Vincent Le Goff

APPRENEZ À PROGRAMMER EN

PYTHON

3e édition

OPENCLASSROOMS

Éditions EYROLLES

APPRENEZ À PROGRAMMER EN

PYTHON

3e édition

Vous n'y connaissez rien en programmation et vous souhaitez apprendre un langage clair et intuitif ? Python est fait pour vous ! Vous découvrirez dans ce livre, conçu pour les débutants, tout ce dont vous avez besoin pour programmer, des bases à la bibliothèque standard, en passant par la programmation orientée objet et l'acquisition d'outils avancés ou professionnels pour devenir plus efficace.

QU'ALLEZ-VOUS APPRENDRE ?

- **Qu'est-ce que la programmation ? Quel langage choisir ? Pourquoi Python ?**
- **Installation de Python et découverte du langage**
- **Les concepts de la programmation orientée objet**
- **Initiation aux interfaces graphiques avec Tkinter**
- **Communication en réseau dans les programmes Python**
- **Les bonnes pratiques pour améliorer vos codes**
- **Les réflexes du « bon programmeur » pour tirer parti de votre code et de celui des autres** Nouveau
- **Les outils du programmeur professionnel (chasse aux erreurs, utilisation de bibliothèques…)** Nouveau

À PROPOS DE L'AUTEUR

Passionné d'informatique, Vincent Le Goff découvre au lycée la programmation en Python, un langage qu'il affectionne tout particulièrement pour son aspect simple et puissant. Étudiant à IN'TECH, il se spécialise en Système et Réseaux. Sur son temps libre, il publie des cours sur OpenClassrooms et participe également à plusieurs projets open source. Une belle réussite quand on sait que Vincent est non-voyant et malentendant !

L'ESPRIT D'OPENCLASSROOMS

Des cours ouverts, riches et vivants, conçus pour tous les niveaux et accessibles à tous gratuitement sur notre plate-forme d'e-éducation : www.openclassrooms.com. Vous y vivrez une véritable expérience communautaire de l'apprentissage, permettant à chacun d'apprendre avec le soutien et l'aide des autres étudiants sur les forums. Vous profiterez des cours disponibles partout, tout le temps.

www.editions-eyrolles.com

APPRENEZ À PROGRAMMER EN

PYTHON

SUR LE MÊME THÈME

G. Swinnen. — **Apprendre à programmer avec Python 3**.
N° 13434, 3e édition, 2012, 435 pages.

J-B. Civet, B. Hanuš. — **Algorithmique et programmation en Python**.
N° 67769, 2019, 96 pages.

M. O'Hanlon, D. Whale. — **Apprendre à coder en Python avec Minecraft**.
N° 67721, 2e édition, 2019, 304 pages.

J. Briggs. — **Python pour les kids**.
N° 14088, 2015, 332 pages.

DANS LA MÊME COLLECTION

É. Lalitte. — **Apprenez le fonctionnement des réseaux TCP/IP**.
N° 67776, 4e édition, 2019, 452 pages.

C. Herby. — **Apprenez à programmer en Java**.
N° 67521, 3e édition, 2018, 788 pages.

M. Nebra. — **Concevez votre site web avec PHP et MySQL**.
N° 67475, 3e édition, 2017, 392 pages.

M. Nebra. — **Réalisez votre site web avec HTML 5 et CSS 3**.
N° 67476, 2e édition, 2017, 364 pages.

V. Thuillier. — **Programmez en orienté objet en PHP**.
N° 14472, 2e édition, 2017, 474 pages.

J. Pardanaud, S. De la Marck. — **Découvrez le langage JavaScript**.
N° 14399, 2017, 478 pages.

Vincent Le Goff

APPRENEZ À PROGRAMMER EN

PYTHON

3e édition

OPENCLASSROOMS

Éditions EYROLLES

Éditions Eyrolles
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Mentions légales :
Conception couverture : Sophie Bai
Illustrations chapitres : Fan Jiyong et Sophie Bai

© Éditions Eyrolles, 2020, ISBN : 978-2-212-67871-0

Avant-propos

J'ai commencé à m'intéresser à l'informatique, et plus particulièrement au monde de la programmation, au début du lycée, il y a maintenant près de quinze ans. J'ai abordé ce terrain inconnu avec une grande curiosité… qui n'a pas encore faibli puisque je suis aujourd'hui étudiant à IN'TECH, une école supérieure d'ingénierie informatique ! Au premier abord, la programmation me semblait un monde aride et froid, rempli d'équations compliquées et de notions abstraites.

Heureusement, le premier langage à avoir attiré mon attention s'est trouvé être Python : à la fois simple et extrêmement puissant, je le considère aujourd'hui comme le meilleur choix quand on souhaite apprendre à programmer. Il est d'ailleurs resté le langage que j'utilise le plus dans les projets libres auxquels je contribue.

Cependant, Python n'est pas seulement simple : c'est un langage puissant. Il existe une différence entre connaître un langage et coder efficacement avec. Plusieurs années de pratique m'ont été nécessaires pour comprendre pleinement cette différence.

Les cours sur le langage Python s'adressant aux débutants ne sont pas rares sur le Web et beaucoup sont de grande qualité. Toutefois, il en existe trop peu, à mon sens, qui permettent de comprendre à la fois la syntaxe et la philosophie du langage.

Mon objectif ici est qu'après avoir lu ce livre, vous sachiez programmer en Python. Et par « programmer », je n'entends pas seulement maîtriser la syntaxe du langage, mais aussi comprendre sa philosophie.

Étant non-voyant et malentendant, je me suis efforcé de rendre ce cours aussi accessible que possible à tous. Ainsi, ne soyez pas surpris si vous y trouvez moins de schémas et d'illustrations que dans d'autres cours. J'ai fait en sorte que leur présence ne soit pas indispensable à la compréhension du lecteur.

Pour ceux qui se demandent comment je travaille, j'ai un ordinateur absolument semblable au vôtre. Pour l'utiliser, j'installe sur mon système un logiciel qu'on appelle *lecteur d'écran*. Ce dernier me dicte une bonne partie des informations affichées dans la fenêtre du logiciel que j'utilise, comme le navigateur Internet. Le lecteur, comme

son nom l'indique, lit grâce à une voix synthétique les informations qu'il détecte sur la fenêtre et peut également les transmettre à une *plage tactile*. C'est un périphérique qui se charge d'afficher automatiquement en braille les informations que lui transmet le lecteur d'écran. Avec ces outils, je peux donc me servir d'un ordinateur, aller sur Internet et même programmer !

