Chapitre 0 : la boîte à outils pour commencer à "pythoner"

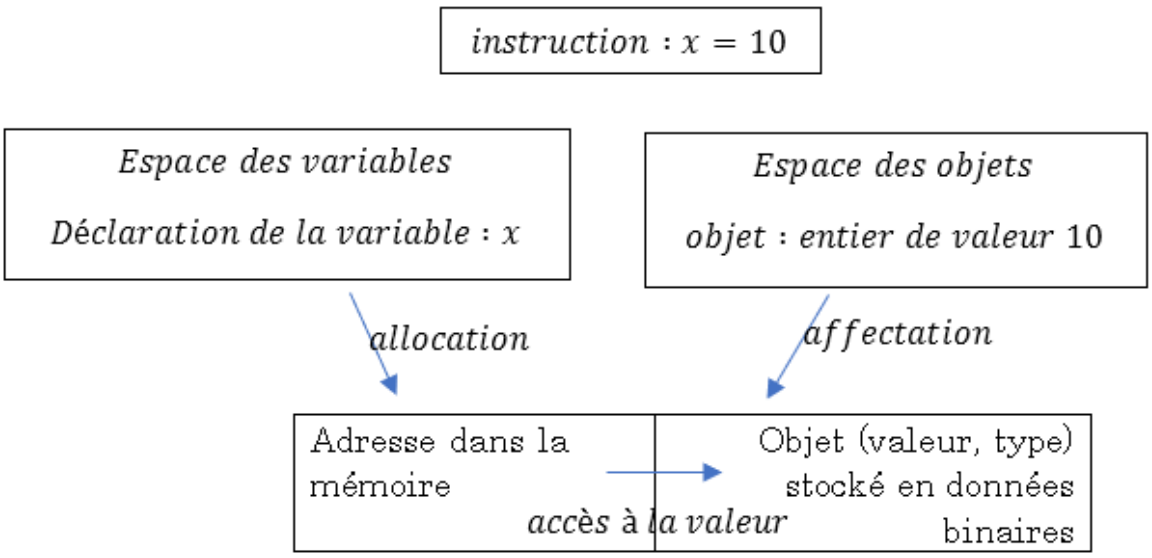
Quelques généralités :

- Pour installer python sous windows : http://winpython.github.io/ (http://winpython.github.io/)
- pour installer python sous mac ou lynux : https://pyzo.org/start.html (https://pyzo.org/start.html)
- plusieurs IDE sont possibles : spyder, pyzo, pycharm, jupyther, tonny,...
- Vous pouvez aussi travailler en ligne avec python tutor
- Vous pouvez travailler sur l'ENT avec Capytal

I- Opérateur d'affectation "="

Analysons les effets de l'instruction très simple $x = 10$:

- 1) Parmi tous les **objets** possibles (entiers, nombre décimaux, données textuelles, ...), c'est l'objet "entier 10" qui a été choisi (avec toutes ses propriétés mathématiques)
- 2) Une variable x est **déclarée**
- 3) Le signe "=" réalise **une allocation** (un espace réservé dans la mémoire) et **une affectation** (x est liée à la valeur 10 par l'adresse mémoire) : on dit que la variable x référence l'objet 10 dans la mémoire



In [10]:
```
1 # Exemple
2 x=10
3 print(id(x))#permet d'avoir l'adresse dans la mémoire de x
```

```
1997239839312
```

In [11]:
```
y=x
print(y,id(y))#même adresse, même valeur => permet une économie de
```

```
10 1997239839312
```

In [15]:
```
x=11.0#il est possible de faire un typage dynamique
print(x)
print(id(x))#Nouvelle allocaction et affectation de la varaibale x
#(la nouvelle affectation délie x de son ancienne valeur)
```

```
11.0
1997343293072
```

In [16]:
```
z=x+10
print(z)#on a accès à la valeur de x pour d'autres calculs
```

```
21.0
```

In [18]:
```
x=x+10#le typage dynamique permet aussi un lecture de l'ancienne va
print(x)
print(id(x))
```

```
22.0
1997343290928
```

In [5]:
```
#Rq : le x=x+10 et le x+=10
x=10
x+=10
print(x)#20
#cependant cette notation présente quelques subtilités
L=[0,1,2,3]
L=L+["a"]#fonctionne
print(L)
L+="a"#fonctionne !
print(L)
L=L+"a"#ne fonctionne pas
```

```
20
[0, 1, 2, 3, 'a']
[0, 1, 2, 3, 'a', 'a']
---------------------------------------------------------------------
-------
TypeError                                 Traceback (most recent cal
l last)
<ipython-input-5-7845a97b4050> in <module>()
      9 L+="a"#fonctionne !
     10 print(L)
---> 11 L=L+"a"#ne fonctionne pas

TypeError: can only concatenate list (not "str") to list
```

