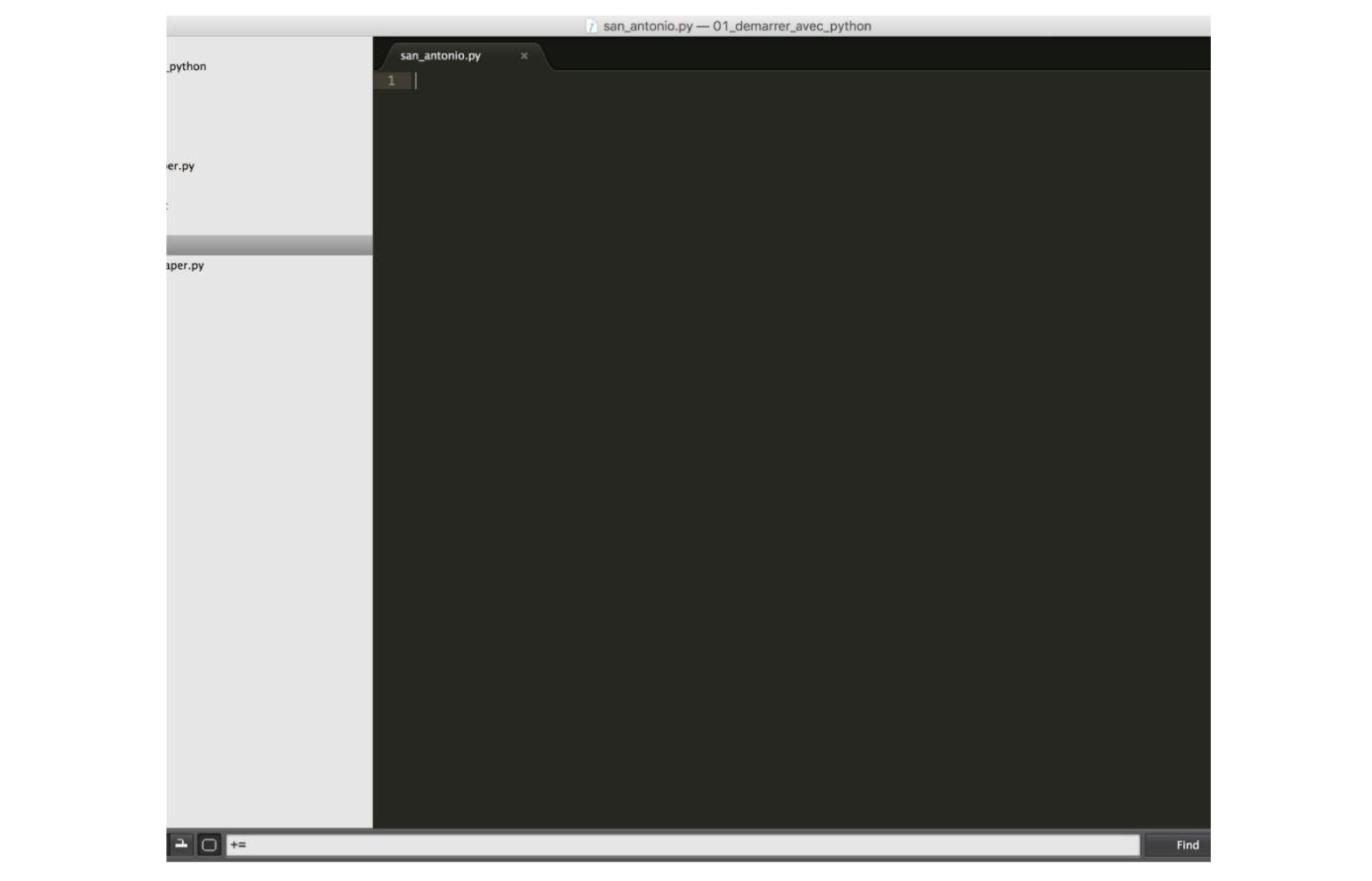
Ouvrez votre console (ou quittez l'interpréteur) et écrivez ces commandes :

-  cd dossierdetravail  : déplacez-vous dans votre dossier de travail. Si votre dossier est enregistré dans votre bureau, tapez : cd  desktop/dossierdetravail .

-  touch nomdevotrefichier.py  : créer un nouveau fichier à l'endroit où vous êtes.

-  subl nomdevotrefichier.py  : ouvrir le fichier avec Sublime Text pour pouvoir le modifier.

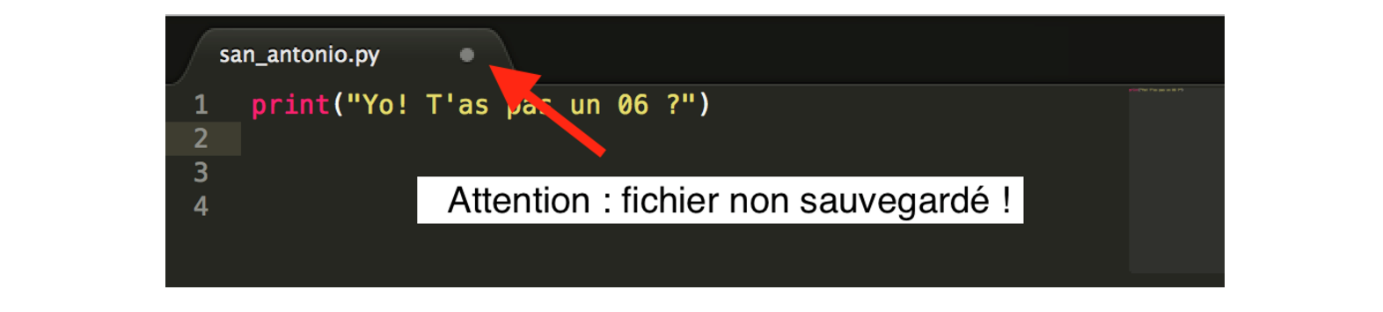




Ajoutez la commande que vous souhaitez exécuter. De mon côté j’y écris  print("Yo ! T’as pas un 06 ?") pour afficher quelque chose sur la console. Nous n’avons pas encore vu cette commande, donc pas de panique. Je l’utilise simplement en test.

Puis sauvegardez.

Sur Sublime Text, un petit rond apparaît à droite de l’onglet si votre fichier n’est pas sauvegardé. Une croix le remplace quand le fichier est bien sauvé ! Pratique !

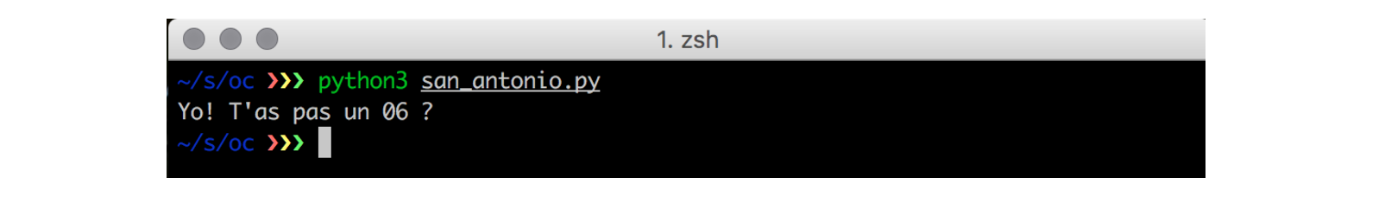


##### Exécuter le fichier avec Python

Enfin, lancez python avec le nom du programme juste après, comme ceci :

python3 san\_antonio.py

Vous devriez voir apparaître votre message !



Si vous avez travaillé sur l’interpréteur précédemment, n’oubliez pas de le quitter avant d’exécuter votre programme ! Pour rappel, tapez  exit()   pour quitter.

#### Sous Windows

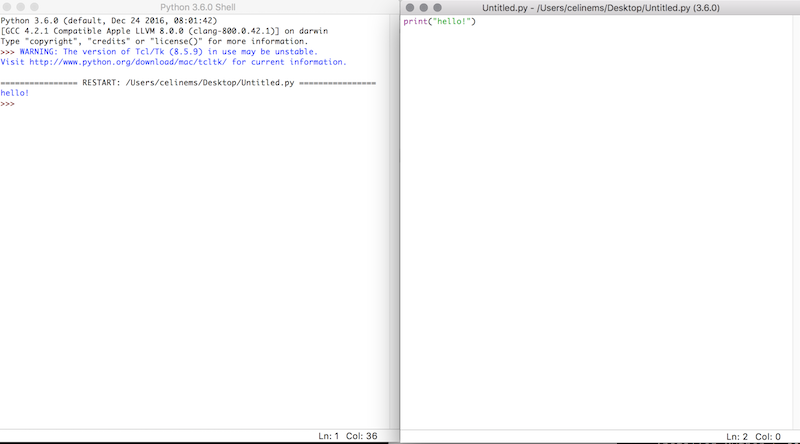
Utilisez IDLE, toujours IDLE, encore IDLE ! ;-)

#### Travailler avec un document externe

Dans le menu, cliquez sur File > New File. Puis sauvegardez le fichier en tapant ctrl + s.  Choisissez bien l'endroit où vous l'enregistrer pour pouvoir le retrouver plus tard !

