Informatique

August 25, 2026

1 Chapitre 0 : la boîte à outils pour commencer à "pythoner" - Code capytal : 0c13-1780985

Quelques généralités :

- Pour installer python sous Windows : http://winpython.github.io/
- pour installer python sous macOs ou Linux : https://pyzo.org/start.html
- plusieurs IDE sont possibles : spyder, pyzo, pycharm, jupyter, Thonny,...
- Vous pouvez aussi travailler en ligne avec Python Tutor
- Vous pouvez travailler sur l'ENT avec Capytale

1.1 I- Opérateur d'affectation "=" (code quizinière : DMK4L7)

En langage Python, l'opérateur d'affectation "=" permet de lier une variable à un objet (entier, nombre décimal, données textuelles, liste,...)

[32]:
```
# on choisit un nom de variable : ici x

#
```

[33]:
```
#affichage :


#
```

[34]:
```
#affichage avec print


#
```

[35]:
```
# Une fois définie, on peut utiliser cette variable pour d'autres calculs:


#
```

[36]:
```

```

```
# On a un typage dynamique qui permet de faire évoluer le type de x au cours du␣
 ↪programme
#(un même nom peut être successivement associé à des objets de types différents)␣
 ↪:



#
```

Rq :Un nom de variable :

- peut contenir des lettres, des chiffres et _ ;
- ne peut pas commencer par un chiffre ;
- ne peut contenir ni espace, ni tiret, ni caractère spécial ;
- est sensible aux majuscules et minuscules

1.2 II- Les opérations numériques sur les entiers et les nombres décimaux (code quizinière : 99BZ9Y)

[37]:
```
x,y=10,3
#la somme
print(x+y)
```

```
13
```

[38]:
```
#la soustraction
print(x-y)
```

```
7
```

[39]:
```
#produit de x par y
print(x*y)
```

```
30
```

[40]:
```
# x puissance y



#
```

Rappels sur la division euclidienne : - le dividende a : quantité à diviser - le diviseur b : quantité par laquelle on divise - le quotient Q, le reste R : tels que a=b*Q+R

[41]:
```
# Division de x par y



#
```

[42]:
```
# Quotient Q de la division entière de x par y
```

```
#
```

[43]:
```
# Reste de la division de x par y


#
```

[44]:
```
# doublet (Quotient, Reste)


#
```

[45]:
```
#valeur absolue


#
```

[46]:
```
#Conversion d'un nombre décimal en entier le plus proche de 0



#
```

[47]:
```
#conversion d'un entier en nombre décimal


#
```

1.3 III- Les booléens (code quizinière : E7Y8ZQ)

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

[48]:
```
#(True or True)#True
#(False or False)# False
#(False or True)#True
#(True or False)#True
#(True or variable_inconnue)#True car opérateur paresseux (pas d'évaluation du␣
↪2nd opérande si le 1e est vrai)
```

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

[49]:
```
#(True and False)#False
#(False and False)# False
#(False and True)#False
#(False and rien)#False car opérateur paresseux (pas d'évaluation du 2nd␣
↪opérande si le 1e est faux)
#(True and True)# True
```

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

[50]:
```
#(not False)#True
#(not True)#False
```

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

[51]:
```
#

#
```

Certains opérateurs sur les types numériques renvoient un résultat booléen :

[52]:
```
x=10
y=3
print(x<y)# x est-il strictement plus petit que y ?
print(x>y)# x est-il strictement plus grand que y ?
print(x<=y)# x est-il inférieur ou égal à y ?
#




#
```

```
False
True
False
```

1.4 IV- Les fonctions (code quizinière : OXD95W)

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

[53]:
```
def nom_fonction(variable1,variable2):
    #corps indenté de la fonction
    return #pour renvoyer un résultat
```

[54]:
```
#exemple f1(x)=2x+4




#
```

Pour exécuter une fonction, on l'appelle en donnant une valeur à chacun de ses arguments.

[55]:
```
#

#
```

Avec le mot clé *return*, on peut récupérer le résultat et l'affecter dans une autre variable pour un calcul ultérieur

[56]:
```
#

#
```

[57]:
```
#Pour généraliser à toute fonction affine, on peut rajouter d'autres arguments



#
```

On distingue les variables locales et globales :

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

[58]:
```
# En général, on commence un programme par les variables globales (par exemple␣
↪des constantes physiques)



#
```

[59]:
```
def chute2(t):
    G=9.81
    return 0.5*G*t**2
print(G)# la variable locale n'est pas accessible en dehors de l'appel de la␣
↪fonction
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[59], line 4
      2     G=9.81
      3     return 0.5*G*t**2
----> 4 print(G)# la variable locale n'est pas accessible en dehors de l'appel de␣
↪la fonction

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

1.5 V- Les chaînes de caractères

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

- création :

[60]:
```
#

```

```
#
```

- Conversion:

[61]:
```
#Effectué par le mot clé $str$


#
```

[62]:
```
# On pourra signaler les f-string:


#
```

- Sélection ou tranche

[63]:
```
#




#
```

- longueur d'une chaîne

[64]:
```
#

#
```

1.6 VI- Structures conditionnelles (code quizinière : 54AXL3)

On a typiquement le squelette suivant :

[65]:
```
#




#
```

Rq : if et elif sont bien différents : Plusieurs instructions if successives sont toutes testées. En revanche, dans une structure if–elif–else, Python s'arrête dès qu'une condition est vraie. Le bloc else est exécuté uniquement si aucune des conditions précédentes n'est vraie.

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

[66]:
```
x=5
if (x>0 and x<10):
    print("0<x<10")
```

```
0<x<10
```

[67]:
```
x=5
if 0<x<10 :
    print("0<x<10")
```

```
0<x<10
```

[ ]:
```

```