ITC

INFORMATIQUE

TRONC COMMUN

PSI-MP-PC

Steeven **JANNY**
Wenqi **SHU-QUARTIER-DIT-MAIRE**
Théodore **CHERRIÈRE**
Jeanne **REDAUD**

ellipses

Cours

Introduction à Python

1 Présentation de Python

1.1 Bref historique

Python est certainement l'un des langages les plus populaires de cette décennie. Depuis son adoption par l'Éducation Nationale en 2013, sa cote de popularité n'a cessé d'augmenter, dépassant même celle du Java en 2020. Il domine largement dans les laboratoires de recherche et développement des entreprises en offrant un outil gratuit, puissant, très polyvalent et pourtant très simple à prendre en main. Python permet de contourner l'apprentissage long et fastidieux des subtilités de la programmation traditionnelle pour permettre à l'ingénieur d'écrire rapidement un code fonctionnel.

Le langage est administré par la *Python Software Fundation* qui publie régulièrement des mises à jour répondant aux PEP (*Python Enhancement Proposals*), c'est-à-dire des suggestions d'améliorations proposées par la communauté. Dans ce livre, nous utiliserons la dernière version disponible au moment de sa rédaction : Python 3.9. Les programme devraient rester compatibles avec les futures versions du langage.

1.2 Principe de fonctionnement

Python, en plus d'avoir le bon goût d'être gratuit, possède de nombreux avantages par rapport à ses concurrents. C'est un langage :

- **De haut niveau** : un code Python fait abstraction du fonctionnement interne d'un ordinateur et se focalise sur le problème à résoudre. Sa syntaxe utilise de nombreux mots anglais du langage courant, ce qui facilite sa lecture et son écriture.
- **Multi-plateforme** : un programme développé en Python pourra être exécuté sur une machine Windows, Linux ou MacOS sans aucune modification.
- **Interprété** : un code Python est exécuté par un interpréteur, une sorte de programme informatique qui exécute les commandes du langage *à la volée*. Cela s'oppose aux langages compilés comme le C ou le Java.

2 Environnement de développement

Pour développer en Python, il faut installer son interpréteur. Plusieurs options sont possibles : la plus simple consiste à télécharger la dernière version de Python sur le site de la fondation (`www.python.org`), puis de l'installer. Nous recommandons cependant l'utilisation d'Anaconda (`www.anaconda.com`). Il s'agit d'une version de Python incluant par défaut de nombreuses bibliothèques scientifiques indispensables aux cours du tronc commun d'Informatique en CPGE.

2.1 L'interpréteur Python

Une fois Python installé, l'interpréteur est immédiatement accessible dans sa forme la plus élémentaire : depuis une console, en ligne de commande. La figure ci-dessous donne un exemple d'utilisation. Une ligne de code commence par trois chevrons >>> et la réponse de l'interpréteur s'inscrit en dessous de celle-ci.

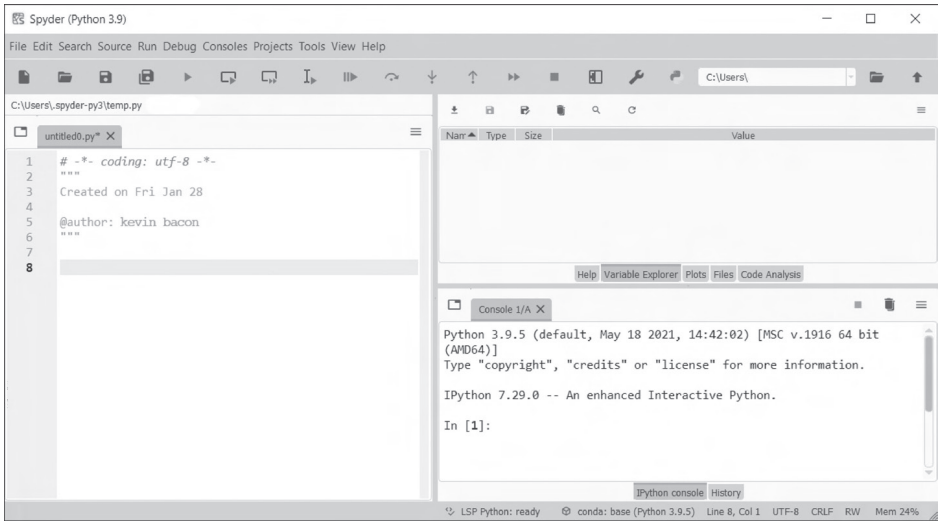
```
Python 3.8.3 (default, Jul  2 2020, 11:26:31)
[Clang 10.0.0 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>> variable = 5
>>> variable = variable + 1
>>> print(variable)
6
>>>
```

Exemple de code en console Python. Chaque ligne est exécutée l'une après l'autre.

