TP1_BCPST_852

September 20, 2017

1 TP 1 Premiers pas en Python

1.1 Installation de l'Ide Pyzo et de la distribution Python Anaconda

- Le plus simple est d'installer l'environnement de développement (ou Integrated Development Environment) Pyzo puis la distribution Python Anaconda en suivant le guide rapide d'installation sur le site de Pyzo. Installez bien Anaconda et non pas Miniconda (sauf sur votre clef USB éventuellement) car Anaconda inclut tous les modules nécessaires et le Jupyter Notebook si pratique que vous êtes en train d'utiliser. Par la suite nous travaillerons plutôt avec l'IDE Pyzo.
- `pygame` pose problème avec conda et pip. Dans ce cas on se rend sur le site http://www.lfd.uci.edu/~gohlke/pythonlibs/ et on télécharge le paquet wheel de pygame qu'on installe avec pip install paquet.whl
- En cas de problème contactez le professeur par la messagerie de l'ENT.

1.2 S'exercer à la programmation en Python

- Le site de référence est `http://www.france-ioi.org`, si vous souhaitez progresser, il faut s'entraîner et ce site est la plateforme idéale : créez un compte puis suivez le parcours lycée pour commencer.
- Pour visualiser ce que fait la machine lors de l'exécution d'un code Python, et mieux comprendre le fonctionnement du langage, un bon outil est `http://pythontutor.com/`

1.3 Exercice 3 Calculs en Python

```
In [1]: 2 + 5

Out[1]: 7

In [2]: 2*5

Out[2]: 10

In [3]: 2**5 #exponentiation

Out[3]: 32
```

```
In [2]: 2.2/3.5

Out[2]: 0.6285714285714287

In [6]: 17.0/5.0

Out[6]: 3.4

In [15]: 17//5

Out[15]: 3

In [18]: %%python
         print(17/5)   #en python2

3


In [21]: 17/5  #en python3

Out[21]: 3.4

In [22]: 3//-2

Out[22]: -2

In [23]: -(3//2)

Out[23]: -1

In [24]: (-3)%2

Out[24]: 1

In [25]: 3%(-2)

Out[25]: -1

In [26]: -(3%2)

Out[26]: -1
```

1.4 Les instructions d'entrée / sortie

1.4.1 La fonction `print`

- On utilise la fonction `print` pour un affichage sur la sortie standard qui par défaut est l'écran.

```
In [3]: print(Bonjour)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-3-f27dac4a7a30> in <module>()
----> 1 print(Bonjour)


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

Pour afficher du texte, celui-ci doit être contenu dans une chaîne de caratère délimitée par des apostrophes comme 'La mer' ou des guillemets comme "L'océan".

```
In [4]: print('Bonjour')
```

```
Bonjour
```

On peut aussi afficher la valeur d'une *expression* obtenue par évaluation d'une combinaison de *littéraux* (valeurs correspondant aux types de base du langage), d'opérateurs et de variables

```
In [5]: n = 2000
        print(2017 - n)
```

```
17
```

On peut passer plusieurs arguments à la fonction `print` en les séparant par une virgule, `print` les affichera par concaténation en utilisant un espace vide par défaut comme séparateur.

```
In [6]: print('Vous avez ', 2017 - n, ' ans')
```

```
Vous avez  17  ans
```

Par défaut, `print` effectue un saut de ligne après chaque affichage mais on peut changer ce comportement par défaut en réglant le paramètre optionnel `end=''`.

```
In [7]: print('Ligne 1')
        print('Ligne 2')
```

```
Ligne 1
Ligne 2
```

```
In [8]: print('Colonne 1', end='|')
        print('Colonne 2')
```

```
Colonne 1|Colonne 2
```

On peut aussi changer le paramètre de séparation par défaut `sep=''`.

```
In [9]: print('Colonne 1', 'Colonne 2', sep='|')
```

```
Colonne 1|Colonne 2
```

Pour afficher l'aide sur une fonction en `python`, on peut utiliser la documentation en ligne sur https://docs.python.org/3/ ou la documentation intégrée avec la fonction `help`.

```
In [11]: help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

1.5 Exercice 5

```
In [5]: 2 * 5
        4 * 5
        print(3 * 5)
```

```
15
```

1.5.1 La fonction `input`

- La fonction `input` récupère un texte un saisi par l'utilisateur après un `prompt` et retourne la saisie sous forme de chaîne de caractères. La fonction prend comme argument optionnel le `prompt` qui doit être une chaîne de caractères.