FIGURE 1 – La plage tactile et le casque transmettent les informations affichées à l'écran

Qu'allez-vous apprendre en lisant ce livre ?

Ce livre s'adresse au plus grand nombre :

- si le mot programmation ne vous évoque rien de précis, ce livre vous guidera pas à pas dans la découverte du monde du **programmeur** ;
- si vous connaissez déjà un autre langage de programmation, ce livre présente de façon aussi claire que possible la syntaxe de Python et des exemples d'utilisation concrète de ce langage ;
- si vous connaissez déjà Python, ce cours peut vous servir de support comparatif avec d'autres livres et cours existants ;
- si vous enseignez le langage Python, j'ai espoir que ce livre pourra être un support utile, autant pour vous que pour vos étudiants.

Ce livre est divisé en cinq parties. Les trois premières sont à lire dans l'ordre, sauf si vous avez déjà de solides bases en Python :

1. **Introduction à Python**. Vous y apprendrez d'abord, si vous l'ignorez, ce que signifie **programmer**, ce qu'est Python et la syntaxe de base du langage.
2. **La Programmation Orientée Objet côté utilisateur**. Après avoir vu les bases de Python, nous allons étudier la **façade objet** de ce langage. Dans cette partie, vous apprendrez à utiliser les **classes** que définit Python. Ne vous inquiétez pas, les concepts d'objet et de classe seront largement détaillés ici. Donc, si ces mots ne vous disent rien au premier abord, pas d'inquiétude !
3. **La Programmation Orientée Objet côté développeur**. Cette partie poursuit l'approche de la façade objet débutée dans la partie précédente. Cette fois, cependant, au lieu d'être utilisateur des classes déjà définies par Python, vous allez apprendre à en créer. Là encore, ne vous inquiétez pas : nous verrons tous ces concepts pas à pas.
4. **Les merveilles de la bibliothèque standard**. Cette partie étudie plus en détail certains modules déjà définis par Python. Vous y apprendrez notamment à manipuler les dates et heures, créer des interfaces graphiques, construire une architecture réseau... et bien plus !
5. **Annexes**. Enfin, cette partie regroupe les annexes et résumés du cours. Il s'agit de notions qui ne sont pas absolument nécessaires pour développer en Python mais que je vous encourage tout de même à lire attentivement.

Comment lire ce livre ?

Suivez l'ordre des chapitres

Lisez ce livre comme on lit un roman. Il a été conçu pour cela.

Contrairement à beaucoup de livres techniques où il est courant de lire en diagonale et de sauter certains chapitres, il est ici très fortement recommandé de suivre l'ordre du cours, à moins que vous ne soyez déjà un peu expérimentés.

Pratiquez en même temps

Pratiquez régulièrement. N'attendez pas d'avoir fini de lire ce livre pour allumer votre ordinateur et faire vos propres essais.

Utilisez les codes web !

Afin de tirer parti d'OpenClassrooms dont ce livre est issu, celui-ci vous propose ce qu'on appelle des « codes web ». Ce sont des codes à six chiffres à saisir sur une page d'OpenClassrooms pour être automatiquement redirigé vers un site web sans avoir à en recopier l'adresse.

Pour les utiliser, rendez-vous sur la page suivante :

`http://fr.openclassrooms.com/codeweb.html`

Un formulaire vous invite à rentrer votre code. Faites un premier essai avec le suivant :

▷ Tester le code web
Code web : 123456

Ces codes ont deux intérêts :

— ils vous redirigent vers les sites web présentés tout au long du cours, vous facilitant l'obtention des logiciels dans leur toute dernière version ;
— ils vous permettent de télécharger les codes sources inclus dans ce livre, ce qui vous évitera d'avoir à recopier certains programmes un peu longs.

Ce système de redirection nous permet de tenir à jour le livre que vous avez entre les mains sans que vous ayez besoin d'acheter systématiquement chaque nouvelle édition. Si un site web change d'adresse, nous modifierons la redirection mais le code web à utiliser restera le même. Si un site web disparaît, nous vous redirigerons vers une page d'OpenClassrooms expliquant ce qui s'est passé et vous proposant une alternative.

En clair, c'est un moyen de nous assurer de la pérennité de cet ouvrage sans que vous ayez à faire quoi que ce soit !

Remerciements

De nombreuses personnes ont, plus ou moins directement, participé à ce livre. Mes remerciements leurs sont adressés :

— à ma famille avant tout, qui a su m'encourager, dans ce projet comme dans tout autre, du début jusqu'à la fin ;
— aux personnes, trop nombreuses pour que j'en dresse ici la liste, qui ont contribué, par leurs encouragements, leurs remarques et parfois leurs critiques, à faire de ce livre ce qu'il est ;
— à l'équipe d'OpenClassrooms qui a rendu ce projet envisageable et a travaillé d'arrache-pied pour qu'il se concrétise ;
— aux membres d'OpenClassrooms (Site du Zéro à l'époque) qui ont contribué à sa correction ou son enrichissement.

Table des matières

Première partie

Introduction à Python

Chapitre 1

Qu'est-ce que Python ?

Difficulté :

Vous avez décidé d'apprendre Python et je ne peux que vous en féliciter. J'essaierai d'anticiper vos questions et de ne laisser personne en arrière.

Dans ce chapitre, je vais d'abord vous expliquer ce qu'est un langage de programmation. Nous verrons ensuite brièvement l'histoire de Python, afin que vous sachiez au moins d'où vient ce langage ! Ce chapitre est théorique mais je vous conseille vivement de le lire quand même.

La dernière section portera sur l'installation de Python, une étape essentielle pour continuer ce cours. Que vous travailliez avec Windows, Linux ou macOS, vous y trouverez des explications précises sur l'installation.

Allez, on attaque !

Un langage de programmation ? Qu'est-ce que c'est ?