Bien sûr, il serait très difficile de développer un programme complexe directement sur la console, c'est pourquoi nous utilisons des **environnements de développement** (*Integrated Development Environnment*, ou *IDE* en anglais) pour écrire nos programmes dans des fichiers textes, qui seront ensuite lus et exécutés par l'interpréteur.

2.2 Pyzo et Spyder

- **Spyder** est un IDE très couramment utilisé pour l'enseignement de Python. Celui-ci dispose des fonctionnalités indispensables à la programmation, sans superflu. Il existe de nombreux plugins lui permettant d'évoluer selon les besoins et le niveau technique du développeur.
- **Pyzo** est un projet *open-source* (i.e. le code source du programme est librement disponible et modifiable) développé dans le but de simplifier l'accès à Python dans les milieux éducatifs. Son interface se compose d'une zone d'édition de code, ainsi que d'une console permettant de l'exécuter.

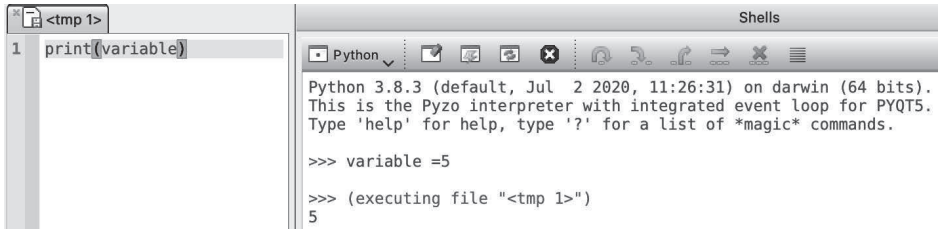


IDE Spyder : l'éditeur de code se trouve à gauche, et la console Python à droite.

Les exercices et exemples de ce livre ne sont pas liés à un IDE particulier, vous êtes libre de choisir celui qui vous conviendra le mieux. Nous vous recommandons d'utiliser Spyder.

3 Lien entre IDE et interpréteur

La relation entre un fichier de code et l'interpréteur est assez subtile de prime abord. Il est important de bien comprendre leur intrication, car celle-ci est souvent source d'erreurs.



Exemple simple du lien entre le ficher de code et l'interpréteur

Voici les étapes qui ont conduit à ce résultat :

1. **Création d'une variable** `variable = 5` dans l'interpréteur (à droite).
2. **Instruction** : Affiche `variable` dans l'éditeur (à gauche).
3. **Exécution du fichier** `tmp1` (en appuyant sur le bouton "Exécuter" dans Pyzo").

Notez que la variable n'est pas redéfinie dans le fichier de code. L'interpréteur a **gardé en mémoire l'instruction précédente** et réutilise la valeur. Pour un débutant, il est courant d'oublier ce comportement.

Imaginons que nous développions une simulation thermique. Nous déclarons une variable `temperature = 20` qui stocke la température initiale de notre modèle et utilisons cette variable à divers endroits du code. Nous exécutons régulièrement le programme dans l'interpréteur (pour tester son bon fonctionnement). À un certain point du développement, nous décidons de renommer la variable `T=25`, et de changer la valeur de la température initiale qui passe de 20 °C à 25 °C.

Si, par inattention, nous oublions de remplacer `temperature` par `T` dans une ligne de code, l'interpréteur utilisera l'ancienne valeur de la variable stockée en mémoire, et non la nouvelle valeur `T=25`. Celui conduira donc à un mauvais résultat, dont nous ne prendrons conscience qu'en **redémarrant l'interpréteur**.