1.6 Exercice 5 compléments

- Testez le programme ci-dessous.
- Comment expliquez-vous la réponse de l'interpréteur ?
- Comment peut-on corriger ce programme ?

```
In [ ]: date = input('Entrez votre date de naissance : ')
        print('Vous avez  ', 2017 - date, 'ans')
```

1.7 Importation de modules

On peut importer le module/bibbliothèque `math` en créant juste un point d'accès (on donne les clefs de la bibliothèque). C'est malin, mais contraignant car il faut faire précéder chaque nom de fonction de bibliothèque du nom de la bbibliothèque.

```
In [7]: import math

In [8]: math.sqrt(4)

Out[8]: 2.0
```

Pour alléger la syntaxe, on peut emprunter une fonction de la bibliothèque et l'importer dans l'espace de nom du programme (quitte à le "polluer" et à créer des conflits de nommage avec des fonctions ou variables existantes ou importées d'autres bibliothèques)

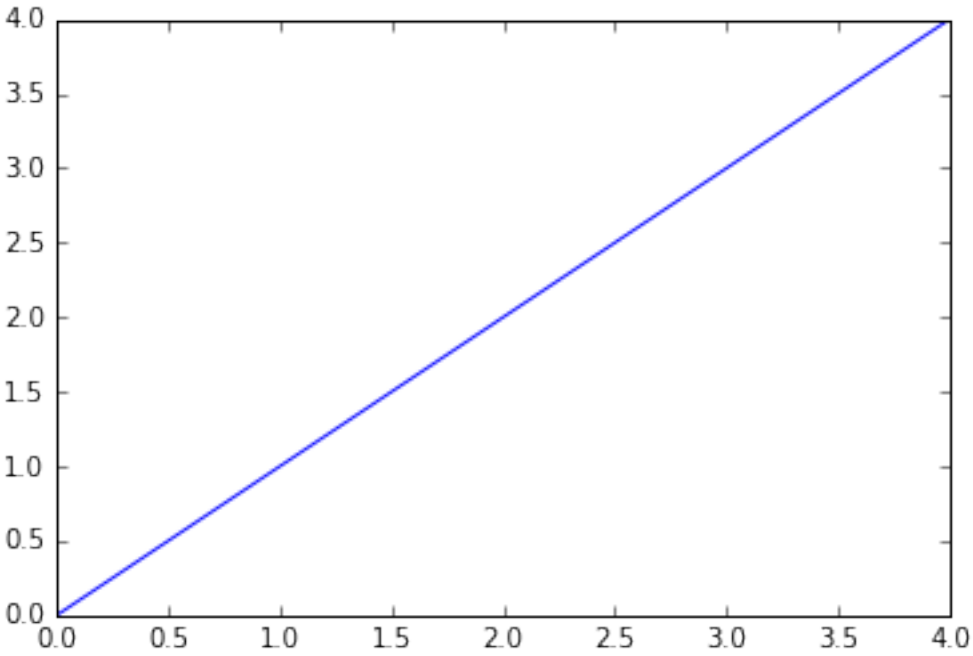
```
In [9]: from math import sqrt

In [10]: sqrt(4)

Out[10]: 2.0

In [12]: log(1)


         ---------------------------------------------------------------------------

         NameError                                 Traceback (most recent call last)

         <ipython-input-12-cfa4946d0225> in <module>()
     ----> 1 log(1)


         NameError: name 'log' is not defined


In [13]: math.log(1)

Out[13]: 0.0
```

Parfois on importe toutes les fonctions de la bibliothèque (on la déménage à l'intérieur de son programme quitte à l'encombrer inutilement avec des fonctions qui ne nous serviront à rien)

```
In [14]: from math import *

In [16]: log(1)

Out[16]: 0.0
```

Une solution intermédiaire est d'utiliser un alias pour la la bibliothèque.

```
In [17]: import matplotlib.pyplot as plt

In [21]: % matplotlib inline
         plt.plot([0,4],[0,4])

Out[21]: [<matplotlib.lines.Line2D at 0xb007258c>]
```



1.8 La boucle `for` ou boucle inconditionnelle

Une boucle inconditionnelle permet de répéter un bloc d'instructions lorsqu'on connaît à l'avance le nombre de répétitions.

