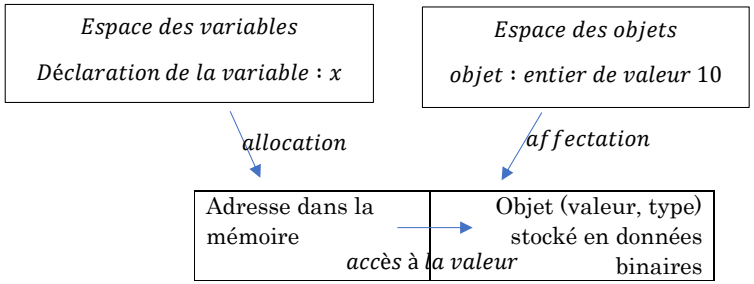
Chapitre 0 : la boîte à outils pour commencer à « Pythoner »

## I- Opérateur d'affectation " = " (DMK4L7)

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

1) Parmi tous les objets disponibles (entier, nombres décimaux, données textuelles, …), c'est l'objet « entier 10 » qui a été ici choisi (avec toutes ses propriétés mathématiques).
2) Une variable $x$ est déclarée.
3) Le signe " = " réalise une allocation et une affectation : on dit que la variable $x$ référence l'objet 10 dans la mémoire (dans laquelle un espace a été alloué). $x$ est liée à la valeur 10 par l'adresse mémoire.

*instruction* : $x = 10$




```
x=10
print(id(x)) # adresse mémoire : 1641978448
y=x# la valeur n'est pas copiée à une autre adresse mémoire
ce qui permet une économie de la mémoire
print(id(y),y) # adresse mémoire : 1641978448 et y=10
z=x+10
print(z) # z=20 on a accès à la valeur de x pour faire des
calculs
x=x+10 #x=20 la variable x=10 est lue et la nouvelle
affectation délie x de cette valeur qui a été mise
#à un autre emplacement mémoire (cela revient à faire une
nouvelle procédure d'affectation)
print(x)#20
print(id(x)) #1641978608
```

## II- Les opérations numériques sur les entiers et les flottants (99BZ9Y)

| Opérations | Résultats | Exemples : x,y=10,3 |
|---|---|---|
| $x + y$ | Somme de $x$ et $y$ | `print(x+y)# 13` |
| $x - y$ | Différence de $x$ et $y$ | `print(x-y)# 7` |
| $x * y$ | Produit de $x$ par $y$ | `print(x*y)# 30` |
| $x ** y$ | $x$ puissance $y$ : $x^y$ | `print(x**y)# 1000` |
| $x/y$ | Division de $x$ par $y$ | `print(x/y)# 3.3333333333333335 !!!` |
| $x//y$ | Quotient de la division entière de $x$ par $y$ | `print(x//y)# 3` |
| $x\%y$ | Reste de la division entière de $x$ par $y$ | `print(x%y)# 1` |
| $divmod(x,y)$ | Renvoie le doublet $(x//y, x\%y)$ | `print(divmod(x,y))# (3, 1)` |
| $abs(x)$ | Valeur absolue de $x$ | `print(abs(-x))# 10` |
| $int(x)$ | Conversion d'un nombre décimale en l'entier le plus proche de 0 | `print(int(x/y))# 3` |
| $float(x)$ | Conversion d'un entier en nombre décimal | `print(float(x//y))# 3.0` |

## III- Les booléens E7Y8ZQ

| | | |
|---|---|---|
| $x$ *or* $y$ | Vrai si $x$ et/ou $y$ sont vrais | `print(True or False)#True`<br>`print(True or rien)#True`<br>`(opérateur paresseux : pas d'évaluation du 2e opérande si le 1e est vrai)` |
| $x$ *and* $y$ | Vrai si $x$ et $y$ sont vrais tous les 2 | `print(True and True)#True`<br>`print(False and rien)#False` |
| *not* $x$ | Vrai si $x$ est faux et inversement | `print(not False and True)#True` |

Certains opérateurs sur les types numériques renvoient un résultat booléen :

| | | |
|---|---|---|
| $x < y$ | $x$ est-il strictement inférieur à $y$ ? | `print(2<3)#True` |
| $x <= y$ | $x$ est-il inférieur ou égal à $y$ ? | `print(2<=3)#True` |
| $x > y$ | $x$ est-il strictement supérieur à $y$ ? | `print(2>3)#False` |
| $x >= y$ | $x$ est-il supérieur ou égal à $y$ ? | `print(2>=3)#False` |
| $x == y$ | $x$ est-il égale à $y$ ? | `print(0.1+0.2==0.3)#F` |
| $x! = y$ | $x$ est-il différent de $y$ | `print(0.1+0.2!=0.3)#T` |

**Commenté [AM1]:** Pour installer Python sur votre machine personnelle :
-Sous windows : http://winpython.github.io/
-Sous Mac ou linux : https://pyzo.org/start.html
Utiliser ensuite comme IDLE (Integrated DeveLopment Environment) ou éditeur spyder ou pyzo afin d'éviter de travailler directement sur l'interpréteur. Cet éditeur propose une auto-complétion des mots que vous êtes en train d'écrire, il ajoute automatiquement les indentations, une coloration syntaxique. Une fois écrit dans l'éditeur, le programme peut être sauvegardé au format .py

**Commenté [AM5]:** Attention à na pas écrire $2(4 + 3)$ au lieu de $2 * (4 + 3)$

**Commenté [AM2]:** L'instruction affectation modifie l'espace mémoire de l'ordinateur à la différence d'une expression mettant en jeu un calcul et retournant une valeur.