La communication humaine

Non, ceci n'est pas une explication biologique ou philosophique, ne partez pas ! Très simplement, si vous arrivez à comprendre ces suites de symboles étranges et déconcertants que sont les lettres de l'alphabet, c'est parce que nous respectons certaines conventions, dans le langage et dans l'écriture. En français, il y a des règles de grammaire et d'orthographe, je ne vous apprends rien. Vous communiquez en connaissant plus ou moins consciemment ces règles et en les appliquant plus ou moins bien, selon les cas. Cependant, ces règles peuvent être aisément contournées : personne ne peut prétendre connaître l'ensemble des règles de la grammaire et de l'orthographe françaises, et peu de gens s'en soucient. Après tout, même si vous faites des fautes, les personnes avec qui vous communiquez pourront facilement vous comprendre. Quand on communique avec un ordinateur, cependant, c'est très différent.

Mon ordinateur communique aussi !

Eh oui, votre ordinateur communique sans cesse avec vous et vous communiquez sans cesse avec lui. D'accord, il vous dit très rarement qu'il a faim, que l'été s'annonce caniculaire et que le dernier disque de ce groupe très connu était à pleurer. Il n'y a rien de magique si, quand vous cliquez sur la petite croix en haut à droite de l'application en cours, celle-ci comprend qu'elle doit se fermer.

Le langage machine

En fait, votre ordinateur se fonde aussi sur un langage pour communiquer avec vous ou avec lui-même. Les opérations qu'un ordinateur peut effectuer à la base sont des plus classiques : l'addition de deux nombres, leur soustraction, leur multiplication, leur division, entière ou non. Et pourtant, ces cinq opérations suffisent amplement à faire fonctionner les logiciels de simulation les plus complexes ou les jeux super-réalistes. Tous ces logiciels fonctionnent en gros de la même façon :

— une suite d'instructions écrites en langage machine compose le programme ;
— lors de l'exécution du programme, ces instructions décrivent à l'ordinateur ce qu'il faut faire (l'ordinateur ne peut pas le deviner).

Une liste d'instructions ? Qu'est-ce que c'est encore que cela ?

En schématisant volontairement, une instruction pourrait demander au programme de se fermer si vous cliquez sur la croix en haut à droite de votre écran, ou de rester en tâche de fond si tel est son bon plaisir. Toutefois, en langage machine, une telle action demande à elle seule un nombre assez important d'instructions. Mais bon, vous pouvez

vous en douter, parler avec l'ordinateur en langage machine, qui ne comprend que le binaire, ce n'est ni très enrichissant, ni très pratique et en tout cas pas très drôle. On a donc inventé des langages de programmation pour faciliter la communication avec l'ordinateur.

Le langage binaire est uniquement constitué de 0 et de 1. « 01000010011011110110111001101010011011110111010101110010 », par exemple, signifie « Bonjour ». Bref, autant vous dire que discuter en binaire avec un ordinateur peut être long (surtout pour vous).

Les langages de programmation

Les langages de programmation sont bien plus faciles à comprendre pour nous, pauvres êtres humains que nous sommes. Le mécanisme reste le même, mais le langage est bien plus compréhensible. Au lieu d'écrire les instructions dans une suite assez peu intelligible de 0 et de 1, les ordres donnés à l'ordinateur sont écrits dans un « langage », souvent en anglais, avec une syntaxe particulière qu'il est nécessaire de respecter. Toutefois, avant que l'ordinateur puisse comprendre ces instructions, celles-ci doivent être traduites en langage machine (figure 1.1).

FIGURE 1.1 – Traduction d'un programme en langage binaire

En gros, le programmeur « n'a qu'à » écrire des **lignes de code** dans le langage qu'il a choisi, les étapes suivantes sont automatisées pour permettre à l'ordinateur de les décoder.

Il existe un grand nombre de langages de programmation et Python en fait partie. Il n'est pas nécessaire pour le moment de donner plus d'explications sur ces mécanismes très schématisés. Si vous n'avez pas réussi à comprendre les mots de vocabulaire et l'ensemble de ces explications, cela ne vous pénalisera pas pour la suite. Néanmoins, je trouve intéressant de donner ces précisions quant aux façons de communiquer avec son ordinateur.

Pour la petite histoire

La première version de Python est sortie en 1991. Créé par **Guido van Rossum**, il a voyagé du Macintosh de son créateur, qui travaillait à cette époque au *Centrum voor*

Wiskunde en Informatica aux Pays-Bas, jusqu'à se voir associer une organisation à but non lucratif particulièrement dévouée, la **Python Software Foundation**, créée en 2001. Ce langage a été baptisé ainsi en hommage à la troupe de comiques les « Monty Python ».

À quoi peut servir Python ?

Python est un langage puissant, à la fois facile à apprendre et riche en possibilités. Dès l'instant où vous l'installez sur votre ordinateur, vous disposez de nombreuses fonctionnalités intégrées que nous allons découvrir tout au long de ce livre.

Il est, en outre, très facile d'étendre les fonctionnalités existantes, comme nous allons le voir. Ainsi, il existe ce qu'on appelle des **bibliothèques** qui aident le développeur à travailler sur des projets particuliers. Plusieurs bibliothèques peuvent être installées pour, par exemple, développer des interfaces graphiques en Python.

Concrètement, voilà ce qu'on peut faire avec Python :

— de petits programmes très simples, appelés **scripts**, chargés d'une mission très précise sur votre ordinateur ;
— des programmes complets, comme des jeux, des suites bureautiques, des logiciels multimédias, des clients de messagerie...
— des projets très complexes, comme des progiciels (groupes de plusieurs logiciels fonctionnant ensemble, principalement utilisés dans le monde professionnel).

Voici quelques-unes des fonctionnalités offertes par Python et ses bibliothèques :

— créer des interfaces graphiques ;
— faire circuler des informations au travers d'un réseau ;
— dialoguer d'une façon avancée avec votre système d'exploitation ;
— créer et héberger un site web dynamique ;
— et bien d'autres...

Bien entendu, vous n'allez pas apprendre tout cela en quelques minutes. Ce cours vous donnera des bases suffisamment larges pour développer des projets qui deviendront, par la suite, assez importants.

