Introduction à Python II : syntaxe et variables

Ahmed Ammar (`ahmed.ammar@fst.utm.tn`)

Institut Préparatoire aux Études Scientifiques et Techniques, Université de Carthage.

Oct 22, 2019

Table des matières

1 Introduction : "Hello World !"

C'est devenu une tradition que lorsque vous apprenez un nouveau langage de programmation, vous démarrez avec un programme permettant à l'ordinateur d'imprimer le message *"Hello World !"*.

```
In [1]: print("Hello World!")
Hello World!
```

Félicitation ! tout à l'heure vous avez fait votre ordinateur saluer le monde en anglais ! La fonction `print()` est utilisée pour imprimer l'instruction entre les parenthèses. De plus, l'utilisation de guillemets simples `print('Hello World!')` affichera le même résultat. Le délimiteur de début et de fin doit être le même.

```
In [2]: print('Hello World!')
Hello World!
```

2 Commentaires

Au fur et à mesure que vos programmes deviennent plus grands et plus compliqués, ils deviennent plus difficiles à lire et à regarder un morceau de code et à comprendre ce qu'il fait ou pourquoi. Pour cette raison, il est conseillé d'ajouter des notes à vos programmes pour expliquer en langage naturel ce qu'il fait. Ces notes s'appellent des commentaires et commencent par le symbole `#`.

Voyez ce qui se passe lorsque nous ajoutons un commentaire au code précédent :

```
In [3]: print('Hello World!') # Ceci est mon premier commentaire
Hello World!
```

Rien ne change dans la sortie ? Oui, et c'est très normal, l'interprète Python ignore cette ligne et ne renvoie rien. La raison en est que les commentaires sont écrits pour les humains, pour comprendre leurs codes, et non pour les machines.

3 Nombres

L'interpréteur Python agit comme une simple calculatrice : vous pouvez y taper une expression et l'interpréteur restituera la valeur. La syntaxe d'expression est simple : les opérateurs +, -, * et / fonctionnent comme dans la plupart des autres langages (par exemple, Pascal ou C) ; les parenthèses (`()`) peuvent être utilisées pour le regroupement. Par exemple :

```
In [4]: 5+3
Out[4]: 8
In [5]: 2 - 9      # les espaces sont optionnels
Out[5]: -7
In [6]: 7 + 3 * 4  #la hiérarchie des opérations mathématique
Out[6]: 19
In [7]: (7 + 3) * 4  # est-elle respectées?
Out[7]: 40
# en python3 la division retourne toujours un nombre en virgule flottante
In [8]: 20 / 3
Out[8]: 6.666666666666667
In [9]: 7 // 2      # une division entière
Out[9]: 3
```

On peut noter l'existence de l'opérateur `%` (appelé opérateur modulo). Cet opérateur fournit le reste de la division entière d'un nombre par un autre. Par exemple :

```
In [10]: 7 % 2       # donne le reste de la division
Out[10]: 1
In [11]: 6 % 2
Out[11]: 0
```

Les exposants peuvent être calculés à l'aide de doubles astérisques `**`.

```
In [12]: 3**2
Out[12]: 9
```

Les puissances de dix peuvent être calculées comme suit :

```
In [13]: 3 * 2e3   # vaut 3 * 2000
Out[13]: 6000.0
```

4 Affectations (ou assignation)

4.1 variables

Dans presque tous les programmes Python que vous allez écrire, vous aurez des variables. Les variables agissent comme des espaces réservés pour les données. Ils peuvent aider à court terme, ainsi qu'à la logique, les variables pouvant changer, d'où leur nom. C'est beaucoup plus facile en Python car aucune déclaration de variables n'est requise. Les noms de variable (ou tout autre objet Python tel que fonction, classe, module, etc.) commencent par une lettre majuscule ou minuscule (A-Z ou a-z). Ils sont sensibles à la casse (`VAR1` et `var1` sont deux variables distinctes). Depuis Python, vous pouvez utiliser n'importe quel caractère Unicode, il est préférable d'ignorer les caractères ASCII (donc pas de caractères accentués).

