Initiation à la programmation python

Dr. Séna APEKE

UL/BERIIA

20/05/2022

1. **Introduction**

2. **Les commentaires**
   - Notion de bloc d'instructions et d'indentation

3. **es variables**
   - Les types de variables
   - Nommage
   - Écriture scientifique

4. **Les opérations**
   - Opérations sur les types numériques
   - Opérations sur les chaînes de caractères
   - Opérations illicites
   - Affichage
   - Autres structures de données

Introduction

Python est un langage interprété, c'est-à-dire que chaque ligne de code est lue puis interprétée afin d'être exécutée par l'ordinateur. Pour vous en rendre compte, ouvrez un shell puis lancez la commande (après installation de python évidemment) :
Python ou **python3**

La commande précédente va lancer l'interpréteur Python. Vous devriez obtenir quelque chose de ce style pour ubuntu.
Python 3.6.9 (default, Jan 26 2021, 15:33:00) [GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.

Les blocs :

- $C : \backslash Users \backslash yawa >$ pour Windows,
- Mac − de − yawa : Downloads$ pour Mac OS,
- sena@sena − apeke :∼ $ pour Linux.

représentent l'invite de commande de votre shell. Par la suite, cette invite de commande sera représentée simplement par le caractère $, que vous soyez sous Windows, Mac ou Linux. Le triple chevron >>> est l'invite de commande (prompt en anglais) de l'interpréteur Python. Ici, Python attend une commande que vous devez saisir au clavier. Tapez par exemple l'instruction : **print("Hello world!")** puis validez cette commande en appuyant sur la touche Entrée.

Python a exécuté la commande directement et a affiché le texte Hello world!. Il attend ensuite votre prochaine instruc- tion en affichant l'invite de l'interpréteur Python (≫). En résumé, voici ce qui a dû apparaıˆtre sur votre écran :

```
>>> print (" Hello world !")
Hello world !
>>>
```

À ce stade, vous pouvez entrer une autre commande ou bien quitter l'interpréteur Python, soit en tapant la commande exit() puis en validant en appuyant sur la touche Entrée, soit en pressant simultanément les touches Ctrl et D sous Linux et Mac OS X ou Ctrl et Z puis Entrée sous Windows.

Premier programme

L'interpréteur présente vite des limites dès lors que l'on veut exécuter une suite d'instructions plus complexe. Comme tout langage informatique, on peut enregistrer ces instructions dans un fichier, que l'on appelle communément un script (ou programme) Python. Pour reprendre l'exemple précédent, ouvrez un éditeur de texte et entrez le code suivant : print("Hello world!") Ensuite, enregistrez votre fichier sous le nom test.py, puis quittez l'éditeur de texte.

Remarque : l'extension de fichier standard des scripts Python est .py. Pour exécuter votre script, ouvrez un shell et entrez la commande : python test.py Vous devriez obtenir un résultat similaire à ceci :

$ python test.py
Hello world !

Si c'est bien le cas, bravo ! Vous avez exécuté votre premier programme Python.