Un langage de programmation interprété

Eh oui, vous allez devoir patienter encore un peu car il me reste deux ou trois choses à vous expliquer et il est important de connaître un minimum ces détails qui peuvent sembler peu pratiques de prime abord. Python est un langage de programmation **interprété**, c'est-à-dire que les instructions que vous lui envoyez sont « transcrites » en langage machine au fur et à mesure de leur lecture. D'autres langages (comme le C / C++) sont appelés « langages **compilés** » car, avant de pouvoir les exécuter, un logiciel spécialisé se charge de transformer le code du programme en langage machine. On appelle cette étape la « **compilation** ». À chaque modification du code, il faut rappeler une étape de compilation.

Les avantages d'un langage interprété sont la simplicité (on ne passe pas par une étape de compilation avant d'exécuter son programme) et la portabilité (un langage tel que Python est censé fonctionner aussi bien sous Windows que sous Linux ou macOS et on ne devrait avoir à effectuer aucun changement dans le code pour le passer d'un système à l'autre). Cela ne veut pas dire que les langages compilés ne sont pas portables, loin de là ! Toutefois, on doit souvent utiliser des compilateurs différents et, d'un système à l'autre, certaines instructions ne sont pas compatibles, voire se comportent différemment.

En contrepartie, un langage compilé se révélera bien plus rapide qu'un langage interprété (la traduction à la volée de votre programme ralentit l'exécution), bien que cette différence tende à se faire de moins en moins sentir au fil des améliorations. De plus, il faudra installer Python sur le système d'exploitation que vous utilisez pour que l'ordinateur comprenne votre code.

Différentes versions de Python

Lors de la création de la Python Software Foundation, en 2001, et durant les années qui ont suivi, le langage est passé par une suite de versions que l'on a englobées dans l'appellation Python 2.x (2.3, 2.5, 2.6...). Depuis le 13 février 2009, la version 3.0.1 est disponible. Elle casse la **compatibilité ascendante** qui prévalait lors des dernières versions.

Compatibilité quoi ?

Quand un langage de programmation est mis à jour, les développeurs se gardent bien de supprimer ou de trop modifier d'anciennes fonctionnalités. L'intérêt est qu'un programme qui fonctionne sous une certaine version marchera toujours avec la nouvelle version en date. Cependant, la Python Software Foundation, observant un bon nombre de fonctionnalités obsolètes, mises en œuvre plusieurs fois... a décidé de nettoyer tout le projet. Un programme qui tourne à la perfection sous Python 2.x devra donc être mis à jour un minimum pour fonctionner de nouveau sous Python 3. C'est pourquoi je vais vous conseiller ultérieurement de télécharger et d'installer la dernière version en date de Python. Je m'attarderai en effet sur les fonctionnalités de Python 3 et certaines d'entre elles ne seront pas accessibles (ou pas sous le même nom) dans les anciennes versions.

Notons que la branche 2.x de Python ne sera plus mise à jour à partir de 2020, c'est-à-dire très prochainement. Vous pourrez toujours apprendre et utiliser Python dans cette version, mais les bogues rencontrés ne seront plus corrigés. C'est une raison supplémentaire de passer à Python 3.

Ceci étant posé, tous à l'installation !

Installer Python

L'installation de Python est un jeu d'enfant, aussi bien sous Windows que sous les systèmes Unix. Quel que soit votre système d'exploitation, vous devez vous rendre sur le site officiel de Python.

▷ Site officiel de Python
Code web : 199501

Sous Windows

1. Dans la section `Download` (téléchargement), vous devriez voir la version la plus récente de Python (3.7.3 à l'heure où j'écris ces lignes). Cliquez sur le numéro de version.
2. On vous propose un (ou plusieurs) lien(s) vers une version Windows : sélectionnez celle qui conviendra à votre processeur. Si vous avez un doute, téléchargez une version « x86 ». Si votre ordinateur vous signale qu'il ne peut exécuter le programme, essayez une autre version de Python.
3. Enregistrez puis exécutez le fichier d'installation et suivez les étapes. Ce n'est ni très long ni très difficile.
4. Une fois l'installation terminée, vous pouvez vous rendre dans le menu `Démarrer` > `Tous les programmes`. Python devrait apparaître dans cette liste (figure 1.2). Nous verrons bientôt comment le lancer, pas d'impatience...

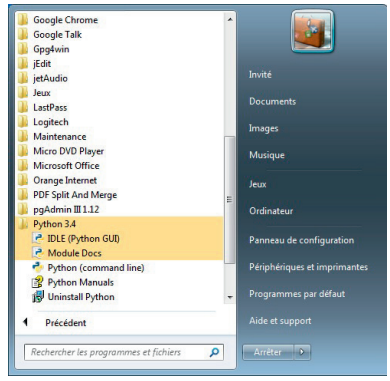

FIGURE 1.2 – Python est installé sous Windows

Sous Linux

Python est pré-installé sur la plupart des distributions Linux. Cependant, il est possible que vous n'ayez pas la dernière version en date. Pour le vérifier, tapez dans un terminal la commande `python -V`. Elle vous renvoie la version de Python actuellement installée sur votre système, très probablement une **2.x**, comme 2.6 ou 2.7, pour des raisons de

compatibilité. Dans tous les cas, je vous conseille d'installer Python 3.x ; la syntaxe est très proche de Python 2.x mais diffère quand même...

Cliquez sur `download` et téléchargez la dernière version de Python (actuellement « Python 3.7 gzipped source tarball (for Linux, Unix or OS X) »). Ouvrez un terminal, puis rendez-vous dans le dossier où se trouve l'archive :

1. Décompressez l'archive en tapant : `tar -xzf Python-3.7.3.tar.bz2` (cette commande est bien entendu à adapter suivant la version et le type de compression).
2. Attendez quelques instants que la décompression se termine, puis rendez-vous dans le dossier qui vient d'être créé dans le répertoire courant (`Python-3.7.3` dans mon cas).
3. Exécutez le script `configure` en tapant `./configure` dans la console.
4. Une fois que la configuration s'est déroulée, il n'y a plus qu'à compiler en tapant `make` puis `make install` en tant que super-utilisateur.

Sous macOS

Téléchargez la dernière version de Python. Ouvrez le fichier `.dmg` et double-cliquez sur le paquet d'installation `Python.mpkg`.