**Commenté [AM6]:** On rappelle le vocabulaire de la division entière ou euclidienne entre deux entiers naturels : $a = bq + r$

**Commenté [AM3]:** Il ne faut donc pas dire que « $x$ c'est 10 » mais que $x$ fait référence à une adresse mémoire dans laquelle la valeur 10 a été mémorisée (ce qui permet

**Commenté [AM7]:** On vient ici de typer la variable $x$ (on dit aussi caster)

**Commenté [AM8]:** Nous traiterons en profondeur la représentation machine des nombres flottants. On pourra

**Commenté [AM9]:** Le type booléen peut être vu comme un sous-type des entiers.
`print(True==1)#True`

**Commenté [AM4]:** Python autorise donc un typage dynamique : on peut rependre le nom $x$ pour référencer un objet de type différent

**Commenté [AM10]:** A ne pas confondre avec une procédure d'affectation.

**Commenté [AM11]:** Résultat surprenant ! En fait, plus un nombre décimal (codé de manière approchée) est petit, meilleur sera sa précision ainsi 0,1+0,2>0,3

## IV- Les chaînes de caractères

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

| | |
|---|---|
| Création | `mot1="Bonjour"`<br>`mot2='Vieljeux'`<br>`phrase = mot1+" "+mot2#concaténation`<br>`print(phrase)#Bonjour Vieljeux` |
| Conversion | Effectuée par le mot clé *str* :<br>`phrase+=" "+str(2023)#Bonjour Vieljeux 2023` |
| Sélection ou tranche | `print(phrase[0])#B`<br>`print(phrase[0:7])#Bonjour` |
| Longueur | `print(len(phrase))#21` |

## V- Structures conditionnelles(54AXL3)

On a typiquement le squelette suivant :

| | | |
|---|---|---|
| Si<br>Indentation | `if x>=0:`<br>`  print("x est plus grand que 0")` | 1e condition<br>Bloc exécuté si 1e condition vraie |
| Sinon<br>Indentation | `elif x>100:`<br>`  print("x est plus grand que 100")` | 2e condition<br>Bloc exécuté si vrai et si 1e condition fausse |
| Sinon<br>Indentation | `else :`<br>`  print("x est négatif")` | Bloc d'instruction exécuté si aucune condition n'est vraie |

## VI- Les fonctions (0XD95W)

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

```
def nom_fonction (variable1,variable2) :
    corps de la fonction effectuant les opérations souhaitées
    return résultat souhaité
```

Définissons la fonction $f_1$ permettant d'obtenir $f_1(x) = 2x + 4$ pour tout $x$ :

```
def f1(x):
    y=2*x+4
    return y
```

Pour appeler cette fonction et récupérer la valeur en $x = 2$, on écrit :

```
y=f1(2)#ainsi y=8
```

Pour généraliser à toute fonction affine, on peut imaginer :

```
def f2(a,b,x):
    return a*x+b
print(f2(2,4,2))#8
```

On distingue les variables globales et locales :

| Variables globales | Variables locales |
|---|---|
| - Elles sont définies en dehors de la fonction.<br>- S'il s'agit d'un nombre alors l'appel de la fonction ne pourra pas modifier cette variable. | - Elles sont définies dans le corps de la fonction.<br>- Les variables introduites dans le corps de la fonction ne sont accessibles que lors de l'exécution de la fonction. |

Si une variable est appelée dans le corps de la fonction alors elle est d'abord recherchée dans l'espace des variables locales puis globales.

```
a=2
def f3():
    print(a)
print(a)#2
f3()#2
print(a)#2
```

```
a=2
def f4():
    a=3
    print(a)
print(a)#2
f4()#3
print(a)#2
```

```
a=2
def f5(a):
    a=a+2
    b=a+3
    return a,b
print(a)#2
print(f5(3))#(5,8)
La fonction renvoie une structure de données immuable appelée tuple
print(a)#2
```

```
a=2
def f5():
    a=a+2#conflit car "on" demande » d'être à la fois locale et globale
    print(a)
```

**Commenté [AM12]:** Petite préférence pour les guillemets double :
`phrase2='L\'année commence bien'`
`phrase3="L'année commence bien"`

**Commenté [AM13]:** A noter que mot1*3 concatène trois fois mot1

**Commenté [AM17]:** On a donc deux espaces bien disjoints entre les variables locales et locales en ce qui concerne les variables associées nombres qui sont donc immuables (l'application d'une fonction ne change pas leur valeur).
A noter que les listes (structure de données) ne suivent pas cette règle : c'est un objet mutable

**Commenté [AM14]:** L'indentation est obligatoire en Python et permet de définir une séquence d'instructions qui sera effectuée selon la valeur de la condition.

**Commenté [AM15]:** La structure de base est
If #se réalise si True
elif#de réalise si True et if False
else# se réalise si tout faux

Avec deux if, si le premier if est True et que le 2e if est False alors il y une « réinitialisation » et on revient au cas de base.

**Commenté [AM18]:** Comme nous le verrons dans le prochain chapitre, comme les listes, les tuples sont itérables et indiçable. A la différence d'une liste, un tuple est non mutable.
Pour récupérer les valeurs (dépaquetage du tuple), on écrirait ici $valeur1, valeur2 = f5(a)$

**Commenté [AM16]:** Le mot clé return est optionnel car une fonction peut servir à afficher à modifier une liste sans pour autant la retourner