In [6]:
```
#Rq : la capy par référence a l'avantage de limité l'encombrement d
L=[0,1,2,3,4]
L_bis=L
L_bis.append(5)
print(L)#effet de bord
```

```
[0, 1, 2, 3, 4, 5]
```

II- Les opérations numériques sur les entiers et les nombres décimaux

In [8]:
```
x,y=10,3
#la somme
print(x+y)
```

```
13
```

In [9]:
```
#la soustraction
print(x-y)
```

```
7
```

In [10]:
```
#produit de x par y
print(x*y)
```

```
30
```

In [11]:
```
# x puissance y
print(x**y)
```

```
1000
```

Rappels sur la division enclidienne :

- le dividende a : quanitité à diviser
- le diviseur b : quantité par laquelle on divise
- le quotient Q, le reste R : a//b, a%b tels que a=b*Q+R

In [12]:
```
# Division de x par y
print(x/y)# on pourra remarquer que le nombre limité de chiffres si
#et l'erreur d'arrondi liée au principe
# de numérisation des nombres décimaux abouti à une erreur
# si on numérise sur 4 bits alors : 1*2^1+1*2^0+0*2^-1+1*2^-2=2+1+0
#on comprend qu'en limitant le nombre de bit sur ce type de nombre
```

```
3.3333333333333335
```

In [25]:
```
# Quotient Q de la division entière de x par y
print(x//y) # x = y*Q + R (où R est le reste, x est le dividende),
```

```
3
```

In [26]:
```
# Reste de la division de x par y
print(x%y)
```

```
1
```

In [27]:
```
# doublet (Quotient, Reste)
divmod(x,y)#un tuple est renvoyé
```

Out[27]:
```
(3, 1)
```

In [13]:
```
#valeur absolue
abs(-x)
```

Out[13]:
```
10
```

In [15]:
```
#Conversion d'un nombre décimal en entier le plus proche de 0
print(int(x/y))
#rq : int(3,9) donne 3 !
```

```
3
```

In [30]:
```
#conversion d'un entier en nombre décimal
print(float(x))
```

```
10.0
```

III- Les booléens

- x or y : Vrai si x et/ou y sont vrais

In [33]:
```
print(True or True)#True
print(False or False)# False
print(False or True)#True
print(True or False)#True
print(True or rien)#True car opérateur paresseux (pas d'évaluation
```

```
True
False
True
True
True
```

- x and y : Vrai si x et y sont vrais

In [34]:
```
print(True and False)#False
print(False and False)# False
print(False and True)#False
print(False and rien)#False car opérateur paresseux (pas d'évaluati
print(True and True)# True
```

```
False
False
False
False
True
```

- not x : Vrai si x faux, faux si x vrai

In [21]:
```
print(not False)#True
print(not True)#False
#les booléens peuvent être vis comme un sous type des entiers
print(1==True)#True
print(0==False)#True
print(not 0)#True
print(not 1)#False
print(not 10.4)#False (tout nombre différent de 0 peut être interpr
i=10
while i:
    i=i-1
    print(i)
```

```
True
False
True
True
True
False
False
9
8
7
6
5
4
3
2
1
0
```

A noter que ces opérateurs suivent un ordre de priorité (not>and>or>xor)

In [1]:
```
#XOR est python est obtenu avec ^
print(True^True)
print(True^False)
print(False^True)
print(False^False)
```

```
False
True
True
False
```

Certains opérateurs sur les types numériques renvoient un résultat booléen :

In [1]:
```
x=10
y=3
print(x<y)# x est-il plus strictement plus petit que y ?
print(x>y)# x est-il plus strictement plus grand que y ?
print(x<=y)# x est-il inférieur ou égale à y ?
print(x>=y)# x est-il inférieur ou égale à y ?
print(x==y)# x est-il identique à y ?
print (x!=y)# x est-il différent de y ?
```

```
False
True
False
True
False
True
```

IV- Les fonctions

La définition d'une fonction se fait à l'aide du mot clé *def* :

In [18]:
```
def nom_fonction(variable1,variable2):
    #corps de la fonction indenté
    return #pour renvoyer un résultat
```

In [22]:
```
#exemple f1(x)=2x+4
def f1(x):
    y=2*x+4
    return y
valeur = f1(2)# On appelle la fonction f1 et on récupère f1(2)
print(valeur)
#Pour généraliser à toute fonction affine
def f2(a,b,x):
    return a*x+b
print(f2(2,4,2))
```

```
8
8
```