##### Exécuter le fichier avec Python

Pour lancer votre programme, cliquez sur Run > Run Module. IDLE va exécuter votre programme dans l'interpréteur Python qui était déjà ouvert.



Renouvelez cette dernière opération à chaque fois que vous souhaitez exécuter votre programme !

### Avant de continuer...

Je prends quelques minutes pour réécrire, dans mon tout nouveau fichier, le code dont j’ai besoin pour mon programme.

quotes = [

"Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !",

"On doit pouvoir choisir entre s'écouter parler et se faire entendre."

]

characters = [

"alvin et les Chipmunks",

"Babar",

"betty boop",

"calimero",

"casper",

"le chat potté",

"Kirikou"

]

Attention : si vous lancez le script en l'état, vous pouvez rencontrer une erreur à cause des accents. Hé oui, notre belle langue de Molière cause parfois des soucis.

Afin d'éviter cela, ajoutez cette ligne en haut de votre fichier :

# -\*- coding: utf8 -\*-

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub en cliquant [sur ce lien](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P2C1).

## Comparez des valeurs avec les opérateurs

Intéressons-nous maintenant à l’interaction avec l’utilisateur. La première phrase qui s’affichera sera une citation au hasard. Puis, nous proposerons deux alternatives :

1. Si l’utilisateur tape “entrée”, une nouvelle citation apparaît.
2. S’il tape “B”, le programme se ferme.

Nous allons commencer par écrire du pseudo-code, c’est-à-dire écrire ce que nous voulons que le programme fasse avec nos propres mots. Il s’agit d’une pratique très courante chez les développeurs.

Tout mon code est écrit en anglais. Non pas que je souhaite mettre en avant mes incroyables talents linguistiques, mais plutôt car la langue du développement est l’anglais. Qui sait ce que deviendra votre projet demain ? Vous pouvez choisir de le rendre disponible en open source pour que chacun puisse y contribuer, y compris des non-francophones. Ou bien vous pouvez le vendre. En plus les accents français ne sont pas acceptés dans les noms de variables. Bref, coder en anglais est une bonne pratique. Rassurez-vous : il s’agit de mots simples et vous comprendrez tout.

# Show random quote

# If user\_answer is ‘B’:

# - leave the program

# Else :

# - show another quote

Commençons par découvrir les opérateurs qui nous permettront de comparer user\_answer et ‘B’.

### Opérateurs de comparaison

La logique voudrait qu’on utilise le signe  =  pour comparer deux valeurs. Mais, si vous vous souvenez bien, ce signe est déjà utilisé pour assigner une valeur à une variable. Nous ne pouvons donc pas l’utiliser pour comparer !

C’est pourquoi nous doublons le signe égal par un autre égal pour signifier la **comparaison**, comme ceci :**==**.

Les opérateurs de comparaison renvoient un booléen ( True  ou  False ) car vous posez une question fermée : c’est vrai ou ça ne l’est pas !

Egal

>>> 1 == 1

True

Différent

>>> 1 != 2

True

Supérieur

>>> 1 > 2

False

Supérieur ou égal

>>> 1 >= 2

False

Inférieur

>>> 1 < 2

True

Inférieur ou égal

>>> 1 <= 2

True

Amusez-vous à changer les valeurs et lancez le programme pour afficher les résultats !

### Les opérateurs mathématiques

Pas de panique ! Ce sont des maths de primaire. 😉

Addition

>>> 1 + 1

2

Soustraction

>>> 1 - 1

0

Multiplication

>>> 1 \* 2

2

Division

>>> 1 / 1

1.0

Modulo

>>> 1 % 2

1

Le modulo est un opérateur un peu spécial mais très pratique. Trop pour que nous le laissions seul dans les abîmes de l’oubli !

Vous vous souvenez du temps où vous faisiez des divisions à la main ? Parfois, ça ne tombait pas pile-poil sur un nombre entier, il y avait aussi un reste. Le reste de la division, c'est le modulo !

Si vous divisez 5 par 2, vous obtiendrez 2,5, voyez plutôt : 5 / 2 = 2,5.

Pour avoir le modulo, regardez combien de fois vous pouvez mettre "2" dans 5. Vous pouvez mettre deux fois "2" dans le nombre "5", et il vous reste… 1. Le 1 est le modulo !

Je prends un autre exemple. Imaginons que j'aie 14 euros, et que je veuille acheter autant de kebabs que possible à 5 euros (j'ai faim !). Si je tends tout mon argent au vendeur, j'aurai 2 kebabs et le vendeur me rendra 4 euros.

>>> 14 % 5

4

Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P2C2).

## Ajoutez un peu de logique avec les conditions

Nous savons comment comparer deux valeurs, mais nous ne savons pas encore comment écrire des conditions.

Les conditions ont le même sens en informatique que dans le langage courant. Si (événement), dans ce cas (action). Exemple : si j’ai 4 euros, je peux m’acheter le dernier Picsou Magazine. Sinon, il faut que j’aille retirer des sous.

Souvenons-nous de notre pseudo-code (qui s’est un peu précisé) :

# Show random quote

# If user\_answer == "B":

# - leave the program

# Else:

# - show another quote

### La condition "Si"

Python étant bien fait, écrire une condition est aussi simple qu’écrire en anglais :

if user\_answer == "B":

# leave the program

Vous commencez par écrire if, puis vous indiquez la condition à remplir et terminez la ligne par deux points (sans espace avant !).

Vous indiquez les actions à effectuer juste en-dessous. Afin de différencier ces actions, qui sont à l’intérieur d’une condition, du reste de votre programme, vous ajoutez quatre espaces au-début de la ligne. Nous appelons cela l’indentation et Python est très strict sur le sujet !

L’importance de l’indentation est souvent sous-estimée, pourtant elle est à la source des principales erreurs de débutant·e·s ! Prenez le temps de vous relire et de vérifier qu’il y a bien quatre espaces, et non trois ou cinq !

### Le cas échéant...

Comment définir ce que se passe si la condition n’est pas remplie ? En utilisant else:

if user\_answer == "B":

# leave the program

else:

# show another quote

### Intégrer une condition à la suite d'une autre condition

Vous pouvez également proposer d’autres choix grâce au mot-clé  elif  (contraction de else et if):

if user\_answer == "B":

# leave the program

elif user\_answer == "C":

print("C pas la bonne réponse ! Et G pas d’humour, je C...")

else:

# show another quote

Comment quitter le programme ? En utilisant le mot-clé  pass  :

if user\_answer == "B":

pass

elif user\_answer == "C":

print("C pas la bonne réponse ! Et G pas d’humour, je C...")

else:

# show another quote

pass  ne sert pas vraiment à quitter un programme. Nous l'utilisons quand aucune action n'est requise.  Par défaut, si vous n'indiquez rien, Python considèrera qu'il y a une erreur. En effet, pourquoi créer une fonction vide ? Il considèrera que vous avez dû oublier un élément. En écrivant  pass , vous lui dites : je te demande de ne rien faire. Et Python passera son chemin, tout simplement.

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P2C3).

## Structurez votre programme en utilisant les fonctions

Bien bien, notre programme s’étoffe ! Intéressons-nous maintenant à la première phrase :

# Show random quote

Nous avons déjà une liste composée de deux citations :

quotes = ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."]

Afin d’afficher dans la console une citation au hasard, nous devons avant choisir une position aléatoirement dans la liste. Il nous faut donc commencer par chercher comment Python peut nous renvoyer un nombre compris entre une valeur minimale (0, le début de la liste) et une valeur maximale (le nombre total d’items - 1).

Je vous propose de diviser cette étape en plusieurs sous-étapes :

# Show random quote:

# get a random number

# get a quote from a list

# show the quote in the interpreter

Nous avons ainsi regroupé plusieurs sous-actions autour d’une grande :  show\_random\_quote . Nous allons utiliser cette grande action à plusieurs endroits : au démarrage de notre programme, puis quand l’utilisateur appuiera sur “entrée”.

L’idéal serait donc de pouvoir stocker cet ensemble d’action sous le nom d’une seule variable : show\_random\_quote.

Nous n’allons pas utiliser les variables pour le faire mais plutôt une fonction.