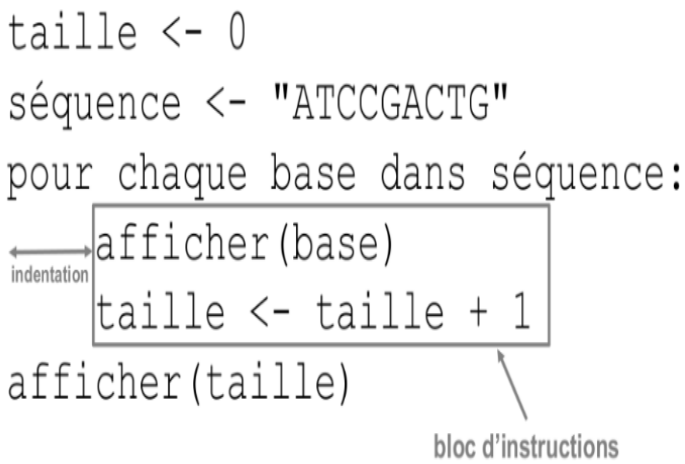
Les commentaires

Dans un script, tout ce qui suit le caractère # est ignoré par Python jusqu'à la fin de la ligne et est considéré comme un commentaire. Pour les commentaires sur plusieurs lignes, on utilise """ ... """.
Voici un exemple :

```
"""
Blablabla. . . ligne 1
Blablabla. . . . ligne 2
Blablabla. . . . ligne 3
Blablabla. . . . ligne 4
"""
```

Notion de bloc d'instructions et d'indentation

En programmation, il est courant de répéter un certain nombre de choses (avec les boucles) ou d'exécuter plusieurs instructions si une condition est vraie (avec les tests,). Par exemple, imaginons que nous souhaitions afficher chacune des bases d'une séquence d'ADN, les compter puis afficher le nombre total de bases à la fin. Nous pourrions utiliser l'algorithme présenté en pseudo-code dans la figure 1.



Pour chaque base de la séquence ATCCGACTG, nous souhaitons effectuer deux actions : d'abord afficher la base puis compter une base de plus.

Pour indiquer cela, on décalera vers la droite ces deux instructions par rapport à la ligne précédente (pour chaque base [...]). Ce décalage est appelé indentation, et l'ensemble des lignes indentées constitue un bloc d'instructions.

Une fois qu'on aura réalisé ces deux actions sur chaque base, on pourra passer à la suite, c'est-à-dire afficher la taille de la séquence.

Pratiquement, l'indentation en Python doit être homogène (soit des espaces, soit des tabulations, mais pas un mélange des deux). Une indentation avec 4 espaces est le style d'indentation recommandé.

Définition

Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.

En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps. Pour vous en convaincre, testez les instructions suivantes après avoir lancé l'interpréteur :

```
>>> x =4
>>> x
4
```

Ligne 1. Dans cet exemple, nous avons déclaré, puis initialisé la variable x avec la valeur 4. Notez bien qu'en réalité, il s'est passé plusieurs choses :

- Python a deviné que la variable était un entier. On dit que Python est un langage au typage dynamique
- Python a alloué (réservé) l'espace en mémoire pour y accueillir un entier. Chaque type de variable prend plus ou moins d'espace en mémoire. Python a aussi fait en sorte qu'on puisse retrouver la variable sous le nom x
- Enfin, Python a assigné la valeur 4 à la variable x

Dans d'autres langages (en C par exemple), il faut coder ces différentes étapes une par une. Python étant un langage dit de haut niveau, la simple instruction $x = 4$ a suffi à réaliser les 3 étapes en une fois !

Lignes 2 et 3. L'interpréteur nous a permis de connaître le contenu de la variable juste en tapant son nom. Retenez ceci car c'est une spécificité de l'interpréteur Python, très pratique pour chasser (debugger) les erreurs dans un programme.

Les types de variables

Le type d'une variable correspond à la nature de celle-ci. Les trois principaux types dont nous aurons besoin dans un premier temps sont les entiers (integer ou int), les nombres décimaux que nous appellerons floats et les chaıˆnes de caractères (string ou str). Bien sûr, il existe de nombreux autres types (par exemple, les booléens, les nombres complexes, etc.). Si vous n'êtes pas effrayés, vous pouvez vous en rendre compte ici. Dans l'exemple précédent, nous avons stocké un nombre entier (int) dans la variable x, mais il est tout à fait possible de stocker des floats, des chaıˆnes de caractères (string ou str) ou de nombreux autres types de variable que nous verrons par la suite :

```
>>> y = 3.14
>>> y
3.14
```

```
>>> a = "bonjour "
>>> a
' bonjour '
>>> b = ' salut '
>>> b
' salut '
>>> c = """girafe"""
>>> c
' girafe '
>>> d = "' lion "'
>>> d
' lion '
```