Si une variable est nécessaire, pensez à un nom et commencez à l'utiliser comme une variable, comme dans l'exemple ci-dessous :

Pour calculer l'aire d'un rectangle par exemple : `largeur` x `hauteur` :

```
In [15]: largeur = 25
    ...: hauteur = 40
    ...: largeur    # essayer d'accéder à la valeur de la variable largeur
Out[15]: 25
```

on peut également utiliser la fonction `print()` pour afficher la valeur de la variable `largeur`

```
In [16]: print(largeur)
25
```

Le produit de ces deux variables donne l'aire du rectangle :

```
In [17]: largeur * hauteur  # donne l'aire du rectangle
Out[17]: 1000
```

Note
Notez ici que le signe égal (=) dans l'affectation ne doit pas être considéré comme **"est égal à"**. Il doit être **"lu"** ou interprété comme **"est définie par"**, ce qui signifie dans notre exemple :

> La variable `largeur` est définie par la valeur 25 et la variable `hauteur` est définie par la valeur 40.

Avertissement
Si une variable n'est pas *définie* (assignée à une valeur), son utilisation vous donnera une erreur :

```
In [18]: aire     # essayer d'accéder à une variable non définie
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-1b03529c1ce5> in <module>()
----> 1 aire     # essayer d'accéder à une variable non définie

NameError: name 'aire' is not defined
```

Laissez-nous résoudre ce problème informatique (ou **bug** tout simplement) !. En d'autres termes, assignons la variable `aire` à sa valeur.

```
In [19]: aire = largeur * hauteur
    ...: aire  # et voila!
Out[19]: 1000
```

4.2 Noms de variables réservés (keywords)

Certains noms de variables ne sont pas disponibles, ils sont réservés à python lui-même. Les mots-clés suivants (que vous pouvez afficher dans l'interpréteur avec la commande `help("keywords")`) sont réservés et ne peuvent pas être utilisés pour définir vos propres identifiants (variables, noms de fonctions, classes, etc.).

```
In [20]: help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
```

```
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass

# par exemple pour éviter d'écraser le nom réservé lambda
In [22]: lambda_ = 630e-9
    ...: lambda_
Out[22]: 6.3e-07
```

4.3 Les types

Les types utilisés dans Python sont : integers, long integers, floats (double prec.), complexes, strings, booleans. La fonction `type()` donne le type de son argument

Le type int (integer : nombres entiers). Pour affecter (on peut dire aussi assigner) la valeur 20 à la variable nommée `age` :

```
age = 20
```

La fonction `print()` affiche la valeur de la variable :

```
In [24]: print(age)
20
```

La fonction `type()` retourne le type de la variable :

```
type(age)
Out[25]: int
```

```
b = 17.0  # le séparateur décimal est un point (et non une virgule)
b
Out[26]: 17.0
In [27]: type(b)
Out[27]: float
In [28]: c = 14.0/3.0
    ...: c
Out[28]: 4.666666666666667
```

Le type float (nombres en virgule flottante). Notation scientifique :

```
In [29]: a = -1.784892e4
    ...: a
Out[29]: -17848.92
```

Les fonctions mathématiques. Pour utiliser les fonctions mathématiques, il faut commencer par importer le module `math` :

```
import math
```

La fonction `help()` retourne la liste des fonctions et données d'un module.
Soit par exemple : `help('math')`
Pour appeler une fonction d'un module, la syntaxe est la suivante : `module.fonction(arguments)`
Pour accéder à une donnée d'un module : `module.data`

```
 # donnée pi du module math (nombre pi)
In [32]: math.pi
Out[32]: 3.141592653589793
# fonction sin() du module math (sinus)
In [33]: math.sin(math.pi/4.0)
Out[33]: 0.7071067811865475
# fonction sqrt() du module math (racine carrée)
In [34]: math.sqrt(2.0)
Out[34]: 1.4142135623730951
# fonction exp() du module math (exponentielle)
In [35]: math.exp(-3.0)
Out[35]: 0.049787068367863944
# fonction log() du module math (logarithme népérien)
In [36]: math.log(math.e)
Out[36]: 1.0
```

Le type complexe. Python possède par défaut un type pour manipuler les nombres complexes. La partie imaginaire est indiquée grâce à la lettre « `j` » ou « `J` ». La lettre mathématique utilisée habituellement, le « `i` », n'est pas utilisée en Python car la variable i est souvent utilisée dans les boucles.

```
In [37]: a = 2 + 3j
    ...: type(a)
Out[37]: complex
In [38]: a
Out[38]: (2+3j)
```

Avertissement

```
In [39]: b = 1 + j
---------------------------------------------------------------
NameError                          Traceback (most recent call last)
<ipython-input-39-0f22d953f29e> in <module>()
----> 1 b = 1 + j

NameError: name 'j' is not defined
```

Dans ce cas, on doit écrire la variable `b` comme suit :

```
In [41]: b = 1 + 1j
    ...: b
Out[41]: (1+1j)
```

sinon Python va considérer `j` comme variable non définie.