### Créez une nouvelle fonction

Une fonction est un ensemble de commandes regroupées sous un seul nom unique. Pour définir une fonction, vous utilisez le mot-clé “def”, suivi d’un nom, de parenthèses et de deux points. Puis vous indentez tout ce qui se trouve à l’intérieur ! Exemple :

def get\_random\_quote() :

# get a random number

# get a quote from an array

# show the quote in the interpreter

pass

Nous retrouvons le mot-clé  pass ! Rappelez-vous : Python n’aime pas le vide. Si vous souhaitez exécuter une fonction qui ne fait rien, vous devez tout de même indiquer à Python… de ne rien faire :) Découvrez-en plus dans la [**documentation**](https://docs.python.org/3.6/tutorial/controlflow.html#pass-statements).

Pour exécuter une fonction, écrivez son nom (avec ses parenthèses !) :

get\_random\_quote()

### Paramètres

Arrivé·e à ce point du cours, je suis certaine qu’une question vous intrigue, que dis-je, vous obsède ! N’en perdez pas le sommeil : je vais vous expliquer à quoi servent les parenthèses d’une fonction !

Les paramètres sont des variables que vous fournissez à votre fonction. En effet, l’intérêt d’une fonction est de pouvoir être réutilisée dans des contextes différents. Par exemple, notre fonction actuelle affichera une citation au hasard, mais nous voudrons certainement en faire de même pour les personnages. Allons-nous copier / coller la fonction et changer son nom ? Non. Nous allons simplement passer la liste en paramètre. Démonstration :

quotes = ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."]

def get\_random\_item\_in(my\_list):

# TODO: get a random item

my\_list[0] # get an item from a list. For the moment, just get the first one.

# TODO: show the quote

get\_random\_item\_in(quotes)

### Utilisez les fonctions standards de Python

Python contient de nombreuses fonctions qui rendent la vie plus simple. Vous voulez connaître le type d’un objet ? Utilisez  type(MyObject) . Vous voulez afficher une valeur ? Utilisez  print(my\_value) .

Une des fonctions par défaut très utiles est  input(mon\_message) . Elle vous permet de “demander” son avis à l’utilisateur ! Le message affiché est passé en paramètre à la fonction.

Exemple :

input('Tapez entrée pour connaître une autre citation ou B pour quitter le programme.')

Quand elle est exécutée, elle renvoie la valeur saisie par l’utilisateur. Pour l’utiliser vous devez l’assigner à une variable.

user\_answer = input('Tapez entrée pour découvrir une autre citation ou B pour quitter le programme.')

### Renvoyez une valeur

Une fonction exécute un ensemble d’action mais, par défaut, elle ne renvoie pas d’information spécifique. Dans le cas de notre fonction, par exemple, Python va effectivement trouver un nombre au hasard, trouver une citation, l’afficher dans l’interpréteur mais ne nous donnera aucun message nous indiquant qu’il a bien tout fait !

Pour que votre fonction renvoie une certaine valeur, utilisez le mot-clé “return” :

def get\_random\_item\_in(my\_list):

# TODO: get a random number

item = my\_list[0] # get a quote from a list

print(item) # show the quote in the interpreter

return "program is over" # returned value

print(get\_random\_item\_in(quotes))

Le résultat dans la console :

Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !

program is over

Vous aurez certainement remarqué ce dièse qui se balade et qui est suivi d’un mot ou d’une phrase. Python vous permet de laisser des commentaires dans votre code, c’est-à-dire des annotations qui vous serviront à mieux comprendre ce que vous écrivez. Vous en trouverez beaucoup dans mon code. Abusez-en, cela permet aux lecteurs de mieux comprendre votre logique !

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P2C4).

## Répétez une action grâce aux boucles

Retournons à notre condition principale. J’ai une bonne nouvelle : saviez-vous que nous pouvions la simplifier en utilisant une boucle ? Hé oui !

Une boucle (loop en anglais) est une action qui se répète automatiquement un certain nombre de fois.

Prenons un exemple de la vie de tous les jours. Vous organisez une soirée ‘Pirates vs Mousquetaires’ (pourquoi pas) et vous envoyez une invitation à 300 personnes. La première manière, très fastidieuse, est d’envoyer un mail individuel à toutes ces personnes. Etant assez feignant·e, vous décidez d’envoyer un mail groupé.

Le programme va faire le travail à votre place. Il va envoyer un mail à chaque personne de votre liste jusqu’à arriver à la fin ! Il entre ainsi dans une boucle qui commence par la première personne de la liste et qui finit par la dernière. Chaque itération correspond à l’envoi d’un mail.

Python nous offre deux boucles principales :  while  et  for . Commençons par la boucle  while  dans notre programme.

### La boucle while

While est la traduction de “tant que...”. Concrètement, la boucle s’exécutera tant qu’une condition est remplie (donc tant qu’elle renverra la valeur True). Reprenons le cas de notre programme de citations: nous affichons une citation tant que la réponse de l’utilisateur n’est pas B.

while user\_answer != ‘B’:

print(get\_random\_item\_in(program['quotes']))

Si nous reprenons tout notre code…

quotes = ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."]

characters = ["alvin et les Chipmunks", "Babar", "betty boop", "calimero", "casper", "le chat potté", "Kirikou"]

def get\_random\_item\_in(my\_list):

# get a random number

item = my\_list[0] # get a quote from a list

return item # return the item

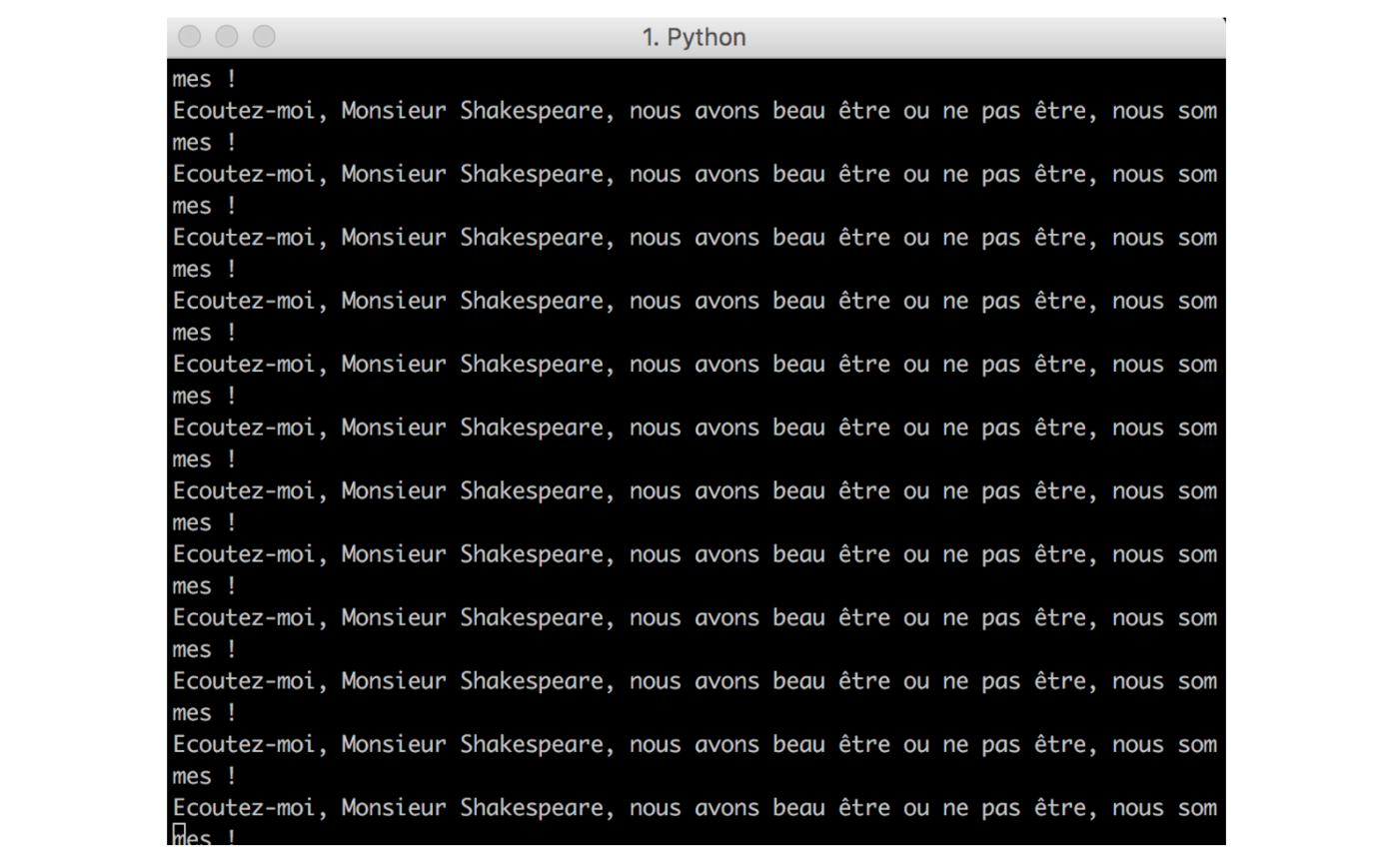
user\_answer = "A"

while user\_answer != "B":

print(get\_random\_item\_in(quotes))

Voilà. Nous avons réduit nos 6 lignes de code à 2 ! (et supprimé la petite blague, mais personne ne va s’en plaindre…)

Si vous exécutez le programme, vous vous apercevrez très vite que l’interpréteur vous affiche, en boucle, le même message ! Encore et encore, il ne s’arrête pas ! Avez-vous deviné pourquoi ?