Nommage

Le nom des variables en Python peut être constitué de lettres minuscules (a à z), de lettres majuscules (A à Z), de nombres (0 à 9) ou du caractère souligné ( ). Vous ne pouvez pas utiliser d'espace dans un nom de variable. Par ailleurs, un nom de variable ne doit pas débuter par un chiffre et il n'est pas recommandé de le faire débuter par le caractère (sauf cas très particuliers). De plus, il faut absolument éviter d'utiliser un mot réservé par Python comme nom de variable (par exemple : print, range, for, from, etc.). Enfin, Python est sensible à la casse, ce qui signifie que les variables TesT, test ou TEST sont différentes.

Écriture scientifique

On peut écrire des nombres très grands ou très petits avec des puissances de 10 en utilisant le symbole e :

```
>>> e+6
1000000.0
>>> 3.12e-3
0.00312
```

On appelle cela écriture ou notation scientifique. On pourra noter deux choses importantes :

— 1e 6 ou $3.12e-3$ n'implique pas l'utilisation du nombre exponentiel e mais signifie $1x10^6$ ou 3.12×10^{-3} respectivement ;

— Même si on ne met que des entiers à gauche et à droite du symbole e, Python génère systématiquement un float.

Enfin, vous avez sans doute constaté qu'il est parfois pénible d'écrire des nombres composés de beaucoup de chiffres, par exemple le nombre d'Avogradro $6.02214076 \times 10^{23}$ ou le nombre d'humains sur Terre (au 26 août 2020) 7807568245. Pour s'y retrouver, Python autorise l'utilisation du caractère souligné (ou underscore) pour séparer des groupes de chiffres. Par exemple :

$>>> avogadro\_number = 6.022\_140\_76e^{23}$

$>>> print(avogadro\_number)$

$6.02214076e^{+23}$

Le caractère est utilisé pour séparer des groupes de 3 chiffres mais on peut faire ce qu'on veut :

$>>> print(7\_80\_7568\_24\_5)$

7807568245

Opérations sur les types numériques

Les quatre opérations arithmétiques de base se font de manière simple sur les types numériques (nombres entiers et floats) :

```
>>> x = 45
>>> x + 2
47
>>> x - 2
43
>>> x * 3
135
>>> y = 2.5
>>> x - y
42.5
>>> (x * 10) + y
452.
```

Remarquez toutefois que si vous mélangez les types entiers et floats, le résultat est renvoyé comme un float (car ce type est plus général). Par ailleurs, l'utilisation de parenthèses permet de gérer les priorités.
L'opérateur / effectue une division. Contrairement aux opérateurs +, - et *, celui-ci renvoie systématiquement un float :

```
>>> 3/4
0.75
>>> 2.5/2
1.25
```

L'opérateur puissance utilise les symboles ** :

```
>>> 2**3
8
```

Opérations sur les chaînes de caractères

Pour les chaînes de caractères, deux opérations sont possibles, l'addition et la multiplication :

```
>>> chaine = "Salut"
>>> chaine
' Salut '
>>> chaine + "Python"
' SalutPython '
>>> chaine * 3
' SalutSalutSalut '
```

L'opérateur d'addition + concatène (assemble) deux chaînes de caractères.
L'opérateur de multiplication * entre un nombre entier et une chaîne de caractères duplique (répète) plusieurs fois une chaînes de caractères.

Attention

Vous observez que les opérateurs + et * se comportent différemment s'il s'agit d'entiers ou de chaînes de caractères : 2 + 2 est une addition alors que "2" + "2" est une concaténation. On appelle ce comportement redéfinition des opérateurs.