Un assistant d'installation s'ouvre, laissez-vous guider : Python est maintenant installé !

Lancer Python

Ouf ! Voilà qui est fait !

Bon, en théorie, on commence à utiliser Python dès le prochain chapitre mais, pour que vous soyez un peu récompensés de votre installation exemplaire, voici les différents moyens d'accéder à la ligne de commande Python que nous allons tout particulièrement étudier dans les prochains chapitres.

Sous Windows

Vous avez plusieurs façons d'accéder à la ligne de commande Python, la plus évidente consistant à passer par les menus `Démarrer` > `Tous les programmes` > `Python 3.7` > `Python (Command Line)`. Si tout se passe bien, vous devriez obtenir une magnifique console (figure 1.3). Il se peut que les informations affichées dans la vôtre ne soient pas les mêmes, mais ne vous en inquiétez pas.

Qu'est-ce que c'est que cela ?

On verra plus tard. L'important, c'est que vous ayez réussi à ouvrir la console d'interprétation de Python ; le reste attendra le prochain chapitre.

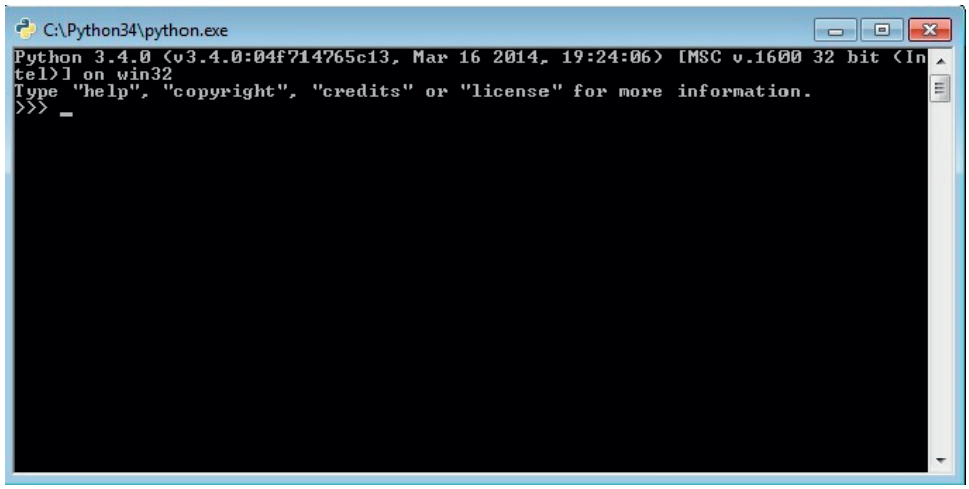

FIGURE 1.3 – La console Python sous Windows

Vous pouvez également passer par la ligne de commande Windows ; à cause des raccourcis, je privilégie en général cette méthode, mais c'est une question de goût. Allez dans le menu `Démarrer`, puis cliquez sur `Exécuter`. Dans la fenêtre qui s'affiche, tapez simplement `python` et la ligne de commande devrait s'afficher de nouveau. Sachez que vous pouvez directement vous rendre dans `Exécuter` en tapant le raccourci `Windows` + `R`.

Sous Windows 8 ou 10, vous pouvez tout simplement rechercher `python`. Cliquez sur le menu `Démarrer` et tapez `python`. Windows devrait vous proposer la version installée en premier choix.

Pour fermer l'interpréteur de commandes Python, tapez `exit()` puis appuyez sur la touche `Entrée`.

Sous Linux

Lorsque vous l'avez installé sur votre système, Python a créé un lien vers l'interpréteur sous la forme **python3.X** (le X étant le numéro de la version installée).

Si, par exemple, vous avez installé Python 3.7, vous pouvez y accéder grâce à la commande :

```
$ python3.7
Python 3.7.3 (default, May 13 2019, 11:56:59)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Pour fermer la ligne de commande Python, n'utilisez pas `CTRL` + `C` mais `CTRL` + `D` (nous verrons plus tard pourquoi).

Sous macOS

Cherchez un dossier `Python` dans le dossier `Applications`. Pour lancer le programme, ouvrez l'application `IDLE` de ce dossier. Vous êtes prêts à passer au concret !

En résumé

— Python est un langage de programmation interprété, à ne pas confondre avec un langage compilé.
— Il permet de créer toutes sortes de programmes, comme des jeux, des logiciels, des progiciels, etc.
— Il est possible d'associer des **bibliothèques** à Python afin d'étendre ses possibilités.
— Il est portable, c'est-à-dire qu'il peut fonctionner sous différents systèmes d'exploitation (Windows, Linux, macOS).

Chapitre 2

Premiers pas avec l'interpréteur de commandes Python

Difficulté :

Après les premières notions théoriques et l'installation de Python, il est temps de découvrir un peu l'interpréteur de commandes de ce langage. Même si ces petits tests vous semblent anodins, vous découvrirez dans ce chapitre les premiers rudiments de la syntaxe du langage et je vous conseille fortement de me suivre pas à pas, surtout si vous êtes face à votre premier langage de programmation.

Comme tout langage de programmation, Python a une syntaxe claire : on ne peut pas lui envoyer n'importe quelle information dans n'importe quel ordre. Nous allons voir ici ce que Python mange... et ce qu'il ne mange pas.

Où est-ce qu'on est, là ?

Pour commencer, je vais vous demander de retourner dans l'interpréteur de commandes Python (je vous ai montré, à la fin du chapitre précédent, comment y accéder en fonction de votre système d'exploitation).

Je vous rappelle les informations qui figurent dans cette fenêtre, même si elles peuvent être différentes chez vous en fonction de votre version et de votre système d'exploitation.