Hé oui : nous sommes dans une **boucle infinie** ! La valeur de  user\_answer  étant  A , et la boucle ne se terminant que si  user\_answer  est égal à  B , le programme ne se termine jamais… 😱

En attendant de voir comment interagir avec l’utilisateur, définissons la variable à la fin de notre programme pour fermer la boucle.

quotes = ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."]

characters = ["alvin et les Chipmunks", "Babar", "betty boop", "calimero", "casper", "le chat potté", "Kirikou"]

def get\_random\_item\_in(my\_list):

# get a random number

item = my\_list[0] # get a quote from a list

return item # return the item

user\_answer = "A"

while user\_answer != "B":

print(get\_random\_item\_in(quotes))

user\_answer = "B"

### La boucle for

Maintenant imaginons que nous voulons utiliser une boucle pour mettre en majuscule chaque nom de personnage. La boucle while n’est pas vraiment utilisable car n’avons pas de condition. Nous voulons juste effectuer une action en boucle.  
Dans ce cas, la boucle  for  est plus adaptée. Voici sa syntaxe :

for item in a\_list:

# do something

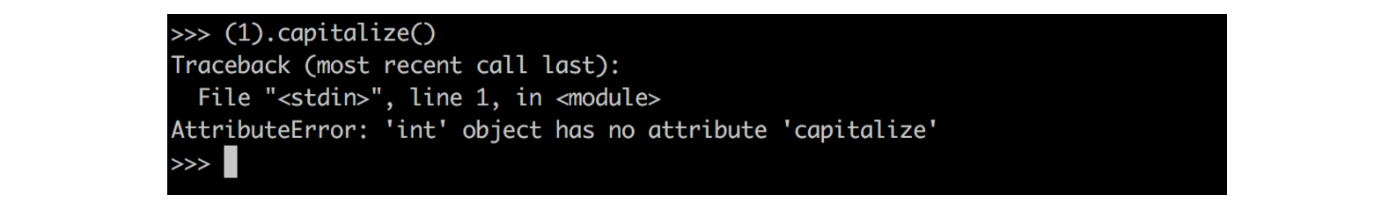
Nous pourrions traduire cela en français : “pour chaque item dans une liste: fais ça”

Nous allons pouvoir agir sur chaque item ! Par exemple :

for quote in quotes:

quote.capitalize()

Nous venons tout juste de voir la méthode  capitalize()  pour mettre en majuscule le premier mot d’une chaîne de caractères. Si vous essayez d’utiliser cette même méthode avec un nombre, Python vous affichera une belle erreur :



Pourquoi ? Cela paraît logique : on ne peut mettre en majuscule que des mots… Il y a donc des fonctions réservées à certains types d’objets. C’est ce que nous appelons des **méthodes**. Découvrons-les en détail dans les prochains chapitres !

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P3C4).

# Les fondations

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Poser les fondations d'un programme Python

### Question 1

**Pour lancer l'interpréteur Python, j'écris dans la console le mot suivant :  python (oupython3, selon votre système d'exploitation).**

* + 

Vrai

* + 

Faux

### Question 2

**Pour lancer un programme  mon\_fichier.py  qui utilise Python, j'utilise la commande suivante :**

* + 

 python mon\_fichier.py -o python

* + 

 python mon\_fichier.py

* + 

 python

* + 

 python5

### Question 3

**Comment comparer si deux valeurs sont différentes ?**

* + 

 a != b

* + 

 a == b

### Question 4

**Comment comparer que deux valeurs sont égales ?**

* + 

 a = b

* + 

 a == b

### Question 5

**Modulo est représenté par un signe pourcentage (%). Il sert à...**

* + 

Changer le contenu d'une variable.

* + 

Enregistrer des échantillons de notes d'instruments de musique.

* + 

Réaliser une division et renvoyer l'entier restant.

### Question 6

**Je souhaite afficher le message "I'm aliiiiiiiiiiiive !" si le contenu de la variable est "Sia". Comment faire ?**

* + 
  + if only singer = "Sia":
  + print("I'm aliiiiiiiiiiiive!")

* + 
  + if singer == "Sia":
  + print("I'm aliiiiiiiiiiiive!")

* + 
  + only singer = "Sia":
  + print("I'm aliiiiiiiiiiiive!")

* + 
  + if singer == "Sia"
  + print("I'm aliiiiiiiiiiiive!")

### Question 7

**Le mot-clé  pass  est notamment utilisé :**

* + 

Pour vérifier que deux valeurs sont identiques.

* + 

à l'intérieur d'une fonction pour que celle-ci s'exécute même si elle ne contient aucune "action".

### Question 8

**Pour créer une nouvelle fonction "message" qui prend deux paramètres ("recipient" et "sender"), j'écris :**

* + 
  + def message(recipient, sender):
  + pass

* + 
  + func message(recipient, sender):
  + pass

* + 
  + function message(recipient, sender):

* + 
  + def message recipient, sender
  + pass

### Question 9

**Pour utiliser une fonction dans votre code, vous suivez ces deux étapes :**

* + 

Exploration et indexation

* + 

Définition et exécution

* + 

Vérification et éxécution

### Question 10

**La fonction  print()  sert notamment à...**

* + 

Envoyer votre code chez l'imprimeur.

* + 

Afficher le message passé entre paramètres dans la console.

* + 

Concaténer deux valeurs.

## Modifiez des chaînes de caractères

Chaque type d’objet dispose de différentes méthodes qui nous permettent d’interagir avec lui. Une méthode est une fonction associée à un type d’objet bien précis. En d’autres termes, une action que seul ce type d’objet peut faire.

La documentation officielle de Python concernant les chaînes est assez longue. Regardons ensemble quelles méthodes sont les plus utiles.

Split a word in several elements and create a new list out of it.

>>> “hello world!”.split()

['hello', 'world!']

Remove all white spaces at the beginning and the end of a string

>>> “       hello world!      “.strip()

“hello world!”

First letter in the first word in capital letters

>>> "hello world!".capitalize()

"Hello world!"

Every word in upper case

>>> "hello world!".upper()

"HELLO WORLD!"

Every word in lower case

>>> "HELLO WORLD".lower()

"hello world!"

Une dernière méthode, très utilisée, vous permet de remplacer des valeurs à l’intérieur d’une chaîne à la manière d’un texte à trous. Il s’agit de la méthode  format() . Tiens, c’est exactement ce que nous souhaitons faire dans notre programme de citations !

Nous utilisons la méthode  format()  de la manière suivante : chaque variable que nous souhaitons utiliser est entourée d’accolades puis définie en paramètres. Exemple :

"{character} a dit : {quote}".format(character="Babar", quote="Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

Sachez que vous pouvez également ne pas définir de variable et vous contenter de donner les valeurs à remplacer dans le même ordre que la phrase. Exemple :

"{} a dit : {}".format("Babar", "Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

Cette dernière forme, plus courte, est également plus commune.

Découvrez-en plus dans l’[excellente documentation](https://docs.python.org/3.5/library/stdtypes.html#text-sequence-type-str) !

### A vous de jouer !

def create\_message(character, quote):

    print('{} a dit : {}'.format(character, quote))

create\_message("Lapalisse","Tout ce qui est rond n'est point carré")

## Modifiez des nombres

Il n’existe pas beaucoup de méthodes sur les nombres (entiers comme décimaux). La plus utile,  is\_integer  , vous permet de vérifier si un nombre est un entier ou un décimal :

Is an integer?

>>> (2.5).is\_integer()

False

Vous remarquerez certainement que j’ai indiqué le nombre entre parenthèses… mais pourquoi donc ? Tout simplement pour que Python ne l’interprète pas comme une méthode (qui s’appelle sur un objet en utilisant un point) mais bien comme un décimal.

Comme dans les opérations mathématiques que nous avons tous faites à l'école, les opérations sont réalisées dans l’ordre. La priorité va aux multiplications, aux divisions et aux opérations entre parenthèses.

>>> 1 + 4 \* 2

9

>>> (1 + 4) \* 2

10

Attention : il est impossible de diviser par zéro !

Découvrez d’autres méthodes dans l’excellente [documentation sur les nombres](https://docs.python.org/3.5/library/stdtypes.html#numeric-types-int-float-complex) !

## Rangez des données dans des listes

Nous avons vu précédemment comment accéder à la première valeur d’une liste ou d’un tuple :  my\_list[0] . Mais comment accéder aux suivantes ? Et à la dernière ?