Opérations illicites

Attention à ne pas faire d'opération illicite car vous obtiendriez un message d'erreur :

```
>>> "toto" * 1.3
Traceback ( most recent call last ):
File " < stdin >" , line 1 , in < module >
TypeError : can ' t multiply sequence by non - int of type ' float '
>>> "toto" + 2
Traceback ( most recent call last ):
File "< stdin >" , line 1 , in < module >
TypeError : can only concatenate str ( not " int ") to str
```

Notez que Python vous donne des informations dans son message d'erreur. Dans le second exemple, il indique que vous devez utiliser une variable de type str c'est-à-dire une chaıˆne de caractères et pas un int, c'est-à-dire un entier.

La fonction type()

Si vous ne vous souvenez plus du type d'une variable, utilisez la fonction type() qui vous le rappellera.

```
>>> x =2
>>> type(x)
< class 'int' >
>>> y = 2.0
>>> type(y)
< class 'float' >
>>> z = '2'
>>> type(z)
< class 'str' >
```

Nous verrons plus tard ce que signifie le mot class.

Conversion de types

En programmation, on est souvent amené à convertir les types, c'est-à-dire passer d'un type numérique à une chaıˆne de caractères ou vice-versa. En Python, rien de plus simple avec les fonctions int(), float() et str(). Pour vous en convaincre, regardez ces exemples :

```
>>> i = '456'
>>> int(i)
456
>>> float(i)
456.0
```

On verra au chapitre 7 Fichiers que ces conversions sont essentielles. En effet, lorsqu'on lit ou écrit des nombres dans un fichier, ils sont considérés comme du texte, donc des chaıˆnes de caractères. Toute conversion d'une variable d'un type en un autre est appelé casting en anglais, il se peut que vous croisiez ce terme si vous consultez d'autres ressources.

Division de deux nombres entiers

En Python 3, la division de deux nombres entiers renvoie par défaut un float.

```
>>> x = 3/4
>>> x
0.75
≫ type(x)
< class 'float' >
```

La fonction print()

Au début de ce cours, nous avons rencontré la fonction print() qui affiche une chaîne de caractères (le fameux "Hello world!"). En fait, la fonction print() affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne. Ce retour à ligne supplémentaire est ajouté par défaut. Si toutefois, on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument par mot-clé end :

1. >>> print(" Hello world !")
2. Hello world !
3. >>> print(" Hello world !" , end ="")
4. Hello world ! >>>

Ligne 1. On a utilisé l'instruction print() classiquement en passant la chaîne de caractères "Hello world!" en argument. Ligne 3. On a ajouté un second argument end=" ", en précisant le mot-clé end. Ligne 4. L'effet de l'argument end=" " est que les trois chevrons >>> se retrouvent collés après la chaîne de caractères "Hello world!".

Une autre manière de s'en rendre compte est d'utiliser deux fonctions print() à la suite. Dans la portion de code suivante, le caractère ';' sert à séparer plusieurs instructions Python sur une même ligne : >>>
print("Hello"); print("Awa")
Hello
Awa
>>> print("Hello", end = ""); print("Awa")
Hello Awa

La fonction print() peut également afficher le contenu d'une variable quel que soit son type. Par exemple, pour un entier :
>>> var = 3
>>> print(var)
3

Il est également possible d'afficher le contenu de plusieurs variables (quel que soit leur type) en les séparant par des virgules :

```
>>> x = 32
>>> nom = "Elinam"
>>> print(nom, "a", x, "ans")
Elinam a 32 ans
```

Python a écrit une phrase complète en remplaçant les variables x et nom par leur contenu. Vous remarquerez que pour afficher plusieurs éléments de texte sur une seule ligne, nous avons utilisé le séparateur , entre les différents éléments. Python a également ajouté un espace à chaque fois que l'on utilisait le séparateur , . On peut modifier ce comportement en passant à la fonction print() l'argument par mot-clé sep :

```
>>> x = 32
>>> nom = "Elinam"
>>> print(nom, "a", x, "ans", sep = "")
Elinam a 32 ans
>>> print(nom, "a", x, "ans", sep = " - ")
Elinam -a -32 - ans
```