In [23]:
```
#Rq : renvoyer et afficher un résultat :
# sous jupyter : si la cellule ne contient qu'une fonction alors le
# =>le print n'est pas n'est pas nécesaaire
f1(2)
```

Out[23]:
```
8
```

In [26]:
```
#le return permet de récupérer la valeur (mais on perd l'affichage)
a=f1(2)
```

On distingue les varaibles locales et globales :

- les varaibles globales ne sont pas définies dans le corps de la fonction. S'il s'agit d'un nombre, cette varaible ne peut être modifiée,

- les variables locales sont définies dans le corps de la fonction et leur lecture n'est possible que pendant l'appel de cette fonction

Si une variable est appelées dans le corps de la fonction alors elle est d'abord recherchées dans l'espace des varaibles locales puis globales

In [22]:
```
#exemple 1 :
a=2
def f3():
    print(a)
print(a)
f3()
print(a)
```

```
2
2
2
```

In [23]:
```
#exemple 2 :
a=2
def f4():
    a=3
    print(a)
print(a)
f4()
print(a)
```

```
2
3
2
```

In [24]:
```
#exemple 3 :
a=2
def f5():
    a=a+2#pose pb car sur la même ligne a est à la fois locale et g
    b=a+3
    return a,b
print(a)
print(f5())
print(a)
```

```
2

---------------------------------------------------------------------------
-------
UnboundLocalError                         Traceback (most recent cal
l last)
<ipython-input-24-efe74b3944c2> in <module>
      6     return a,b
      7 print(a)
----> 8 print(f5())
      9 print(a)

<ipython-input-24-efe74b3944c2> in f5()
      2 a=2
      3 def f5():
----> 4     a=a+2
      5     b=a+3
      6     return a,b

UnboundLocalError: local variable 'a' referenced before assignment
```

In [25]:
```
#exemple 4 :
a=2
def f5(a):
    a=a+2#pose pb car sur la même ligne a est à la fois locale et g
    b=a+3
    return a,b
print(a)
print(f5(a))#renvoie un tuple
print(a)
```

```
2
(4, 7)
2
```

V- Les chaînes de caractères

Les données textuelles oo chaînes de caractères ont les propriétés suivantes :

- création :

In [28]:
```
mot1="Bonjour"#création manuelle
mot2=' l\'ensemble de la classe'#nécessité de mettre un antislash =
phrase=mot1+mot2
print(phrase)#concaténation
```

```
Bonjour l'ensemble de la classe
```

- Conversion:

In [29]:
```
#Effectué par le mot clé $str$
phrase=phrase+" rentrée "+str(2023)#concaténation possible si de mé
print(phrase)
```

```
Bonjour l'ensemble de la classe rentrée 2023
```

- Sélection ou tranche

In [31]:
```
#sélection d'une lettre (indice de départ nul)
print(phrase[0])#le premier indice est à 0
#selection d'une sous chaîne [indice de départ : indice fin exclu]
print(phrase[0:7])
#remarque, on peut choisir le pas :
print(phrase[0:7:2])#ici pas de 2
```

```
B
Bonjour
Bnor
```

- longueur d'une chaîne

In [16]:
```
print(len(phrase))
```

```
44
```

VI- Structures conditionnelles

On a typiquement le squelette suivant :

In [9]:
```
x=1.1
if x >0 :#présence d'une indentation
    print ("x est strictement positif")#instruction réalisée si la
elif x<0 :
    print ("x est strictement négatif")#instruction réalisée si la
else :
    print("x est nul")#instruction réalisée si les deux autres conc
```

```
x est strictement positif
```

Rq : if et elif sont bien différents : un if est toujours testé, un elif et un else ne sont pas testés si le if précédent est True

In [17]:
```
x=11
if x >0 :#présence d'une indentation
    print ("x est strictement positif")#instruction réalisée si la
if x<10:
    print ("x est plus petit que 10")#instruction réalisée si la 2e
elif x<5 :
    print ("x est plus petit que 5")#instruction réalisée si la 2e
else :
    print("x est nul")#instruction réalisée si les deux autres conc
```

```
x est strictement positif
x est nul
```

In [1]:
```
x=11
if x>10:
    print("if")
elif x>10:
    print("elif")
elif x>10:
    print("elif2")
#si if OK, les autres ne sont pas testé
```

```
if
```

On peut cumuler des conditions, il ne faut pas oublier que c'est *if* condition:

In [14]:
```
x=5
if (x>0 and x<10):
    print("0<x<10")
```

```
0<x<10
```