Chaque élément est associé à un index, c’est à dire à un nombre qui indique sa position dans la liste.

Le premier index d’une liste est toujours 0 !

Pour accéder aux différents éléments, indiquez son index entre crochets. Comme ceci :

Accéder au 4e élément d’une liste

>>> my\_list[3]

Python vous renvoie alors la valeur stockée à cet index, et non sa position ! Si vous souhaitez connaitre l’index d’un élément, utilisez la méthode  .index(element\_to\_find)

>>> characters.index("Babar")

1

Ajouter un élément à la fin de la liste

>>> characters.append("Mowgli")

Ajouter un élément à un certain index. Le premier argument est l’index, le second la valeur à insérer.

>>> characters.insert(4, "Balou")

Modifier un élément : y accéder grâce à son index et lui donner une nouvelle valeur

>>> characters[1] = "La Fée Clochette"

Supprimer le dernier élément de la liste et ***renvoyer sa valeur***. De manière optionnelle, vous pouvez passer en paramètre l'index de l'élément à supprimer.

>>> characters.pop() # removes the last item and returns it

'Mowgli'

>>> characters.pop(4) # removes the item at index 4 and returns it

'Balou'

Supprimer un élément de la liste ***sans renvoyer sa valeur.***C'est le premier élément de la liste qui correspond à cette valeur qui sera supprimé. Vous devez toujours passer une valeur en paramètre.

>>> characters.remove("Mowgli")

En savoir plus sur les différences entre  del() ,  .pop()  et  .remove()  : **[Difference between del, remove and pop on lists](https://stackoverflow.com/questions/11520492/difference-between-del-remove-and-pop-on-lists" \t "_blank)** 🇬🇧.

Connaitre le nombre d’éléments dans une liste

>>> len(characters)

Accéder au dernier élément d’une liste

>>> characters[-1]

Découvrez bien plus de méthodes dans l’[excellente documentation](https://docs.python.org/3.5/tutorial/datastructures.html) de Python.

### A vous de jouer !

characters = ['Dumbo', 'Moogli', 'Oscar', 'Pocahontas','Jafar']

characters.remove('Oscar')

print(characters)

characters.append('Picsou')

print(characters)

characters[1]='Balou'

print(characters)

## Organisez vos données dans un dictionnaire

Nous avons déjà vu comment accéder à la valeur d’un élément stocké dans un dictionnaire : grâce à sa clé.

Reprenons un exemple :

>>> program = {"quotes": ["Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !", "On doit pouvoir choisir entre s'écouter parler et se faire entendre."], "characters": ["alvin et les Chipmunks", "Babar", "betty boop", "balimero", "casper", "le chat potté", "Kirikou"]}

>>> program["characters"]

[“alvin et les Chipmunks”, “Babar”, “betty boop”, “balimero”, “casper”, “le chat potté”, “Kirikou”]

La première question que je me pose est la suivante :

Comment accéder aux éléments d’une liste qui est elle-même dans un dictionnaire ?

En ajoutant l’index de l’élément souhaité à la suite, entre crochets !

>>> program["characters"][0]

"alvin et les Chipmunks"

Regardons maintenant les principales méthodes que nous pouvons utiliser avec un dictionnaire…

Remplacer ou ajouter une valeur : même méthode que pour une liste.

>>> program["characters"] = "Un nouveau nom"

Mettre à jour ou ajouter plusieurs valeurs en même temps

>>> program.update({"characters" : ["Alvin", "Père Noël"], "quotes": ["Une citation unique qui sera sauvegardée"]})

Supprimer une clé et renvoyer sa valeur. Vous pouvez utiliser cette même méthode sur une liste !

>>> program.pop("quotes")

"quotes"

Découvrez-en plus dans la [documentation en ligne](https://docs.python.org/3.5/library/stdtypes.html#mapping-types-dict) !

### A vous de jouer !

friends = {"Doctor Who": "Le Tardis", "Frodon": "Terre du Milieu", "Red": "Storybrooke" }

### Retour à notre programme

Notre programme s’étoffe ! Youhou ! Il nous manque pourtant 3 grandes fonctionnalités :

* Afficher une citation au hasard quand on lance le programme.
* BONUS : Stocker nos citations et nos personnages dans un fichier externe.
* BONUS : Coder un petit robot qui va parcourir le Web à la recherche de citations et de personnages puis les stocker dans un fichier sur notre ordinateur.

Voyons, dans le prochain chapitre, comment afficher une citation au hasard.

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P3C4).

Si vous êtes**Data Analyst** ou **Data Scientist**, vous devrez maîtriser les librairies Python pour la Data Science. Suivez ce cours : [**Découvrez les librairies Python pour la Data Science**](https://openclassrooms.com/fr/courses/4452741-decouvrez-les-librairies-python-pour-la-data-science)

# Manipulez des objets

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Manipuler des objets en Python

### Question 1

**Pour transformer une chaîne de caractères en liste, j'utilise la méthode suivante :**

* + 

 .list()

* + 

 .to\_list()

* + 

 .split()

### Question 2

**Pour supprimer les espaces blancs au début et à la fin d'une chaîne de caractères, j'utilise la méthode suivante :**

* + 

 .without\_blanks()

* + 

  .strip()

* + 

  .remove\_blanks()

### Question 3

**La méthode format s'utilise de la manière suivante (2 bonnes réponses) :**

*Attention, plusieurs réponses sont possibles.*

* + 
  + "{character} a dit : {quote}".format(character="Babar", quote="Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

* + 
  + "(character) a dit : (quote)".format(character="Babar", quote="Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

* + 
  + "{} a dit : {}".format("Babar", "Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

* + 

"[] a dit : []".format("Babar", "Tout n'est pas cirrhose dans la vie, comme dit l'alcoolique.")

### Question 4

**Pourquoi met-on un nombre entre parenthèses quand nous voulons exécuter une méthode sur cet objet ? Exemple :  (1.5).is\_integer()**

* + 

C'est purement décoratif, je peux exécuter cette méthode sans les parenthèses.

* + 

Sinon il y aurait trop d'ambiguïté : Python ne saurait pas si la méthode fait partie du nombre (pour former un décimal) ou s'il s'agit bel et bien d'une méthode.

### Question 5

**books = ["Cent ans de solitude", "Amours aux temps du Choléra",  "L'automne du Patriarche"]**

**books  représente :**

* + 

une chaîne de caractère.

* + 

une liste.

* + 

un dictionnaire.

* + 

un tuple.

### Question 6

**Chaque élément d'une liste est associé à son index, c'est à dire à sa position.**

* + 

Vrai

* + 

Faux

### Question 7

**friends = {"Doctor Who" : "Le tardis", "Red" : "Storybrooke", "Frodon" : "La Terre du Milieu"}**

**friends  représente :**

* + 

une liste.

* + 

un tuple.

* + 

un dictionnaire.

* + 

une chaîne de caractère.

### Question 8

**Chaque élément d'un dictionnaire est associé à une clé.**

* + 

Vrai

* + 

Faux

### Question 9

**books = ["Cent ans de solitude", "Amours aux temps du choléra", "L'automne du patriarche"]**

**Pour remplacer "L'automne du patriarche" par "Le Général dans son labyrinthe", j'écris :**

* + 

books[0] = "Le Général dans son labyrinthe"

* + 

books[2] = "Le Général dans son labyrinthe"

* + 

books.replace("L'automne du patriarche", "Le Général dans son labyrinthe")

### Question 10

**friends = {"Doctor Who" : "Le tardis", "Red" : "Storybrooke", "Frodon" : "La Terre du Milieu"}**

**Pour remplacer "Le tardis" par "Gallifrey", j'écris :**

* + 

friends[0] = "Gallifrey"

* + 

friends["Doctor Who"] = "Gallifrey"

* + 

friends.replace('Le tardis', "Gallifrey")

## Installez votre premier module

Nous avons beaucoup parlé des fonctions et des méthodes par défaut de Python. Pourtant, nous avons seulement aperçu une partie infime des possibilités offertes par ce langage !

Python offre de nombreux modules par défaut qui vous permettent d’aller au-delà des fonctionnalités. Un module est comme une extension : elle vous donne accès à d’autres méthodes spécialisées.

### Importer un module

Afin de ne pas alourdir Python bien des modules ne sont pas chargés par défaut. Mais ils sont déjà installés sur votre ordinateur ! Il vous suffit de les activer en utilisant le mot clé  import  suivi du nom du module. Comme ceci :

>>> import module\_name

Faisons un essai avec l’interpréteur. Lançons-le et importons le module turtle.

>>> import turtle

Maintenant, tapons  turtle.forward(100) .

>>> turtle.forward(100)

Allez ! Une surprise vous attend ! ^\_^  
…

Une fenêtre s’ouvre et Python dessine, pour vous, un trait ! Turtle est une librairie de dessin graphique. [La documentation est ici](https://docs.python.org/3/library/turtle.html).