On peut faire l'addition des variables complexes :

```
In [42]: a + b
Out[42]: (3+4j)
```

```
In [43]: nom = 'Tounsi' # entre apostrophes
    ...: nom
Out[43]: 'Tounsi'
In [44]: type(nom)
Out[44]: str
In [45]: prenom = "Ali"  # on peut aussi utiliser les guillemets
    ...: prenom
Out[45]: 'Ali'
In [46]: print(nom, prenom)  # ne pas oublier la virgule
Tounsi Ali
```

Le type str (string : chaîne de caractères).

La concaténation désigne la mise bout à bout de plusieurs chaînes de caractères. La concaténation utilise l'opérateur `+` :

```
In [47]: chaine = nom + prenom  # concaténation de deux chaînes de caractères
    ...: chaine
Out[47]: 'TounsiAli'
```

Vous voyez dans cet exemple que le nom et le prénom sont collé. Pour ajouter une espace entre ces deux chaînes de caractères :

```
In [48]: chaine = prenom + ' ' + nom
    ...: chaine # et voila
Out[48]: 'Ali Tounsi'
```

On peut modifier/ajouter une nouvelle chaîne à notre variable `chaine` par :

```
In [49]: chaine = chaine + ' 22 ans'  # en plus court : chaine += ' 22 ans'
    ...: chaine
Out[49]: 'Ali Tounsi 22 ans'
```

La fonction `len()` renvoie la longueur (*length*) de la chaîne de caractères :

```
In [53]: print(nom)
    ...: len(nom)
Tounsi
Out[53]: 6
```

```
 +---+---+---+---+---+---+
|-----------------------|
 | T | o | u | n | s | i |
 +---+---+---+---+---+---+
 |-----------------------|
 0   1   2   3   4   5   6
 --->
-6  -5  -4  -3  -2  -1
                  <----
```

Indexage et slicing :

```
In [55]: nom[0]  # premier caractère (indice 0)
Out[55]: 'T'

In [56]: nom[:] # toute la chaine
Out[56]: 'Tounsi'

In [57]: nom[1] # deuxième caractère (indice 1)
Out[57]: 'o'

In [58]: nom[1:4]   # slicing
Out[58]: 'oun'

In [59]: nom[2:]  # slicing
Out[59]: 'unsi'

In [60]: nom[-1]   # dernier caractère (indice -1)
Out[60]: 'i'

In [61]: nom[-3:]    # slicing
Out[61]: 'nsi'
```

Avertissement

On ne peut pas mélanger le type `str` et type `int`.
Soit par exemple :

```
In [63]: chaine = '22'
    ...: annee_naissance = 2018 - chaine
---------------------------------------------------------
TypeError                 Traceback (most recent call last)
<ipython-input-63-8607078f78d2> in <module>()
      1 chaine = '22'
----> 2 annee_naissance = 2018 - chaine

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Pour corriger cette erreur, la fonction `int()` permet de convertir un type `str` en type `int` :

```
In [64]: nombre = int(chaine)
    ...: type(nombre) # et voila!
Out[64]: int
```

Maintenant on peut trouver `annee_naissance` sans aucun problème :

```
In [65]: annee_naissance = 2018 - nombre
    ...: annee_naissance
Out[65]: 1996
```

Interaction avec l'utilisateur (la fonction `input()`) La fonction `input()` lance une case pour saisir une chaîne de caractères.

```
In [66]: prenom = input('Entrez votre prénom : ')
    ...: age = input('Entrez votre age : ')

Entrez votre prénom : Foulen

Entrez votre age : 25
```

Formatage des chaînes Un problème qui se retrouve souvent, c'est le besoin d'afficher un message qui contient des valeurs de variables.

Soit le message : Bonjour Mr/Mme `prenom`, votre age est `age`.

La solution est d'utiliser la méthode `format()` de l'objet chaîne `str()` et le `{}` pour définir la valeur à afficher.