La situation la plus courante est :

```
Pour k allant de 1 à 10 répéter
    Bloc d'instructions
FinPour
```

Sa traduction en Python est :

```
# flux parent
for k in range(1, 11):
    # bloc d'instructions
# retour au flux parent
```

On remarque l'utilisation `range(1, 11)` alors qu'on attendrait `range(1, 10)`.

En fait `range(1, 11)` retourne un *itérateur*, qui va parcourir avec un incrément de 1 tous les entiers n tels que $1 \leqslant n < 11$.

Il faut bien se souvenir que la borne supérieure de `range(n, m)` est exclue mais il est facile de retenir que le nombre de tours de boucles est $m - n$.

Par exemple, pour calculer la somme des tous les entiers consécutifs de 100 à 200, on peut écrire :

```
somme = 0
for k in range(100, 201):
    somme = somme + k
print("La somme est ", somme)
```

S'il s'agit juste de répéter `n` fois un bloc d'instructions on utilise le raccourci `range(n)` au lieu de `range(0, n)` ou de `range(1, n + 1)`.

Par exemple pour dire 100 fois "Merci", on peut écrire :

```
for k in range(100):
    print("Merci !")
print("Ouf!")
```

La fonction range offre un troisième paramètre optionnel qui permet de changer l'incrément par défaut qui est 1.

Par exemple, pour pour calculer la somme des tous les entiers pairs consécutifs entre 100 et 200, on peut écrire :

```
sommePair = 0
for k in range(100, 201, 2):
    sommePair = sommePair + k
print("La somme est ", sommePair)
```

Enfin, la boucle `for` permet aussi de parcourir des objets itérables comme les chaînes de caractères, les listes, les fichiers etc ...

Par exemple pour afficher les caractères consécutifs de la chaîne "Bonjour" avec un caractère par ligne, on peut écrire :

```
chaine = "Bonjour"
for c in chaine:
    print(c)
```

Autre exemple, pour afficher des messages d'invitation personnalisés :

```
for nom in ["Jean-Luc", "Emmanuel", "François", "Marine"]:
    print("Salut", nom, "je t'invite à mon anniversaire samedi !")
```

1.9 Exercice 6

```
In [27]: for i in range(2,5):
             print(i**2)

4
9
16
```

1.10 L'affectation et le type d'une variable

Une *variable* est l'association d'un espace de la mémoire de l'ordinateur, accessible par son nom, et d'une valeur que l'on y stocke. En Python on définit une variable par une instruction d'affectation, par exemple pour affecter la valeur 12 à la variable de nom `a` on écrit :

```
a = 12
```

Un langage de programmation fixe des contraintes pour les nomes de variables. En Python les seuls caractères autorisés sont les caractères non accentués, les chiffres (sauf en première position) et le tiret bas (mais pas le tiret haut).

```
1var = 13      #nom incorrect
var1 = 13      #nom correct
var-1 = 14      #nom incorrect
var_1 = 14      #nom correct
```

Un langage de programmation n'accepte par défaut qu'un nombre fini de types de valeurs, ainsi une variable est caractérisée par son **type** accessible par la fonction `type` et qu'on peut afficher avec la fonction `print`:

```
type(a)
```

1.11 Exercice 7 Variables et affectations

```
In [24]: a = 3

In [25]: b = 4

In [23]: a + 1

Out[23]: 4

In [26]: (2**a + 12)/b

Out[26]: 5.0

In [27]: a = 851

In [28]: a = a + 1

In [29]: a

Out[29]: 852

In [30]: 852 = a


          File "<ipython-input-30-eecc9f2b5236>", line 1
        852 = a
              ^
    SyntaxError: can't assign to literal
```

1.12 Exercice 8

```
In [31]: from math import pi, floor

In [32]: a, b, c = 5, 3**4, 2*pi

In [33]: a + b

Out[33]: 86

In [34]: a*c/b

Out[34]: 0.3878509448876288

In [35]: (a - b)/c

Out[35]: -12.095775674984045

In [36]: a = a + 2

In [37]: a = 5
         b = 7
         a = b
         a + 3

Out[37]: 10

In [38]: a, b

Out[38]: (7, 7)

In [39]: d = c/4

In [40]: d

Out[40]: 1.5707963267948966

In [41]: int(d)

Out[41]: 1

In [42]: round(d)

Out[42]: 2

In [43]: floor(d)

Out[43]: 1

In [44]: int(-d)

Out[44]: -1

In [45]: round(-d)

Out[45]: -2

In [46]: floor(-d)

Out[46]: -2
```

On peut remarquer que `int` tronque la partie décimale et convertit le réel approché en entier et que `floor` retourne la partie entière du réel approché. Quant à `round` elle arrondit à l'entier le plus proche.