Vous voulez vous amuser un peu ? Devinez ce que va dessiner ce code :

from turtle import \*

color('red', 'yellow')

begin\_fill()

while True:

forward(200)

left(170)

if abs(pos()) < 1:

break

end\_fill()

done()

### Utiliser les fonctions d’un module

Pourquoi la manière d’importer le module est-elle différente de celle que tu as montrée ?

**Méthode 1 (conseillée) :** Ecrire  import module\_name  va importer dans notre projet tous les objets utilisés par ce module. Parfait. Pour utiliser une méthode, nous écrivons  module\_name.method() . C’est la manière la plus commune, et la plus explicite, d’utiliser les modules.

**Méthode 2 (déconseillée)** : Néanmoins, parfois, nous ne souhaitons pas écrire le nom du module puis la méthode. Nous voulons utiliser la méthode telle qu’elle, par soucis de simplification. C’est souvent le cas dans des documentations. Dans ce cas, nous utilisons l’autre méthode d’import :  from module import \* . Nous pourrons alors écrire  method()  sans la précéder du nom du module.

Le souci, dans ce cas, est qu’il faut éviter de travailler avec plusieurs modules dans le même projet. En effet, elles peuvent (même si c’est assez rare) utiliser un même nom de méthode.

Je vous conseille donc d’utiliser la première méthode qui est plus sûre !

A noter que vous pouvez importer certains objets en particulier en écrivant  from module import some\_object .

### Utiliser le module random

Prenons un autre exemple qui nous servira à améliorer notre programme. Importons le module random dans notre projet.

import random

En lisant la documentation, je découvre la méthode  randint(a, b) . Elle renvoie un nombre au hasard compris entre a et b (donnés en paramètre). Par exemple :  random.randint(3, 7)  renverra un nombre entre 3 et 7 (compris).

Nous allons pouvoir mettre à jour notre fonction  get\_random\_item(object\_list) :

def get\_random\_item\_in(my\_list):

rand\_numb = random.randint(0, len(my\_list) - 1)

item = my\_list[rand\_numb] # get a quote from a list

return item # return the item

Découpons :

* 0  : l’index de ma première citation est 0
* len(object\_list) - 1  : calcule le nombre d’items dans ma liste (puisque le premier index est 0).  object\_list[-1]  renverra le dernier objet de la liste.
* random.randint(0, len(my\_list) - 1)  : renvoie un nombre compris entre 0 et la longueur totale de ma liste.
* item = object\_list[rand\_numb]  : trouve l’item qui correspond à l’index déterminé au hasard.
* return item  : renvoie la valeur de l’item

Paaaaaarfait ! Nous avons fini notre programme ! (ou presque…) Ajoutez ou supprimez autant de citations que vous le souhaitez dans les deux premières listes, puis lancez le programme.

C’est un peu fastidieux, n’est-ce pas ? Nous aimerions récupérer des citations automatiquement depuis internet et les stocker dans un fichier externe.

Nous pouvons le faire ! Oui, nous pouvons tout avec Python !

Découvrez avec moi, dans les deux prochains chapitres, comment faire.

### Code de ce chapitre

Retrouvez le code de ce chapitre  sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P4C1).

### Aller plus loin

Les deux prochains chapitres sont en bonus mais très fortement conseillés si vous souhaitez finir votre programme.

Si néanmoins vous décidez de vous arrêter là, pas de pression, c'est très bien aussi ! Merci d'avoir suivi ce cours (et d'avoir supporté mon humour ^^).

Votre apprentissage de Python ne s'arrête pas là ! [Découvrez le concept de Programmation Orientée objet](https://openclassrooms.com/courses/decouvrez-la-programmation-orientee-objet-avec-python) en réalisant un programme qui vous permettra de savoir si l'enfer c'est vraiment les autres !

## [BONUS] Stockez des citations dans un fichier avec le module json

Ce chapitre concerne une utilisation plus avancée de Python. Vous êtes prêt·e pour un nouveau défi ? Suivez-moi !

Attaquons-nous en premier à la fonctionnalité “Stocker nos citations et nos personnages dans un fichier externe.”

Afin que ce soit plus lisible, nous allons déplacer la liste de nos citations et celle de nos personnages dans deux fichiers différents. Nous pourrions les stocker dans un fichier Python mais il n’y a pas de calcul à faire donc ce serait inapproprié. Nous cherchons un format de donnée pour enregistrer du texte. Idéalement, j’aimerais beaucoup que mes citations et mes personnages soient stockés sous forme de dictionnaire pour que vous puissiez continuer à vous entraîner ! Hé oui, je suis un peu sadique ! ^\_^

La meilleure manière de le faire est d’utiliser un format de données qui s’appelle JSON (rien à voir avec le célèbre péplum). Il conserve, sous forme de dictionnaire, tout le texte que nous souhaitons.

Avant de l'utiliser, pensez à importer le module JSON !

import json

Ce format, qui signifie JavaScript Object Notation, est souvent utilisé pour représenter le contenu d’une page web car il est très facile d’imbriquer des listes à l’intérieur d’autres listes. Vous ne serez pas en pays inconnu : la syntaxe ressemble beaucoup à celle de Python.

Chaque valeur ayant forcément une clé, voici la manière dont nous allons stocker nos objets :

{"nom\_objet": "valeur"}

Commençons par les citations. Créons un nouveau document   quotes.json   et remplissons-le :

quotes.json

[{"quote": "Ecoutez-moi, Monsieur Shakespeare, nous avons beau être ou ne pas être, nous sommes !"},

{"quote": "On doit pouvoir choisir entre s'écouter parler et se faire entendre."}]

Faisons-en de même pour les personnages :

characters.json

[{"character": "alvin et les Chimpunks"},

{"character": "Babar"}]

Bien. Je veux maintenant lire le fichier characters.json et le convertir en liste. Vous êtes prêt·e ? Ecrivons du pseudo-code :

def read\_values\_from\_json():

# Create a new empty list

# open a json file with my objects

# load all the data contained in this file

# add each item in my list

# return my completed list

Oooooookay ! On se frotte les mains, on s’étire un peu les épaules et on y va !

Complétons avec ce que nous savons déjà faire :

def read\_values\_from\_json():

values = []

# open a json file with my objects

# load all the data contained in this file. data = entries

for entry in data:

values.append(entry["character"])

return values

Trouvons maintenant, comment charger un document. La documentation de Python nous propose la méthode  open('path\_to\_file')  qui est souvent utilisée avec un bloc  with .

On n’a pas vu ça encore !

Effectivement. Le bloc  with file as variable:  vous permet de travailler avec un document externe en toute sécurité. Lorsque vous utilisez la méthode  open() , Python ouvre le fichier pour vous mais ne le ferme pas après avoir fini de l’utiliser ! Or vous risquez des erreurs si vous essayez d’ouvrir un fichier qui est déjà ouvert… Le bloc  with  ouvre le fichier et le ferme à la fin. Magique !

Pour en savoir plus, lisez la [documentation](https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects). 😉

Complétons :

def read\_values\_from\_json(key):

values = []

with open("characters.json") as f:

# load all the data contained in this file. data = entries

for entry in data:

values.append(entry["character"])

return values

Fouillons maintenant dans la documentation du module json pour trouver comment charger le contenu d’un fichier json et le convertir en objet Python (car pour l’instant ce n’est que du texte !).

La méthode [load()](https://docs.python.org/3/library/json.html" \l "json.load" \t "_blank) semble parfaite : elle transforme un fichier JSON en objet Python.

def read\_values\_from\_json(key):

values = []

with open("characters.json") as f:

data = json.load(f)

for entry in data:

values.append(entry["character"])

return values

Nous pouvons aller plus loin en spécifiant le fichier à lire et la clé à utiliser. Ceci nous permettra de lire le fichier de personnages et celui de citations !

def read\_values\_from\_json(file, key):

values = []

with open(file) as f:

data = json.load(f)

for entry in data:

values.append(entry[key])

return values

Bravo ! Nous avons fait le plus difficile !  
Afin de finir ce chapitre en toute beauté, créons une nouvelle fonction qui renverra un personnage au hasard :

# Return a random value from a json file

def random\_character():

all\_values = read\_values\_from\_json()

return get\_random\_item\_in(all\_values)

Bien. Il nous manque encore une dernière étape : récupérer des citations venant d’Internet ! Je vous laisse faire une petite pause, prendre un café ou marcher un peu et je vous retrouve en pleine forme dans le prochain chapitre !

N'hésitez pas à vous référer à la doc de toutes les librairies Python lorsque vous avez un projet en tête pour vérifier les briques que vous allez déjà pouvoir réutiliser !

### Code du chapitre

Retrouvez le code de ce chapitre sur le dépôt GitHub [en cliquant ici](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/P4C2).

