TD n°03 : Initiation aux éléments de programmation en Python

2 heures

Rédigé par Pascal Delahaye

14 septembre 2015

Sauf dans les exercices de la fin, les manipulations suivantes seront effectuées dans la console.

1 Opérateurs et fonctions mathématiques usuels

Pour vous familiariser avec l'utilisation de la console et les instructions de base en Python, taper les instructions suivantes. En utilisant la flèche ↑ on retrouve les instructions tapées précédemment : cela nous évite de devoir les recopier.

Opérations usuelles :

```
10%4      # Cela nous donne le reste de la division euclidienne de 10 par 4
10/4      # Cela nous donne le résultat de la division de 10 par 4
10//4     # Cela nous donne le quotient de la division euclidienne de 10 par 4

1+2       # Calcul de la somme de 1 et 2
5*2       # Calcul du produit de 5 et 2
1-7       # Calcul de la différence 1 moins 7

2**3      # Calcul de 2 à la puissance 3 (sous forme d'entier)
2.**3     # Calcul de 2 à la puissance 3 (sous forme de nombre flottant)
```

Remarque 1. La différence entre 1 et 1. est fondammentale !
Cette différence de syntaxe indique au langage le type des nombres (flottant ou entier) qui sont manipulés et donc la façon dont ceux-ci seront codés dans l'ordinateur : un nombre flottant mobilise beaucoup plus de mémoire qu'un nombre entier.

1. la notation "1" indique qu'il s'agit d'un nombre entier
2. la notation "1." indique qu'il s'agit d'un nombre flottant.

Fonctions mathématiques usuelles :

```
>>> sqrt(2)    # Calcul de la racine de 2
>>> sin(2)     # Calcul du sinus de 2
>>> log(2)     # Calcul du logarithme népérien de 2 : ln(2)
>>> sinh(2)    # Calcul du sinus hyperbolique de 2
>>> arccos(0.5)# Calcul de l'arc-cosinus de 2
>>> 8**(1/3)   # Calcul de la racine cubique de 8

>>> floor(3.5) # Renvoie la partie entière de 3.5
>>> random()   # Renvoie un flottant aléatoire entre 0 et 1
```

Remarque 2. Les fonctions mathématiques usuelles (racine carrée, logarithme, sinus, tangente hyperbolique, arc-cosinus...) sont reconnues par Python. Oufff... Attention cependant à la fonction ln qui s'écrit `log()` en Python.

Exercice : 1

(∗) Sessa demanda au roi d'Egypte de lui donner la quantité de blé obtenue de la façon suivante : sur un échiquier, on place 1 grain sur la première case et plus généralement 2^i grains sur la $i+1$ ième case.
Sachant qu'un grain de blé pèse 0,035g et que la production mondiale de blé est de 600 millions de tonnes par an, combien faudrait-il d'années de production pour satisfaire le désir de Sessa ? *Réponse : autour du millier d'années...*

2 Quelques structures de données usuelles

2.1 Entiers et flottants

Nous ne reviendrons pas sur la structure d'*entier* et la structure de nombre *flottant*, vues en détail dans le cours sur le codage des nombres. Cependant, on retiendra les deux fonctions suivantes permettant la conversion "entier ↔ flottant".

```
*** Conversion Entier <-> Flottant ***

>>> int(3.7)    # Convertit le flottant 3.7 en l'entier 3
>>> float(5)    # Convertit l'entier 5 en le flottant 5.0
```

2.2 Les Chaînes de caractères

Une *chaîne de caractères* est une succession de caractères qui forme une phrase au sens large.
En Python, les chaînes de caractères se définissent soit avec des guillemets (`>>> "toto"`), soit avec des apostrophes (`>>> 'toto'`).

Taper les instructions suivantes dans la console :

```
*** Les Chaînes de caractères ***

>>> "e"
>>> "toto"
>>> " va à la plage"
>>> "toto" + " va à la plage"      # Concaténation de deux chaînes de caractères
>>> "g#\‘pd [[[#{~~~~"
```

2.3 Les Listes

En Python, une *liste* est une succession ordonnée de valeurs de natures éventuellement diverses.
Elles se définissent à l'aide de crochets (`>>> [1,"toto",3.14]`).

Taper les instructions suivantes dans la console :

```
*** Les listes ***

[0,[2],"toto",-1,3,5]         # Construit une liste
[0,"toto"] + ["titi",3.14]    # Concatène deux listes
len([1,5,23,9,"tutu"])        # Renvoie la longueur de la liste
```

Remarque 3.
Nous verrons plus tard qu'une liste peut-être convertie en une structure de *tableau* avec la fonction `array()`.
La différence principale entre les deux structures étant que les tableaux ne peuvent contenir des éléments de types différents, ce qui présente l'avantage de diminuer fortement le temps d'accès aux différents éléments de la structure.
Taper `>>> array([1,3.,"toto"])` dans la console et observez le résultat.