```
1 Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32
  ↪  bit (Intel)] on win32
2 Type "help", "copyright", "credits" or "license" for more information.
3 >>>
```

À sa façon, Python vous souhaite la bienvenue dans son interpréteur de commandes.

Attends, attends. C'est quoi cet interpréteur ?

Souvenez-vous, au chapitre précédent, je vous ai donné une brève explication sur la différence entre langages compilés et langages interprétés. Eh bien, cet interpréteur de commandes va nous permettre de tester directement du code. Je saisis une ligne d'instructions, j'appuie sur la touche Entrée de mon clavier, je regarde ce que me répond Python (s'il me dit quelque chose), puis j'en saisis une deuxième, une troisième... Cet interpréteur est particulièrement utile pour comprendre les bases de Python et réaliser nos premiers petits programmes. Le principal inconvénient, c'est que le code que vous saisissez n'est pas sauvegardé (sauf si vous l'enregistrez manuellement, mais chaque chose en son temps).

Dans la fenêtre que vous avez sous les yeux, l'information qui ne change pas d'un système d'exploitation à l'autre est la série de trois chevrons qui se trouve en bas à gauche des informations : >>>. Ces trois signes signifient : « je suis prêt à recevoir tes instructions ».

Comme je l'ai dit, les langages de programmation respectent une syntaxe claire. Vous ne pouvez pas espérer que l'ordinateur comprenne si, dans cette fenêtre, vous commencez par lui demander : « j'aimerais que tu me codes un jeu vidéo génial ». Et autant que vous le sachiez tout de suite (bien qu'à mon avis, vous vous en doutiez), on est très loin d'obtenir des résultats aussi spectaculaires à notre niveau.

Tout cela pour dire que, si vous saisissez n'importe quoi dans cette fenêtre, la probabilité est grande que Python vous indique, clairement et fermement, qu'il n'a rien compris.

Si, par exemple, vous saisissez « premier test avec Python », vous obtenez le résultat suivant :

```
>>> premier test avec Python
  File "<stdin>", line 1
    premier test avec Python
               ^
SyntaxError: invalid syntax
>>>
```

Eh oui, l'interpréteur parle en anglais et les instructions que vous saisirez, comme pour l'écrasante majorité des langages de programmation, seront également en anglais. Mais pour l'instant, rien de bien compliqué : l'interpréteur vous indique qu'il a trouvé un problème dans votre ligne d'instruction. Il vous indique le numéro de la ligne (en l'occurence la première), qu'il vous répète obligeamment (ceci est très utile quand on travaille sur un programme de plusieurs centaines de lignes). Puis il vous dit ce qui l'arrête, ici : `SyntaxError: invalid syntax`. Limpide n'est-ce pas ? Ce que vous avez saisi est incompréhensible pour Python. Enfin, la preuve qu'il n'est pas rancunier, c'est qu'il vous affiche à nouveau une série de trois chevrons, montrant bien qu'il est prêt à retenter l'aventure.

Bon, c'est bien joli de recevoir un message d'erreur au premier test mais je me doute que vous aimeriez bien voir quelque chose qui fonctionne maintenant. C'est parti donc.

Vos premières instructions : un peu de calcul mental pour l'ordinateur

C'est assez trivial, quand on y pense, mais je trouve qu'il s'agit d'une excellente manière d'aborder pas à pas la syntaxe de Python. Nous allons donc essayer d'obtenir les résultats de calculs plus ou moins compliqués. Je vous rappelle encore une fois qu'exécuter les tests en même temps que moi sur votre machine est une très bonne façon de vous rendre compte de la syntaxe et surtout, de la retenir.

Saisir un nombre

Vous avez pu voir sur notre premier (et à ce jour notre dernier) test que Python n'aimait pas particulièrement les suites de lettres qu'il ne comprend pas. En revanche, l'interpréteur adore les nombres. D'ailleurs, il les accepte sans sourciller, sans une seule erreur :

```
>>> 7
7
>>>
```

D'accord, ce n'est pas extraordinaire. On saisit un nombre et l'interpréteur le renvoie. Pourtant, dans bien des cas, ce simple retour indique que l'interpréteur a bien compris et que votre saisie est en accord avec sa syntaxe. De même, vous pouvez saisir des nombres à virgule.

```
>>> 9.5
9.5
>>>
```

Attention : on utilise ici la notation anglo-saxonne, c'est-à-dire que le point remplace la virgule. La virgule a un tout autre sens pour Python ; prenez donc cette habitude dès maintenant.

Il va de soi que l'on peut tout aussi bien saisir des nombres négatifs (faites d'ailleurs l'essai).

Opérations courantes

Bon, il est temps d'apprendre à utiliser les principaux opérateurs de Python, qui vont vous servir pour la grande majorité de vos programmes.

Addition, soustraction, multiplication, division

Pour effectuer ces opérations, on utilise respectivement les symboles +, -, * et /.

```
>>> 3 + 4
7
>>> -2 + 93
91
>>> 9.5 + 2
11.5
>>> 3.11 + 2.08
5.1899999999999995
>>>
```

Pourquoi ce dernier résultat approximatif ?

Python n'y est pas pour grand-chose. En fait, le problème vient en grande partie de la façon dont les nombres à virgule sont écrits dans la mémoire de votre ordinateur. C'est pourquoi, en programmation, on préfère travailler autant que possible avec des nombres entiers. Cependant, vous remarquerez que l'erreur est infime et qu'elle n'aura pas de réel impact sur les calculs. Les applications qui ont besoin d'une précision mathématique à toute épreuve essayent de pallier ces défauts par d'autres moyens mais ici, ce ne sera pas nécessaire.

Faites également des tests pour la soustraction, la multiplication et la division : il n'y a rien de difficile.

Division entière et modulo

Si vous avez pris le temps de tester la division, vous vous êtes rendu compte que le résultat est donné avec une virgule flottante.

```
>>> 10 / 5
2.0
>>> 10 / 3
3.3333333333333335
>>>
```

Il existe deux autres opérateurs qui permettent de connaître le résultat d'une division entière et le reste de cette division.

Le premier opérateur utilise le symbole « // ». Il renvoie la partie entière d'une division.

```
>>> 10 // 3
3
>>>
```

L'opérateur « % », que l'on appelle le « modulo », retourne le reste de la division.