## [BONUS] Collectez des citations automatiquement avec Scrapy

La librairie standard de Python a beau être fournie et disponible, elle n’est pas pour autant exhaustive. Que faire si vous n’y trouvez pas votre bonheur? Vous voudrez en importer d’une source externe.

La communauté Python est, comme je vous l’ai expliqué plus tôt, très active et a produit plusieurs milliers de modules. Pip est l’outil de gestion de paquets utilisé pour installer et gérer des modules externes. Il est installé automatiquement quand vous installez Python. Vous les trouverez toutes sur [PyPI](https://pypi.python.org/pypi?%3Aaction=browse" \t "_blank), l’index des librairies Python ([Python Package Index](https://pypi.python.org/pypi?%3Aaction=browse)).

Comment installer une librairie ?

Très simplement ! Vous écrivez  python3 -m pip install ma\_librairie .

Prenons un exemple.

Une des librairies les plus populaires pour récupérer de l’information sur Internet est [Scrapy](https://scrapy.org/" \t "_blank). Il s’agit d’un scraper entièrement personnalisable, en d’autres termes un petit robot que vous pouvez configurer vous-même pour qu’il cherche ce que vous souhaitez.

Pour l’installer, nous écrivons la ligne de commande suivante dans la console :

>>> python3 -m pip install scrapy

C’est d’ailleurs la toute première instruction de leur [page d’accueil](https://scrapy.org/).

Et voilà, c’est installé !

### Le fichier requirements.txt

Une bonne pratique est de réunir toutes les librairies externes dans un fichier qui s’appelle  requirements.txt  et qui se trouve à la racine de votre dossier (au même niveau que  san\_antonio.py ). Ainsi, il vous suffit d’écrire  python3 -m pip install -r requirements.txt  pour que toutes les librairies s’installent d’un seul coup !

Certains développeurs spécifient la version de chaque librairie. Par exemple :  BeautifulSoup==3.2.0

Avant de continuer et de configurer Scrapy, j’ai nettoyé un peu le code en le réorganisant. Je vous laisse regarder avant de passer à la suite !

# -\*- coding: utf8 -\*-

import json

import random

# Give a Json file and return a List

def read\_values\_from\_json(path, key):

values = []

with open(path) as f:

data = json.load(f)

for entry in data:

values.append(entry[key])

return values

# Give a json and return a list

def clean\_strings(sentences):

cleaned = []

# Store quotes on a list. Create an empty list and add each sentence one by one.

for sentence in sentences:

# Clean quotes from whitespace and so on

clean\_sentence = sentence.strip()

# don't use extend as it adds each letter one by one!

cleaned.append(clean\_sentence)

return cleaned

# Return a random item in a list

def random\_item\_in(object\_list):

rand\_numb = random.randint(0, len(object\_list) - 1)

return object\_list[rand\_numb]

# Return a random value from a json file

def random\_value(source\_path, key):

all\_values = read\_values\_from\_json(source\_path, key)

clean\_values = clean\_strings(all\_values)

return random\_item\_in(clean\_values)

#####################

###### QUOTES #######

#####################

# Gather quotes from San Antonio

def random\_quote():

return random\_value('quotes.json', 'quote')

######################

#### CHARACTERS ######

######################

# Gather characters from Wikipedia

def random\_character():

return random\_value('characters.json', 'character')

######################

#### INTERACTION ######

######################

# Print a random sentence.

def print\_random\_sentence():

rand\_quote = random\_quote()

rand\_character = random\_character()

print(">>>> {} a dit : {}".format(rand\_character, rand\_quote))

def main\_loop():

while True:

print\_random\_sentence()

message = ('Voulez-vous voir une autre citation ?'

'Pour sortir du programme, tapez [B].')

choice = input(message).upper()

if choice == 'B':

break

# This will stop the loop!

if \_\_name\_\_ == '\_\_main\_\_':

main\_loop()

C'est quoi cette ligne  \_\_name\_\_ == '\_\_main\_\_':  ?

Une fois n'est pas coutume, je vous donne un code tout cuit ! Vous apprendrez à l'écrire vous-mêmes en avançant dans Python. Vous êtes trop impatient·e·s pour attendre ? C'est le moment d'une petite recherche sur votre moteur de recherche préféré ! :D

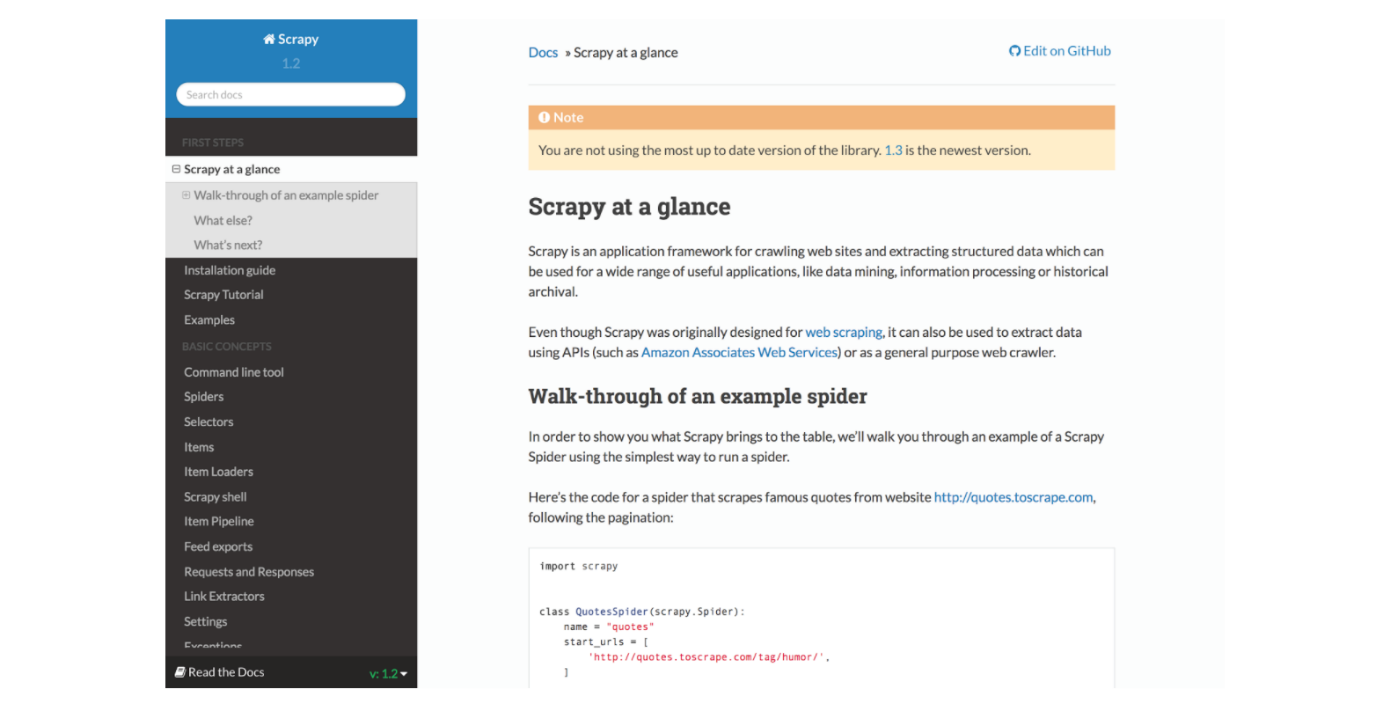
Bien, nous sommes prêt·e·s à commencer.

### Utiliser Scrapy

Mon objectif n’est pas de faire un cours sur Scrapy mais de vous montrer comment lire une documentation. Vous verrez, c’est parfois facile, parfois moins, et il n’y a pas de règle précise. L’essentiel est de s’accrocher !

Heureusement pour nous, les développeurs derrière Scrapy ont tout prévu. Ils ont créé un [robot d’exemple](https://doc.scrapy.org/en/1.2/intro/overview.html) avec toutes les instructions ! \o/

À nous de l’adapter !



Suivons les instructions et créons un nouveau document  characters\_scrapper.py . A l’intérieur, je colle le code fourni par Scrapy et je commence à l’adapter.

Comment fonctionne un scraper ?

Il va sur une url que nous lui donnons puis va trouver les éléments qui correspondent à un sélecteur CSS.

CSS ? Pour Couch Surfer Stylé ?

😀

Non… Pour Cascading StyleSheet. Vous n’avez pas besoin d’être expert en CSS pour configurer un scraper basique. Il vous suffit de trouver le sélecteur.

Voici un petit rappel :

 div  : selecteur d’une balise HTML  div

 .myclass  : cible toutes les balises ayant cette classe

 #myid  : cible la balise ayant cet id

 div .myclass  : cible tous les éléments ayant la classe  .myclass  contenus dans une balise  div .