2.4 Les n-uplets

Comme les listes, les *n-uplets* sont des successions ordonnées de valeurs de natures éventuellement différentes.
En Python, les n-uplets se définissent soit avec des parenthèses en séparant les éléments avec des virgules (`>>> ("toto",2,3.14)`), soit simplement en séparant les éléments avec des virgules (`>>> "toto",2,3.14`).

Taper les instructions suivantes dans la console :

```
*** Les n-uplets ***

>>> (1,2)
>>> 1,2
>>> (1,"toto",3.14)
>>> 1,"toto",3.14
>>> (1,2) + (3,"tutu")    # Il est possible de concaténer des n-uplets
```

Remarque 4. La différence avec les listes est la façon dont l'une et l'autre des structures sont codées dans la mémoire de l'ordinateur et les opérations que l'on a la possibilité d'effectuer.

2.5 Les propositions

Les *propositions*, aussi appelées *booléens* sont des affirmations ayant la propriété d'être soit VRAIE soit FAUSSE.

Taper les instructions suivantes dans la console :

Exemples de propositions :

```
4==5             # Ceci est la proposition ''4 égal à 5''
4!=5             # Ceci est la proposition ''4 est différent de 5''
5<0              # Ceci est la proposition ''5 est strictement plus petit que 0''
a>=0             # Ceci est la proposition ''a est supérieur ou égal à 0''

((2 > 3) or (1 !=2)) and (not (-1 <= 3))    # Ceci est une proposition plus compliquée...
```

Remarque 5.

1. Lorsqu'on entre une proposition en instruction, la console renvoie la valeur de vérité (`true` ou `false`) correspondante. Cela signifie que Python remplace automatiquement les booléens par leur valeur de vérité.
2. Les opérateurs logique `or`, `and` et `not` permettent de construire des propositions plus complexes.

3 Les variables

En programmation, il est indispensable de pouvoir nommer les données avec lesquelles on travaille. Pour cela, on fait appel à la notion de *variable*. Chaque variable porte un *nom* et une *valeur* et se définit par une instruction de la forme :

```
>>> nom = valeur
```

On dit alors que la variable `nom` est *affectée* de la valeur `valeur`.
Cette instruction effectue deux choses en même temps : elle définit la variable `nom` tout en lui affectant une valeur.

Taper les instructions suivantes dans la console :

```
*** Les variables de type INT ***

a = 2            # On créer la variable "a" dans laquelle on stocke la valeur 2

a = eval(input("Choisissez une valeur entière : "))
                 # La valeur entrée par l'utilisateur est stockée dans une variable
                 # de type int ou float que l'on appelle "a"
print(a)         # Affiche la valeur contenue dans la variable a
a = a + 1        # Incrémente la variable a de 1

b,c = 1,2        # On définit deux variables "a" et "b" prenant les valeurs 1 et 2
b,c              # Utilisation des valeurs contenues dans b et c
print(b,c)       # Affichage à l'écran des valeurs contenues dans b et c
```

Remarque 6.

1. Attention à l'ordre des termes de part et d'autre du signe "=".
   Le (ou les) terme de gauche correspond au *nom* de la variable que l'on définit tandis que le terme de droite correspond à la *valeur* que l'on stocke dans la variable. Ici, on dit que :

   "La variable b est affectée de la valeur 1" et que *"la variable L1 est affectée de la valeur* $[2,3,5,7,11]$*"*.

2. Dans le cas de l'exemple avec les variables "b" et "c", on constate que la structure de n-uplet permet de définir et d'affecter plusieurs variables en même temps.

```
*** Les variables de type CHAINE de CARACTERES***

debut = "toto "
debut[0], debut[1], debut[2] # Renvoie les 3 premières lettres de la chaîne de caractères
debut[4]                     # Erreur !!
debut*3                      # Concatène 3 fois la chaîne de caractères d

fin = "va à la plage"
phrase = debut + fin # Concaténation des deux chaînes de caractères
                     # Et stockage dans une nouvelle variable "phrase"
```

Remarque 7. Définir des variables facilite la manipulation des données.
Dans la mesure du possible, on donnera aux variables des noms explicites.

```
*** Les variables de type LISTE ***

L1 = [2,3,5,7,11]     # Création d'une liste de valeurs
L1[0], L[2], L[4]     # Renvoie certaines composantes de la liste
L1[1]="toto"          # Modifie la deuxième valeur de la liste
L1.pop(0)             # Renvoie la première valeur de L et l'élimine de L
L1                    # Vérification de l'opération précédente
L1.append(3)          # Ajoute la valeur 3 en queue de liste
L1                    # Vérification de l'opération précédente

L2 = ["toto", 5]
L = L1 + L2           # Concatène les deux listes et stocke le résultat dans une
                      # nouvelle liste L
```

Remarque 8. ⚠ Les noms des variables doivent respecter certaines règles !