1.13 Exercice 9

```
In [54]: x = 10
         y = 15
         z = x + y
         x = y
         y = z
         print(x + y + z)
```

```
65
```

1.14 Exercice 10

```
In [ ]: a = 100
        b = 17
        c = a - b
        a = 2
        c = a + b
```

```
In [64]: (a, b, c)
```

```
Out[64]: (3, 4, 6.283185307179586)
```

```
In [65]: a = 3
         b = 4
         c = a
         a = b
         b = c
```

```
In [66]: (a, b, c)
```

```
Out[66]: (4, 3, 3)
```

1.15 Exercice 11

```
In [58]: a, b = 3, 4
```

```
In [59]: a + 1
```

```
Out[59]: 4
```

```
In [60]: (2**a + 12)/b
```

```
Out[60]: 5.0
```

```
In [61]: (a == 2) or (a != 2 and b > 6)
```

```
Out[61]: False
```

```
In [62]: "j'ai " + str(b**a) + " ans"
```

```
Out[62]: "j'ai 64 ans"
```

1.15.1 Opérateurs

Une condition de test doit avoir une valeur booléenne, ce peut être le résultat :

- d'une *comparaison arithmétique* entre deux valeurs numériques `a` et `b` :
  - *l'égalité* dont la syntaxe en `Python` est `a == b`
  - *la différence* dont la syntaxe en `Python` est `a != b`
  - *l'inégalité* dont la syntaxe en `Python` est `a < b` ou `a > b` ou `a <= b` ou `a >= b`
- d'une *opération logique* entre deux valeurs booléennes `x` ou `y` :
  - la *négation logique* dont dont la syntaxe en `Python` est `not x`
  - le *ou inclusif* dont la syntaxe en `Python` est `x or y`
  - le *et* dont la syntaxe en `Python` est `x and y`

Il existe des règles de priorité : opérateurs arithmétiques prioritaires sur les opérateurs logiques, `not` prioritaire sur `and` qui est prioritaire sur `or`. Des parenthèses permettent de changer les priorités.

1.16 Exercice 11 compléments sur les tests

Deviner les valeurs booléennes affichées par le programme ci-dessous puis vérifier en l'exécutant.

```
a = 504
b = 505
print(not (a > b))
print( a == b or a < b)
print( not ( a > 0 and a < b))
print((b - a) ** 2 == 1 and a != b - 1)
print((False or not False) and not(False and True))
```

1.17 Exercice 12 Calculs

```
In [ ]: from math import pi, exp, floor

In [57]: x = float(input('Entrez un réel x : '))
         print(x**5*exp(pi*floor(x)))

Entrez un réel x : 2.5
52294.10698484029
```

1.18 Exercice 12 Moyenne de notes

```
In [55]: n = int(input('Entre le nombre de notes : '))

Entre le nombre de notes : 3
```

```
In [56]: total = 0
         for k in range(n):
             total += float(input('Entrez la note {:d} '.format(k+1)))
         print("Moyenne arrondie au centième : {:02.2f}".format(total/n))


Entrez la note 1 14
Entrez la note 2 12.5
Entrez la note 3 10
Moyenne arrondie au centième : 12.17
```

1.19 Exercice 13

```
In [72]: pttc = float(input('Prix TTC (TVA à 19.6 %) au centime près : \n'))
         pht = pttc/1.196
         tva = pttc-pht
         print('Le prix HT est de {:.2f} et la TVA est de {:.2f}'.format(pht,tva))

Prix TTC (TVA à 19.6 %) au centime près :
852
Le prix HT est de 712.37 et la TVA est de 139.63
```

```
In [73]: pht = float(input('Entrez un prix  HT au centime près : \n'))
         taux = float(input('Entrez un taux de TVA sous la forme 19.6 : \n'))
         print('Le prix TTC est de {:.2f}'.format((1+taux/100)*pht))

Entrez un prix  HT au centime près :
852
Entrez un taux de TVA sous la forme 19.6 :
5.5
Le prix TTC est de 898.86
```