Écriture formatée

L'écriture formatée est un mécanisme permettant d'afficher des variables avec un certain format, par exemple justifiées à gauche ou à droite, ou encore avec un certain nombre de décimales pour les floats. L'écriture formatée est incontournable lorsqu'on veut créer des fichiers organisés en "belles colonnes"

Prise en main des f-strings

f-string est le diminutif de formatted string literals. Les f-strings permettent une meilleure organisation de l'affichage des variables, mise en place depuis python 3.6. Dans la section précédente, nous avons vu les chaînes de caractères ou encore strings qui étaient représentées par un texte entouré de guillemets simples ou doubles. Par exemple : " Ceci est une chaînes de caractères ".
L'équivalent en f-string est tout simplement la même chaînes de caractères précédée du caractère f sans espace entre les deux : f" Ceci est une chaînes de caractères "

Ce caractère f avant les guillemets va indiquer à Python qu'il s'agit d'une f-string permettant de mettre en place le mécanisme de l'écriture formatée, contrairement à une string normale.

```
>>> x = 26
>>> Prenom = "Fo - Komi"
>>> print(f "{Prenom} a {x} ans")
Fo-Komi a 26 ans
```

Remarque

Une variable est utilisable plus d'une fois pour une f-string donnée :

```
>>> var = "to"
>>> print(f "{var } et {var } font {var }{var }")
```

On peut également spécifier le format de leur affichage.

```
>>> x = 0.4780405405405405
>>> print("x est égale à : ", x)
x est égale à : 0.4780405405405405
>>> print(f "x est égale à {x : .2f }")
x est égale à : 0.48
>>> print(f "x est égale à {x : .3f }")
x est égale à : 0.478
```

La méthode .format()

La méthode .format() permet une meilleure organisation de l'affichage des variables (nous expliquerons à la fin de cette section ce que le terme "méthode" signifie en Python). Si on reprend l'exemple précédent :

```
>>> x = 32
>>> nom = "Elinam"
>>> print("{} a {} ans ".format(nom, x))
Elinam a 32 ans
```

— Dans la chaıˆne de caractères, les accolades vides précisent l'endroit où le contenu de la variable doit être inséré.

— Juste après la chaıˆne de caractères, l'instruction .format(nom, x) fournie la liste des variables à insérer, d'abord la variable nom puis la variable x. La méthode .format() agit sur la chaıˆne de caractères à laquelle elle est attachée par le point.

Remarque

Il est possible d'indiquer entre les accolades dans quel ordre afficher les variables, avec 0 pour la variable à afficher en premier, 1 pour la variable à afficher en second, etc. (attention, Python commence à compter à 0). Cela permet de modifier l'ordre dans lequel sont affichées les variables.

```
>>> x = 32
>>> nom = "Elinam"
>>> print("{0} a {1} ans ".format(nom, x))
Elinam a 32 ans
>>> print("{1} a {0} ans ".format(nom, x))
32 a Elinam ans
```

Vous voulez formater un float pour l'afficher avec deux puis trois décimales :

```
>>> x = 0.4780405405405405
>>> print (" La proportion de x est {:.2 f }". format(x))
La proportion de x est 0.48
>>> print (" La proportion de x est {:.3 f }". format (x))
La proportion de x est 0.478
```

Détaillons le contenu des accolades de la première ligne ({:.2f}) :
— Les deux points : indiquent qu'on veut préciser le format.
— La lettre f indique qu'on souhaite afficher la variable sous forme d'un float.
— Les caractères .2 indiquent la précision voulue, soit ici deux chiffres après la virgule.