4 Les messages d'erreurs

Lorsque l'interpréteur rencontre une erreur dans votre code, l'exécution s'arrête et affiche un message d'erreur en précisant le numéro de la ligne qui pose problème. Ces messages sont souvent assez explicites : les connaître permet de corriger rapidement votre code. En voici quelques-uns avec leurs causes fréquentes :

- `NameError` : vous utilisez dans votre programme une variable qui n'est pas définie.
- `SyntaxError` : en général, il s'agit d'une faute de frappe, de l'oubli d'une parenthèse, ou du symbole " :" par exemple.
- `FileNotFoundError` : vous essayez d'ouvrir un fichier qui ne se trouve pas à l'endroit que vous avez indiqué.
- `IndexError` : vous essayez d'accéder à un élément d'une liste dont l'indice est supérieur au nombre d'éléments dans cette liste.
- `IndentationError` : il s'agit d'un problème d'indentation, vérifiez vos espaces et tabulations.

Il en existe bien d'autres. Vous apprendrez à les reconnaître et à les comprendre en pratiquant les exercices de ce livre.

5 Librairies complémentaires

Python est installé de base avec plusieurs librairies dites *standards*. Une librairie (ou bibliothèque) est un ensemble de fonctions pré-implémentées permettant de répondre à un besoin particulier. Par exemple, la librairie `math` intègre les fonctions sin, cos et tan ainsi que la valeur π. Ces librairies doivent être importées manuellement dans le code pour être utilisées.

En plus des librairies standards, il est possible d'en installer d'autres, en fonction des besoins et du type d'applications que l'on souhaite concevoir. Parmi les plus populaires en sciences, on trouve :

- **Numpy** : quasiment indispensable pour toutes les applications scientifiques, cette librairie contient tous les outils nécessaires au calcul mathématique.
- **SciPy** : construite comme une extension de `NumPy`, elle contient des fonctions mathématiques de plus haut niveau, comme des solveurs d'équations.
- **Matplotlib** : c'est une librairie d'affichage permettant de tracer différents types de graphes. C'est une librairie très utilisée car polyvalente et simple à appréhender.
- **TensorFlow**, **PyTorch** et **Scikit-Learn** sont des librairies très couramment utilisées en intelligence artificielle.

La plupart des librairies courantes sont pré-installées avec Anaconda. Si vous avez choisi d'installer Python depuis le site officiel, il sera nécessaire d'installer les librairies à la main, en utilisant l'utilitaire `pip`. Il s'agit d'un outil en ligne de commande : il s'utilise dans la console de votre système d'exploitation (attention à ne pas confondre avec la console Python).

```
pip install numpy scipy matplotlib
```

Dans le cas où vous auriez à installer une librairie dans Anaconda, l'utilitaire s'appelle `conda` et s'utilise de la même manière.

Cours

Les variables informatiques

1 Qu'est ce qu'une variable ?

Définition : Les *variables* sont les briques élémentaires de la programmation. On représente souvent la mémoire d'un ordinateur sous la forme d'un tableau à deux colonnes : la première contient le nom d'une case mémoire, et la seconde la valeur qui y est stockée. Un programme informatique manipule ce tableau, en modifiant et créant des cases.

Adresse mémoire	Valeur
# 12048	12
# 12049	'a'
...	...

Utiliser directement les adresses mémoires n'est pas chose aisée (même si cela est possible). On préfère nommer distinctement les noms des cases que nous utilisons. Voilà ce qu'est une variable : la donnée d'une case mémoire dans l'ordinateur représentée par un **nom**.

Telles les variables en mathématiques, les variables informatiques peuvent changer de valeur. Dire que la variable x est égale à 10 signifie qu'à ce moment précis, on a une variable dont le nom est x et qu'à l'emplacement mémoire réservé se trouve stocké le nombre 10. Au cours de son utilisation, si on veut changer la valeur de x, on est alors obligé de définir explicitement le nouveau contenu de la variable : c'est **l'affectation**.

2 Affectation

Définition : L'**affectation** d'une variable, c'est l'opération qui consiste à donner un nom à une case mémoire, et à y stocker une donnée.

Le nom des variables peut être constitué de lettres majuscules et minuscules (sauf les caractères spéciaux tels que 'é', 'à', etc), de chiffres (sauf comme premier caractère) et du symbole '_' (*underscore*).