```
>>> 10 % 3
1
>>>
```

Ces notions de *partie entière* et de *reste de division* ne sont pas bien difficiles à comprendre et vous serviront très probablement par la suite.

Si vous avez du mal à en saisir le sens, sachez donc que :

- La partie entière de la division de 10 par 3 est le résultat de cette division, sans tenir compte des chiffres au-delà de la virgule (en l'occurence, 3).
- Pour obtenir le modulo d'une division, on « récupère » son reste. Dans notre exemple, $10/3 = 3$ et il reste 1. Une fois que l'on a compris cela, ce n'est pas bien compliqué.

Souvenez-vous bien de ces deux opérateurs, et surtout du modulo « % », dont vous aurez besoin dans vos programmes futurs.

En résumé

- L'interpréteur de commandes Python permet de tester du code au fur et à mesure qu'on l'écrit.
- L'interpréteur Python accepte des nombres et est capable d'effectuer des calculs.
- Un nombre décimal s'écrit avec un point et non une virgule.
- Les calculs impliquant des nombres décimaux donnent parfois des résultats approximatifs. C'est pourquoi on préfèrera, dans la mesure du possible, travailler avec des nombres entiers.

Chapitre 3

Le monde merveilleux des variables

Difficulté : 

Au chapitre précédent, vous avez saisi vos premières instructions en langage Python, bien que vous ne vous en soyez peut-être pas rendu compte. Il est également vrai que les instructions saisies auraient fonctionné dans la plupart des langages. Ici, cependant, nous allons commencer à approfondir un petit peu la syntaxe du langage, tout en découvrant un concept important de la programmation : les variables.

Ce concept est essentiel et vous ne pouvez absolument pas faire l'impasse dessus. Cependant, je vous rassure, il n'y a rien de compliqué ; que de l'utile et de l'agréable.

Qu'est-ce qu'une variable ? Et à quoi cela sert-il ?

Les variables sont l'un des concepts qui se retrouvent dans tous les langages de programmation. Autant dire que sans variable, on ne peut pas programmer ; et ce n'est pas une exagération.

Qu'est-ce qu'une variable ?

Une variable est une donnée de votre programme, stockée dans votre ordinateur. C'est un code alpha-numérique que vous allez lier à une donnée de votre programme, afin de pouvoir l'utiliser à plusieurs reprises et faire des calculs un peu plus intéressants avec. C'est bien joli de savoir faire des opérations mais, si on ne peut pas stocker le résultat quelque part, cela devient très vite ennuyeux.

Voyez la mémoire de votre ordinateur comme une grosse armoire avec plein de tiroirs. Chaque tiroir peut contenir une donnée ; certaines de ces données seront des variables de votre programme.

Comment cela fonctionne-t-il ?

Le plus simplement du monde. Vous allez dire à Python : « je veux que, dans une variable que je nomme `âge`, tu stockes mon âge, pour que je puisse le retenir (si j'ai la mémoire très courte), l'augmenter (à mon anniversaire) et l'afficher si besoin est ».

Comme je vous l'ai dit, on ne peut pas passer à côté des variables. Vous ne voyez peut-être pas encore tout l'intérêt de stocker des informations de votre programme et pourtant, si vous ne stockez rien, vous ne pouvez pratiquement rien faire.

En Python, pour donner une valeur à une variable, il suffit d'écrire `nom_de_la_variable = valeur`.

Une variable doit respecter quelques règles de syntaxe incontournables :

1. Le nom de la variable ne peut être composé que de lettres, majuscules ou minuscules, de chiffres et du symbole souligné « _ » (appelé *underscore* en anglais).
2. Le nom de la variable ne peut pas commencer par un chiffre.
3. Le langage Python est sensible à la casse, ce qui signifie que des lettres majuscules et minuscules ne constituent pas la même variable (la variable `AGE` est différente de `aGe`, elle-même différente de `age`).
4. Le nom de la variable peut contenir des accents. Notez cependant que Python fera alors la différence entre `age` et `âge`.

L'utilisation des accents dans des variables est encore relativement récente. À l'heure actuelle, peu de langages de programmation l'autorisent et beaucoup de programmeurs, que ce soit en Python ou ailleurs, recommandent d'éviter les accents dans les noms de variables. Ce n'est pas mon cas. Python le permet, le français comporte des accents, autant les utiliser. Rien ne vous empêche de retirer les accents dans les exemples que je vous donne, bien entendu.

Au-delà de ces règles de syntaxe incontournables, il existe des conventions définies par les programmeurs eux-mêmes. L'une d'elles, que j'utilise assez souvent, consiste à écrire la variable en minuscules et à remplacer les espaces éventuels par le caractère souligné « _ ». Si je dois créer une variable contenant mon âge, elle se nommera donc `mon_âge`. Une autre convention utilisée consiste à passer en majuscule l'initiale de chaque mot constituant le nom de la variable, sauf le premier ; cela donnerait par exemple `monÂge`.

Vous pouvez appliquer la convention qui vous plaît, ou même en créer une bien à vous, mais essayez de rester cohérent et de n'en utiliser qu'une seule. En effet, il est essentiel de pouvoir vous repérer dans vos variables dès que vous commencez à travailler sur des programmes volumineux.

Ainsi, si je veux associer mon âge à une variable, la syntaxe sera :

```
1 mon_âge = 21
```

L'interpréteur vous affiche aussitôt trois chevrons sans aucun message. Cela signifie qu'il a bien compris et qu'il n'y a eu aucune erreur.

Sachez qu'on appelle cette étape *l'affectation de valeur à une variable* (parfois raccourci en « affectation de variable »). On dit en effet qu'on a affecté la valeur 21 à la variable `mon_âge`.

On peut afficher la valeur de cette variable en la saisissant simplement dans l'interpréteur de commandes.

```
1 >>> mon_âge
2 21
3 >>>
```

Les espaces séparant « = » du nom et de la valeur de la variable sont facultatifs. Je les mets pour des raisons de lisibilité.

Bon, c'est bien joli tout cela, mais qu'est-ce qu'on fait avec cette variable ?

Eh bien, tout ce que vous avez déjà fait au chapitre précédent, mais cette fois en utilisant la variable comme un nombre à part entière. Vous pouvez même affecter à d'autres variables des valeurs obtenues en effectuant des calculs sur la première et c'est là toute la puissance de ce mécanisme.

Essayons par exemple d'augmenter de 2 la variable `mon_âge`.

```
>>> mon_âge = mon_âge + 2
>>> mon_âge
23
>>>
```

Encore une fois, lors de l'affectation de la valeur, rien ne s'affiche, ce qui est parfaitement normal.

Maintenant, essayons d'affecter une valeur à une autre variable d'après la valeur de `mon_âge`.