1.20 Exercice 14 Echange de deux variables de type entier

1.20.1 Méthode classique avec variable de stockage

```
In [48]: x = int(input('Entrez la valeur de la variable x : '))

Entrez la valeur de la variable x : 851
```

```
In [49]: y = int(input('Entrez la valeur de la variable y : '))

Entrez la valeur de la variable y : 852
```

```
In [50]: z = x
         x = y
         y = z

In [51]: x, y

Out[51]: (852, 851)
```

1.20.2 Méthode classique sans variable de stockage

```
In [52]: x = x + y
         y = x - y
         x = x - y

In [53]: x, y

Out[53]: (851, 852)
```

1.20.3 Méthode Pythonique avec affectation parallèle

```
In [78]: a = int(input('Entrez la valeur de la variable a : '))

Entrez la valeur de la variable a : 842


In [79]: b = int(input('Entrez la valeur de la variable b : '))

Entrez la valeur de la variable b : 843


In [80]: a, b = b, a

In [81]: a, b

Out[81]: (843, 842)
```

1.21 Exercice 15

```
In [86]: c = 1 - 1j

In [94]: type(c)

Out[94]: builtins.complex

In [92]: c.__class__

Out[92]: builtins.complex

In [96]: type(1j)

Out[96]: builtins.complex
```

```
In [88]: dir(c)

Out[88]: ['__abs__',
          '__add__',
          '__bool__',
          '__class__',
          '__delattr__',
          '__dir__',
          '__divmod__',
          '__doc__',
          '__eq__',
          '__float__',
          '__floordiv__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getnewargs__',
          '__gt__',
          '__hash__',
          '__init__',
          '__int__',
          '__le__',
          '__lt__',
          '__mod__',
          '__mul__',
          '__ne__',
          '__neg__',
          '__new__',
          '__pos__',
          '__pow__',
          '__radd__',
          '__rdivmod__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__rfloordiv__',
          '__rmod__',
          '__rmul__',
          '__rpow__',
          '__rsub__',
          '__rtruediv__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__sub__',
          '__subclasshook__',
          '__truediv__',
          'conjugate',
```

```
          'imag',
          'real']
```

```
In [89]: c.real, c.imag
```

```
Out[89]: (1.0, -1.0)
```

```
In [97]: def module(n):
             if type(n) == type(1j):
                 return ((n.real)**2 + (n.imag)**2)**(1/2)
             else:
                 return None
```

```
In [98]: module(c)
```

```
Out[98]: 1.4142135623730951
```

```
In [101]: def argument(n):
              import math
              if type(n) == type(1j):
                  real, imag = n.real, n.imag
                  if imag > 0:
                      return math.acos(real/module(n))
                  else:
                      return -math.acos(real/module(n))
              else:
                  return None
```

```
In [102]: argument(c)
```

```
Out[102]: -0.7853981633974484
```

```
In [103]: -math.pi/4
```

```
Out[103]: -0.7853981633974483
```

```
In [108]: c.conjugate()
```

```
Out[108]: (1+1j)
```

```
In [109]: argument(c.conjugate())
```

```
Out[109]: 0.7853981633974484
```

1.21.1 Exercice 16 compléments

- Le programme ci-dessous calcule la somme des `n` premiers entiers de 1 à `n`, la valeur entière strictement positive de `n` étant saisie par l'utilisateur. On notera l'utilisation de `k += 1` comme raccourci pour l'incrémentation `k = k + 1`. Recopier et compléter ce programme pour qu'il calcule également le produit des `n` premiers entiers consécutifs $1 \times 2 \times 3 \times \ldots \times n$. Tester pour $n = 100$, que peut-on dire de l'ordre de grandeur du produit ?

```
n = int(input('Entrez un entier n strictement positif : '))
somme = 0
for k in range(1, n + 1):
    somme += k
print(somme)
```

- Écrire un programme qui affiche les 10 premiers multiples de 2 sous la forme :

  2 x 1 = 2
  2 x 2 = 4
  ....
  2 x 10 = 20

- Modifier le programme précédent pour qu'il affiche les tables de mulplications pour les 10 premiers multiples de tous les entiers compris entre 1 et 9