```
print(" Bonjour Mr/Mme {}, votre age est {}.".format(prenom, age))
```

Le type list (liste) Une liste est une structure de données.

Le premier élément d'une liste possède l'indice (l'index) 0.

Dans une liste, on peut avoir des éléments de plusieurs types.

```
In [1]: info = ['Tunisie', 'Afrique', 3000, 36.8, 10.08]

In [2]: type(info)
Out[2]: list
```

La liste info contient 5 éléments de types str, str, int, float et float

```
In [3]: info
Out[3]: ['Tunisie', 'Afrique', 3000, 36.8, 10.08]

In [4]: print('Pays : ', info[0])    # premier élément (indice 0)
Pays :  Tunisie

In [5]: print('Age : ', info[2])     # le troisième élément a l'indice 2
Age :  3000

In [6]: print('Latitude : ', info[3]) # le quatrième élément a l'indice 3
Latitude :  36.8
```

La fonction `range()` crée une liste d'entiers régulièrement espacés :

```
In [7]: maliste = range(10) # équivalent à range(0,10,1)
   ...: type(maliste)
Out[7]: range
```

Pour convertir une range en une liste, on applique la fonction `list()` à notre variable :

```
In [8]: list(maliste)   # pour convertir range en une liste
Out[8]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

On peut spécifier le début, la fin et l'intervalle d'une range :

```
In [9]: maliste = range(1,10,2)   # range(début,fin non comprise,intervalle)
   ...: list(maliste)
Out[9]: [1, 3, 5, 7, 9]

In [10]: maliste[2] # le troisième élément a l'indice 2
Out[10]: 5
```

On peut créer une liste de listes, qui s'apparente à un tableau à 2 dimensions (ligne, colonne) :

```
0   1   2
10  11  12
20  21  22
```

```
In [11]: maliste = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
    ...: maliste[0]
Out[11]: [0, 1, 2]

In [12]: maliste[0][0]
Out[12]: 0

In [13]: maliste[2][1] # élément à la troisième ligne et deuxième colonne
Out[13]: 21

In [14]: maliste[2][1] = 78   # nouvelle affectation

In [15]: maliste
Out[15]: [[0, 1, 2], [10, 11, 12], [20, 78, 22]]
```

Le type bool (booléen). Deux valeurs sont possibles : `True` et `False`

```
In [16]: choix = True # NOTE: "True" différent de "true"
    ...: type(choix)
Out[16]: bool
```

Les opérateurs de comparaison :

Opérateur	Signification	Remarques
`<`	strictement inférieur	
`<=`	inférieur ou égal	
`>`	strictement supérieur	
`>=`	supérieur ou égal	
`==`	égal	Attention : deux signes `==`
`!=`	différent	

```
In [17]: b = 10
    ...: b > 8
Out[17]: True

In [18]: b == 5
Out[18]: False

In [19]: b != 5
Out[19]: True

In [20]: 0 <= b <= 20
Out[20]: True
```

Les opérateurs logiques : `and`, `or`, `not`

```
In [21]: note = 13.0

In [22]: mention_ab = note >= 12.0 and note < 14.0

In [23]: # ou bien : mention_ab = 12.0 <= note < 14.0

In [24]: mention_ab
Out[24]: True
```

```
In [25]: not mention_ab
Out[25]: False

In [26]: note == 20.0 or note == 0.0
Out[26]: False
```

L'opérateur `in` s'utilise avec des chaînes (type `str`) ou des listes (type `list`).

Pour une chaînes :

```
In [30]: chaine = 'Bonsoir'
    ...: #la sous-chaîne 'soir' fait-elle partie de la chaîne 'Bonsoir' ?

In [31]: resultat = 'soir' in chaine
    ...: resultat
Out[31]: True
```

Pour une liste :

```
In [32]: maliste = [4, 8, 15]
    ...: #le nombre entier 9 est-il dans la liste ?

In [33]: 9 in maliste
Out[33]: False

In [34]: 8 in maliste
Out[34]: True

In [35]: 14 not in maliste
Out[35]: True
```

5 Lectures complémentaires

— Documentation Python 3.6 : https://docs.python.org/fr/3.6/tutorial/index.html
— Apprendre à programmer avec Python, par Gérard Swinnen : http://inforef.be/swi/python.htm
— Think Python, par Allen B. Downey : https://greenteapress.com/wp/think-python/