```
>>> mon_âge_x2 = mon_âge * 2
>>> mon_âge_x2
46
>>>
```

Encore une fois, je vous invite à tester en long, en large et en travers cette possibilité. Le concept n'est pas compliqué mais extrêmement puissant. De plus, comparé à certains langages, affecter une valeur à une variable est extrêmement simple. Si la variable n'est pas créée, Python s'en charge automatiquement. Si la variable existe déjà, l'ancienne valeur est supprimée et remplacée par la nouvelle. Quoi de plus simple ?

Certains mots-clés de Python sont **réservés**, c'est-à-dire que vous ne pouvez pas vous en servir pour nommer des variables.

En voici la liste pour Python 3 :

`and`	`continue`	`finally`	`is`	`raise`
`as`	`def`	`for`	`lambda`	`return`
`assert`	`del`	`from`	`None`	`True`
`async`	`elif`	`global`	`nonlocal`	`try`
`await`	`else`	`if`	`not`	`while`
`break`	`except`	`import`	`or`	`with`
`class`	`False`	`in`	`pass`	`yield`

Notez que les mots-clés `async` et `await` ont été ajoutés dans la version 3.7 de Python et n'apparaissent pas sur la plupart des tableaux récapitulatifs que l'on trouve sur Internet, pour l'heure. Ne soyez donc pas surpris.

Ces mots-clés sont utilisés par Python, vous ne pouvez pas construire de variables portant ces noms. Vous allez découvrir dans la suite de ce cours la majorité de ces mots-clés et comment ils s'utilisent.

Les types de données en Python

Là se trouve un concept très important, que l'on retrouve dans beaucoup de langages de programmation. Ouvrez grand vos oreilles, ou plutôt vos yeux, car vous devrez être parfaitement à l'aise avec ce concept pour continuer la lecture de ce livre. Rassurez-vous toutefois, du moment que vous êtes attentifs, il n'y a rien de compliqué à comprendre.

Qu'entend-on par « type de donnée » ?

Jusqu'ici, vous n'avez travaillé qu'avec des nombres. Et, s'il faut bien avouer qu'on ne fera que très rarement un programme sans aucun nombre, c'est loin d'être la seule donnée que l'on peut utiliser en Python. À terme, vous serez même capables de créer vos propres types de données, mais n'anticipons pas.

Python a besoin de connaître quels types de données sont utilisés pour savoir quelles opérations il peut effectuer avec. Dans ce chapitre, vous allez apprendre à travailler avec des chaînes de caractères et multiplier une chaîne de caractères ne se fait pas du tout comme la multiplication d'un nombre. Pour certains types de données, la multiplication n'a d'ailleurs aucun sens. Python associe donc à chaque donnée un type, qui va définir les opérations autorisées sur cette donnée en particulier.

Les différents types de données

Nous n'allons voir ici que les incontournables et les plus faciles à manier. Des chapitres entiers seront consacrés aux types plus complexes.

Les nombres entiers

Eh oui, Python différencie les entiers des nombres à virgule flottante !

Pourquoi cela ?

Initialement, c'est surtout pour une question de place en mémoire mais, pour un ordinateur, les opérations que l'on effectue sur des nombres à virgule ne sont pas les mêmes que celles sur les entiers et cette distinction reste encore d'actualité de nos jours.

Le type entier se nomme `int` en Python (qui correspond à l'anglais « integer », c'est-à-dire entier). La forme d'un entier est un nombre sans virgule.

```
1 3
```

Nous avons vu au chapitre précédent les opérations que l'on pouvait effectuer sur ce type de données et, même si vous ne vous en souvenez pas, les deviner est assez élémentaire.

Les nombres flottants

Les flottants sont les nombres à virgule. Ils se nomment `float` en Python (ce qui signifie « flottant » en anglais). N'oubliez pas de remplacer la virgule par un point. Si vous voulez qu'un nombre sans partie décimale soit considéré par le système comme un flottant, ajoutez-lui une partie flottante de 0 (exemple **52.0**).

```
3.152
```

Les nombres après la virgule ne sont pas infinis, puisque rien n'est infini en informatique. Néanmoins, la précision est assez importante pour travailler sur des données très fines.

Les chaînes de caractères

Heureusement, les types de données disponibles en Python ne sont pas limités aux seuls nombres, bien loin de là. Le dernier type « simple » que nous verrons dans ce chapitre est la chaîne de caractères. Ce type de donnée sert à stocker une série de lettres, voire une phrase.

On peut écrire une chaîne de caractères de différentes façons :

- entre guillemets (`"ceci est une chaîne de caractères"`) ;
- entre apostrophes (`'ceci est une chaîne de caractères'`) ;
- entre triples guillemets (`"""ceci est une chaîne de caractères"""`) ;
- plus rarement, entre triples apostrophes (`'''ceci est une chaîne de caractères'''`).

On peut, à l'instar des nombres (et de tous les types de données) stocker une chaîne de caractères dans une variable (`ma_chaîne = "Bonjour, la foule !"`)

Si vous utilisez les délimiteurs simples (le guillemet ou l'apostrophe) pour encadrer une chaîne de caractères, il se pose le problème des guillemets ou apostrophes que peut contenir ladite chaîne. Par exemple, si vous tapez `chaîne = 'J'aime Python !'`, vous obtenez le message suivant :

```
  File "<stdin>", line 1
    chaîne = 'J'aime Python !'
                   ^
SyntaxError: invalid syntax
```

Ceci est dû au fait que l'apostrophe de « J'aime » est considérée par Python comme la fin de la chaîne et qu'il ne sait pas quoi faire de tout ce qui se trouve au-delà. Pour pallier ce problème, il faut **échapper** les apostrophes se trouvant au cœur de la chaîne. On insère ainsi une barre oblique inverse « \ » avant les apostrophes contenues dans le message.

```
chaîne = 'J\'aime Python !'
```

On doit également échapper les guillemets si on les utilise comme délimiteurs.

```
chaîne2 = "\"Le seul individu formé, c'est celui qui a appris comment
↪  apprendre (...)\" (Karl Rogers, 1976)"
```