1. Il faut donner un nom explicite afin de connaître d'un seul coup d'oeil la nature de la donnée stockée.
2. On ne peut pas donner n'importe quel nom à une variable.
   (a) Un nom commence toujours par une lettre
   (b) Un nom peut comporter des lettres et des chiffres
   (c) Un nom ne comporte pas d'espace mais peut contenir des underscores "_"
   (d) Python distingue les majuscules des minuscules

4 Trois fonctions Python de base

• **La fonction** `print()`

Taper les instructions suivantes dans la console :

```
*** Affichage à l'écran ***

phrase = "ça va ?"  # Création de la variable 'phrase'
print(phrase)       # Affiche la phrase ''Ca va ?'' à l'écran
phrase              # Affiche la phrase ''Ca va ?'' à l'écran mais...

L = [2,5,-3]        # Création d'une liste L
print(L)            # Affiche le contenu de la liste L à l'écran
L                   # Affiche le contenu de la liste L à l'écran mais...

r = 2               # création de la variable r
print("Le rayon du cercle est : ", r)
```

Il semblerait, d'après les manipulations effectuées, que la fonction `print()` soit inutile.
C'est vrai lorsqu'on travaille dans la console, mais elle est en revanche indispensable lorsqu'on travaille dans l'éditeur.

• **Les fonctions** `input()` **et** `eval()`

Taper les instructions suivantes dans la console :

```
*** Interaction avec l'utilisateur ***

nom = input("Quel est ton nom ? ") # Demande à l'utilisateur d'entrer une donnée
                                   # Entrez alors votre nom !!
print(nom)                         # Affiche le contenu de la variable nom

a = input("Quel est ton âge ? ")   # Validez et entrez votre âge
print(a)                           # Attention : la donnée est de type "chaine de caractères"

a = eval(input("Quel est ton âge ? "))
print(a)                           # la donnée est ici de type numérique (int ou float)
```

Remarque 9. ⚠ L'utilisation de la fonction `eval()` est indispensable pour convertir en "entier" ou en "flottant" les données entrées grâce à la fonction `input()` par l'utilisateur.

5 Les fonctions lambda

Il s'agit de fonctions simples se définissant à l'aide d'une unique instruction.
Taper les instructions suivantes dans la console :

Les fonctions lambda :

```
>>> f = lambda x: x**2 + 1    # On définit la fonction x -> x^2+1
>>> f(2)

>>> g = lambda x,y: x*y       # On définit la fonction (x,y) -> x*y
>>> g(2,3)

>>> h = lambda x: (sin(x))**2
>>> h(2)                      # On définit la fonction x -> (sin(x))^2

>>> k = lambda x: (x, 2*x)    # Une fonction lambda peut renvoyer un couple
>>> k(2)
>>> a,b = k(2)                # On stocke le résultat de k(2) dans les variables a et b
>>> a,b                       # Vérification

>>> equiv = lambda a,b :
            (a and b) or (not(a) and not(b))  # Fonction qui teste une équivalence
>>> equiv (True,True)                         # Utilisation de cette fonction
```

Remarque 10. Les fonctions lambda sont des fonctions simples a un ou plusieurs arguments qui ne contiennent qu'une seule instruction. Elles peuvent renvoyer n'importe quel type de données.

Exercice : 2

1. Programmer une fonction qui à un cercle de rayon R associe son périmètre et l'aire du disque correspondant.

2. Programmer une fonction qui a un couple d'entiers associe un couple donnant leur somme et leur produit. Vous ferez deux programmes : l'un dans la console sous la forme d'une fonction *lambda* et l'autre dans l'éditeur qui demande ces deux entiers à l'utilisateur et n'utilise pas de fonction *lambda*.

3. Programmer une fonction qui à une liste L et deux entiers i et j renvoie les ième et jème éléments de cette liste.

4. Programmer une fonction qui teste si les deux premiers éléments d'une liste sont égaux modulo 5.

5. Programmer les fonctions suivantes :

   (a) $f(x) = \dfrac{1}{x^2-1}$

   (b) $g(x,y) =$ le reste fois le quotient de la division euclidienne de x par y

   (c) $h(x) = (x-1)^2 + \dfrac{x}{x^2+2}$

   (d) $i(x,y) = x^2 - x * y + y^2$

6. Programmer une fonction qui teste si une année donnée est bissextile.
   On rappelle que les années bissextiles sont les années divisibles par 4 à l'exception des débuts de siècle sauf si l'année est divisible par 400.