CSS est un langage qui sert à la mise en page d’un site. Sans CSS, le web est bien triste !

Démonstration !

Rendez-vous sur [cette page Wikipedia](https://fr.wikipedia.org/wiki/Cat%C3%A9gorie:Personnage_d%27animation) qui affiche une liste de personnages. Nous voulons récupérer le nom de tous les personnages.

Afin de trouver le sélecteur, je clique droit sur l’élément que je souhaite cibler et je clique sur “inspecter l’élément”. Le sélecteur est le nom de la balise HTML (a dans mon cas).

Sans aucune notion d’HTML et de CSS il n’est pas possible de configurer un scraper. Mais pas de panique ! Vous pouvez reprendre mon exemple et l’adapter à vos besoins très facilement. Et vous pourrez approfondir vos notions d’HTML et CSS plus tard dans [ce cours](https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3).

Voici mon scraper :

import scrapy

class BlogSpider(scrapy.Spider):

name = 'characterspider'

start\_urls = ['https://fr.wikipedia.org/wiki/Cat%C3%A9gorie:Personnage\_d\'animation']

def parse(self, response):

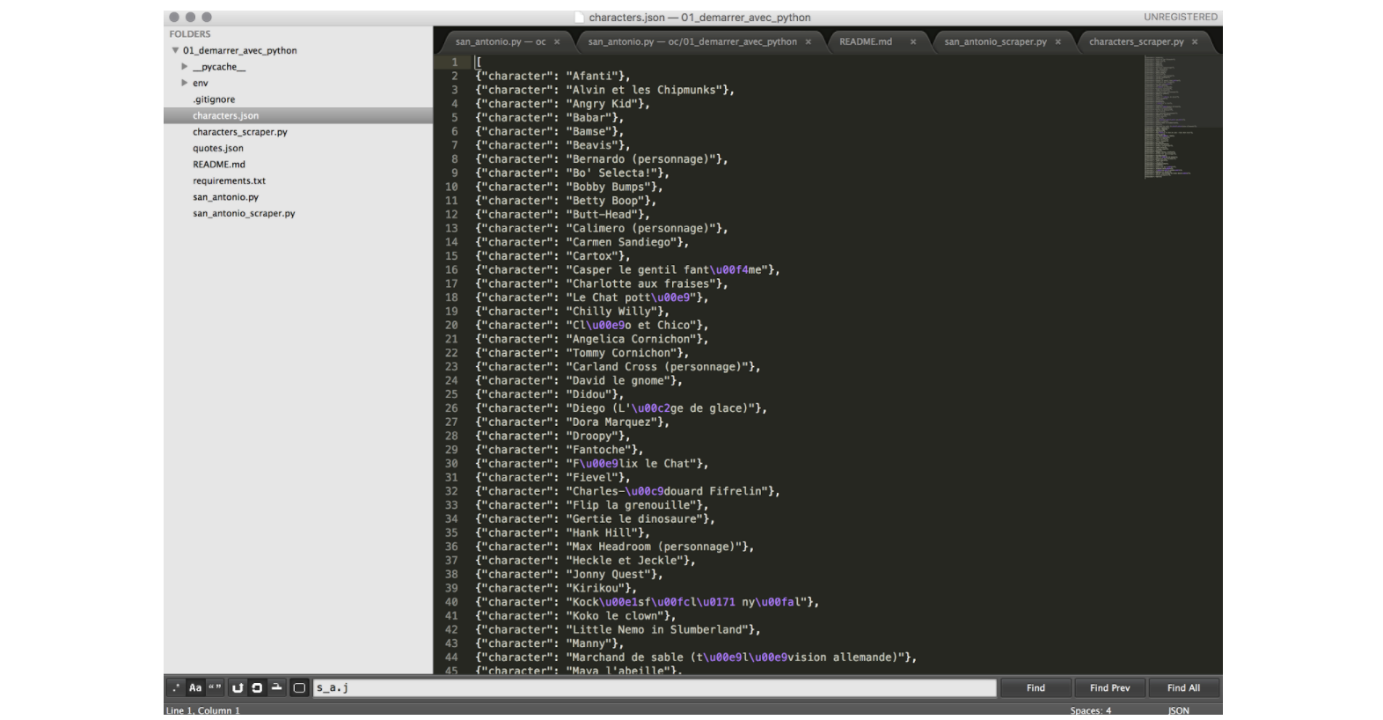
for link in response.css('div#mw-pages div.mw-content-ltr li'):

yield {'character': link.css('a ::text').extract\_first()}

Pour récupérer les personnages, et les stocker dans un nouveau fichier  characters.json , je lance la commande suivante :

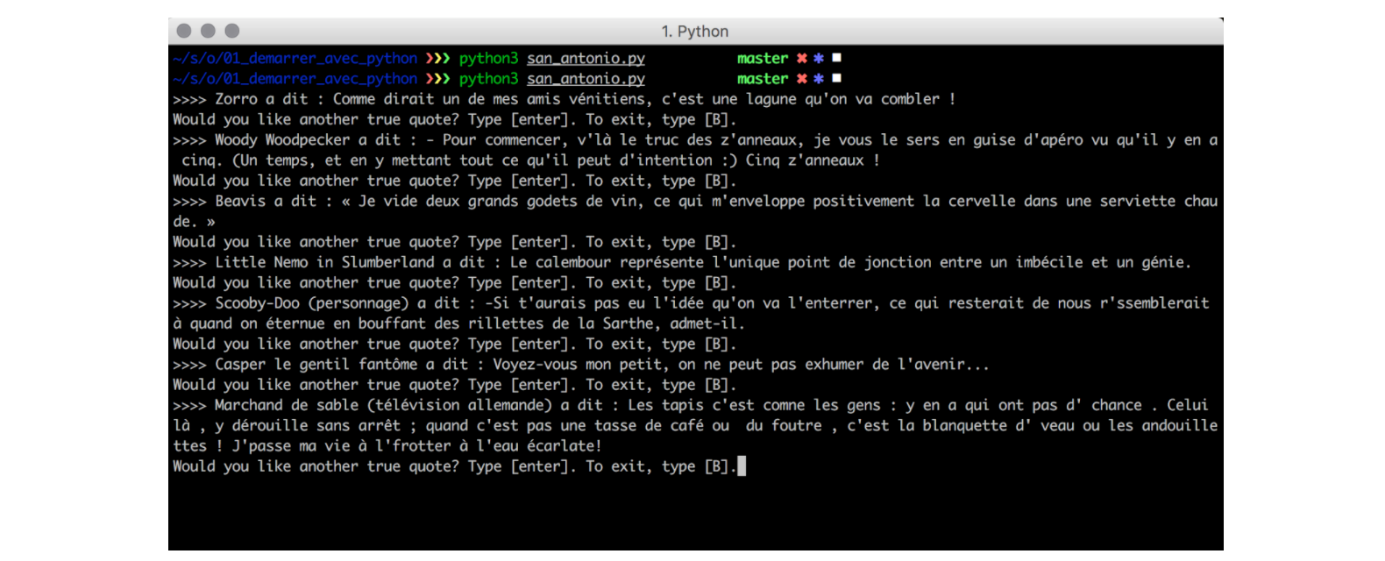
$ scrapy runspider characters.py -o characters.json

Et là, magie ! Tous mes personnages apparaissent !



Je vous laisse prendre connaissance avec l’intégralité du code à cette adresse :  [voir le code final du Scraper](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/master).

Bravo ! C’est terminé ! Vous pouvez être fièr·e de vous !



Vous connaissez maintenant l’existence des scrapers et vous avez une petite idée des informations que vous pouvez récolter.

Pour information, ces robots sont utilisés par de nombreuses industries pour agréger les données “publiques” accessibles librement ou pour publier du contenu massivement en l’automatisant. Vous pourriez ainsi, tout à fait illégalement, scraper le site du Monde, copier l’intégralité des articles et les publier sur votre site. Il s’agit d’une technique si répandue que Google a créé un formulaire pour dénoncer les sites de scraping ! [Lire l’article de MyPoseo sur Google Scraper Report](http://fr.myposeo.com/blog/google-scraper-report-formulaire-denoncer-les-sites-scraping/).

L’utilisation de robot est également une technique d’acquisition assez répandue. Airbnb a ainsi utilisé un scraper pour augmenter sa base d’utilisateurs ([lire l’analyse de GrowthHackers](https://growthhackers.com/growth-studies/airbnb) 🇬🇧) !

Sans devenir parano, songez à toutes les données que de tels robots peuvent récupérer sur vous...

### Code de ce chapitre

Cliquez sur l'image pour visualiser le code source ou cliquez sur [ce lien](https://github.com/OpenClassrooms-Student-Center/demarrez_votre_projet_avec_python/tree/master).

Pour télécharger le projet, cliquez sur "Clone or Download" puis sur "Download Zip".