Il n'y a pas de limite de longueur pour un nom de variable. Toutefois, comme il est appelé à être conservé et réutilisé, il est important de choisir des noms cohérents et explicites pour éviter les confusions. Il vaut mieux éviter d'utiliser une lettre unique ou des mots trop longs. Bien nommer ses variables (et par la suite ses fonctions) est capital. Pour remplacer l'espace, on peut séparer les composantes du nom de la variable par '_' (`ma_variable`) ou utiliser des majuscules (`maVariable`). Il faut noter que Python distingue les minuscules des majuscules (on dit que Python est sensible à la **casse**) : `ma_variable`, `Ma_variable` et `ma_Variable` représentent trois variables différentes.

Piège

Attention, il existe des **mots réservés** (*built-in*), qui ne peuvent pas être utilisés comme noms de variables. Il s'agit de mots standards, tels que les opérateurs logiques (`and, or, not...`), les instructions (`try, pass, except, import`...), ou des noms de fonctions déjà implémentées dans Python (`print, min, len`...). Faites-y très attention !

Pour **affecter** une valeur à une variable, on utilise le symbole '='. La partie située à gauche du signe d'affectation est le nom de la variable, et elle doit respecter les règles citées précédemment, ou cela entraînera une erreur de syntaxe. La partie à droite est la valeur qu'on souhaite affecter : elle peut être le résultat d'une opération, qui peut même contenir la variable elle-même.

Console Python

```
>>> 1variable = 1 # le nom d'une variable ne peut pas commencer par un chiffre
SyntaxError: invalid syntax
>>> 3 = 1 # une variable ne peut pas s'appeler '3'
SyntaxError: cannot assign to literal
>>> ma variable = 1 # le nom d'une variable ne peut pas contenir d'espace
SyntaxError: invalid syntax
>>> var = 1 # pas d'erreur : on stocke 1 dans la variable 'var'
>>> var = 5 + 2
>>> var # on affiche la nouvelle valeur de 'var'
7
>>> var = var + 1 # on ajoute 1 à var et on stocke le résultat dans var
>>> var
8
```

On peut ré-utiliser une variable déjà initialisée. L'ancienne valeur associée est alors **écrasée**, comme le montre l'exemple ci-dessus avec `var` : elle était initialisée avec la valeur 1, puis on lui a affecté la valeur 7, écrasant ainsi la valeur précédente. Puis on y a ajouté 1 et on a affecté le résultat à `var` de nouveau.

Notez que nous ferons la distinction entre la **déclaration** d'une variable et son affectation. La déclaration est en fait la première affectation de celle-ci. On emploie également le terme d'**initialisation**.

Astuce

Il est possible en Python d'affecter des valeurs à plusieurs variables en même temps, mais aussi d'affecter la même valeur à plusieurs variables. On parle d'**affectation multiple**. On peut utiliser ce principe pour échanger les valeurs de deux variables sans passer par une variable temporaire.

Python

```
1 var1, var2, var3 = 1, 2, 3;
2 var1 = var2 = var3 = 1;
3 var1, var2 = var2, var1; # permet de ne pas utiliser de variable temporaire
```

Il existe également le mot-clé `None` (abréviation de '`NoneType`'), qui ne correspond à rien. Il permet d'initialiser une variable sans lui affecter de valeur.

3 Type

Une variable peut être de différents **types**, selon la nature de l'objet qu'elle stocke : un nombre entier (de type `'int'`), un réel (ou flottant en informatique `'float'`), un nombre complexe (`'complex'`), une chaîne de caractères (`'str'`), un booléen (`'bool'`), etc... On y accède par la commande `type()`.

Console Python

```
>>> var = 'hello' # on stocke le mot "hello" dans la variable 'var'
>>> type(var) # renvoie le type de la variable 'var' : c'est une chaine de caracteres
<class 'str'>
```

Console Python

```
>>> var = 3.14 # on affecte la valeur 3.14 à la variable 'var'
>>> type(var) # renvoie à nouveau le type de la variable 'var' : cette fois c'est un nombre flottant
<class 'float'>
>>> int(var) # renvoie le nombre obtenu en convertissant 'var' en entier
3
```

De nombreux langages sont dits **typés** : une variable doit être déclarée dans un certain type et ne doit pas en changer. C'est le cas de Java et C++. Comme l'illustre le code ci-dessus, Python va *deviner*, lors de la déclaration, le type de la variable selon la valeur affectée, et allouer en conséquence un espace de mémoire pour stocker la valeur. De manière générale, Python attribue également automatiquement le type le plus approprié lors du résultat d'une opération. Le type d'une variable peut donc évoluer au fil des affectations : on parle de **typage dynamique**.