Notez enfin que le formatage avec .xf (x étant un entier positif) renvoie un résultat arrondi. Il est par ailleurs possible de combiner le formatage (à droite des 2 points) ainsi que l'emplacement des variables à substituer (à gauche des 2 points), par exemple :

```
>>> print(" x (2 déci.) = {0:.2 f } , x (3 déci.) = {0:.3 f }". format ( x ))
x (2 déci.) = 0.48 , x (3 déci.) = 0.478
```

Les listes

Définition Une liste est une structure de données qui contient une série de valeurs. Python autorise la construction de liste contenant des valeurs de types différents (par exemple entier et chaıˆne de caractères), ce qui leur confère une grande flexibilité. Une liste est déclarée par une série de valeurs (n'oubliez pas les guillemets, simples ou doubles, s'il s'agit de chaıˆnes de caractères) séparées par des virgules, et le tout encadré par des crochets. En voici quelques exemples :

>>> animaux = ['girafe', 'tigre', 'singe', 'souris']

>>> animaux

['girafe','tigre','singe','souris']

>>> mixte = ['girafe', 5, 'souris', 0.15]

≫ mixte

['girafe', 5,'souris', 0.15]

```
>>> tailles = [5, 2.5, 1.75, 0.15]
>>> tailles
[5, 2.5, 1.75, 0.15]
```

Lorsque l'on affiche une liste, Python la restitue telle qu'elle a été saisie.
Un des gros avantages d'une liste est que vous pouvez appeler ses éléments par leur position. Ce numéro est appelé indice (ou index) de la liste.
liste : ['girafe', 'tigre', 'singe', 'souris']
indice : [0, 1, 2, 3]

```
>>> animaux[0]
' girafe '
>>> animaux[3]
' souris '
```

Tout comme les chaı^nes de caractères, les listes supportent l'opérateur + de concaténation, ainsi que l'opérateur * pour la duplication :

```
>>> ani1 = ['girafe', 'tigre']
>>> ani2 = ['singe', 'souris']
```

Opération sur les listes

>>> ani1 + ani2
['girafe',' tigre',' singe',' souris']
>>> ani1 * 3
['girafe',' tigre',' girafe',' tigre',' girafe',' tigre']
L'opérateur + est très pratique pour concaténer deux listes. Vous pouvez aussi utiliser la méthode .append() lorsque vous souhaitez ajouter un seul élément à la fin d'une liste. Dans l'exemple suivant nous allons créer une liste vide :
>>> a = []
>>> a
puis lui ajouter deux éléments, l'un après l'autre, d'abord avec la concaténation :
>>> a = a + [15]
>>> a

```
>>> a = a + [-5]
>>> a
[15, -5]
```

puis avec la méthode .append() :

```
>>> a.append(13)
>>> a
[15, -5, 13]
>>> a.append(-3)
>>> a
[15, -5, 13, -3]
```

Dans l'exemple ci-dessus, nous ajoutons des éléments à une liste en utilisant l'opérateur de concaténation + ou la méthode .append(). Nous vous conseillons dans ce cas précis d'utiliser la méthode .append() dont la syntaxe est plus élégante.

Indiçage négatif

La liste peut également être indexée avec des nombres négatifs selon le modèle suivant :
Liste : ['girafe', 'tigre', 'singe', 'souris']
indice positif : 0 1 2 3
Indice négatif : -4 -3 -2 -1

Les indices négatifs reviennent à compter à partir de la fin. Leur principal avantage est que vous pouvez accéder au dernier élément d'une liste à l'aide de l'indice -1 sans pour autant connaıˆtre la longueur de cette liste. L'avant-dernier élément a lui l'indice -2, l'avant-avant dernier l'indice -3, etc.

```
>>> animaux = ['girafe', 'tigre', 'singe', 'souris']
>>> animaux[-1]
' souris '
>>> animaux[-2]
' singe '
```

Pour accéder au premier élément de la liste avec un indice négatif, il faut par contre connaître le bon indice :

```
>>> animaux[-4]
' girafe '
```

Dans ce cas, on utilise plutôt animaux[0].