Puisque le type d'une variable n'est pas fixe en Python, vous pouvez convertir une variable numérique flottante ou une chaîne de caractères en entier avec la fonction `int`. De même, `float()` permet de convertir en nombre flottant, `ord()` convertit un entier en un caractère, `str()` convertit en chaîne de caractères, etc. Le chapitre suivant, "Les différents types d'objets" (p.17), détaillera cet aspect des variables.

Chaque type de variable n'occupe pas la même taille en mémoire. Lorsqu'on modifie le type d'une variable, comme lors d'une affectation, Python modifie l'emplacement mémoire : c'est ainsi qu'il gère par exemple le fait qu'une variable entière devenue une variable flottante prend plus de place en mémoire. Nous développerons ce point au deuxième semestre, dans le chapitre "Représentation informatique des nombres" (p.213).

4 Identifiant

Chaque variable possède un identifiant unique qui lui est associé, et qui correspond grossièrement à son adresse en mémoire. On y accède par la commande `id()`.

Console Python

```
>>> var1 = 2
>>> id(var1)
505304408
>>> var1 += 1 # c'est équivalent à écrire var1 = var1 + 1 : à présent, var1 contient la valeur 3
>>> id(var1)
505304424
>>> var2 = var1 # var2 aussi contient la valeur 3
>>> id(var2)
505304424
```

5 Affichage

Il existe plusieurs moyens d'afficher la valeur d'une variable. Dans la console Python, il suffit de taper le nom de la variable, puis d'appuyer sur la touche Entrée.

Console Python

```
>>> x = 5
>>> x
5
```

Mais lorsqu'on exécute un programme et qu'on souhaite afficher le contenu d'une variable, il faut utiliser la fonction `print()`. On peut bien sûr aussi l'utiliser en console. Cette fonction affiche le contenu d'une variable quel que soit son type, ou d'un ensemble de variables séparées par des virgules.

Console Python

```
>>> nom, age = 'Jeanne', 7
>>> print(nom)
Jeanne
>>> print(nom, age)
Jeanne 7
>>> print("Je m'appelle", nom, "et j'ai", age, "ans")
Je m'appelle Jeanne et j'ai 7 ans
```

À la dernière ligne, Python a écrit une phrase complète en remplaçant les variables `nom` et `age` par leur valeur, mais ce n'est pas très pratique à coder avec tous ces guillemets. Afin de spécifier des valeurs dans une chaîne de caractères, on peut aussi utiliser **l'écriture formatée** . Pour cela, il suffit de faire précéder les chaînes de caractères par la lettre `f`. On peut également spécifier le format d'affichage des variables (nombre de décimales, justifié à gauche ou à droite...). Une autre alternative consiste à utiliser la méthode `.format()`.

Console Python

```
>>> nom, age = 'Jeanne', 7
>>> print(f"Je m'appelle {nom} et j'ai {age} ans")
Je m'appelle Jeanne et j'ai 7 ans
>>> print("Je m'appelle {} et j'ai {} ans".format(nom, age)) # strictement équivalent
à l'instruction précédente
Je m'appelle Jeanne et j'ai 7 ans
>> prix = 1.28456
>>> txt = "la pomme coute {prix:.2f} euros" # fixe le nombre de décimales affichées à 2
>>> print(txt)
la pomme coute 1.28 euros
```

Enfin, dans l'exemple précédent, remarquez que la fonction `print` a ajouté un espace à chaque fois qu'on utilisait le séparateur "," entre les variables qu'on affiche. Nous pouvons modifier ce comportement en passant l'argument `sep` à la fonction `print()` :

Console Python

```
>>> x, y = 5, 12
>>> print(x, y, sep="+")
5+12
```