Python-ECG1-02-fonctions-if-cor

September 17, 2021

1 TP2 - Fonctions et instruction if

1.1 1- Définir et utiliser une fonction

Une **fonction** est définie de cette façon :

```
def nom_fonction(argument1, argument2, etc.):
    bloc_instructions
    return valeur_renvoyée_1, valeur_renvoyée_2, etc.
```

- Attention à l'**indentation** et aux **deux points** à la fin de la première ligne.
- **nom_fonction** est le nom de la fonction qui sera utilisé ensuite. Pas d'espace, pas de caractère spécial. On utilisera des noms de fonctions explicites.
- **argument1, argument2,...** sont les paramètres d'entrées. Ce sont les variables de la fonctions. Ne pas mettre de valeur ici, on garde des noms génériques.
- Le mot clé **return** est essentiel. Il permet de définir le résultat final. **L'exécution d'une fonction s'arrête dès que l'on tombe sur return** et la fonction renvoie ce résultat.

Traitons un exemple. Exécutez les cellules suivantes.

[1]:
```
def fonction_f(x) :
    y = 3*x**2 - 2*x + 1
    return y
```

Pour **faire appel** à cette fonction, on donnera son nom et la valeur des paramètres d'entrée (ici x).

[2]:
```
fonction_f(2)
```

[2]:
```
9
```

Les variables x (paramètre d'entrée) et y (définie dans la fonction) sont des **variables locales**. Elle n'existent que dans cette fonction. Elles sont créés temporairement lors de l'exécution de la fonction mais ne restent pas en mémoire.

[3]:
```
x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
<ipython-input-3-6fcf9dfbd479> in <module>
----> 1 x

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

[4]:
```
y
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-9063a9f0e032> in <module>
----> 1 y

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

[5]:
```
x = 3
fonction_f(5)
print("x =", x)
```

```
x = 3
```

La valeur de x n'a pas été changée par l'exécution de la fonction.

1.1.1 Exercice 1

Définir une fonction **EurosToLivres(PE)** qui, étant donné un nombre PE correspondant à un prix en euros, renvoie le prix associé PL en livres Sterling. 1 EUR = 0.85 GBP

[6]:
```
def EurostoLivres(PE):
    PL = PE * 0.85
    return PL
```

Tester votre fonction.

[7]:
```
EurostoLivres(10000)
```

[7]:
```
8500.0
```

Écrire la fonction de conversion inverse, puis tester.

[8]:
```
def LivresToEuros(PL):
    PE = PL/0.85
    return PE
```

[9]:
```
LivresToEuros(10000)
```

[9]:
```
11764.705882352942
```

1.2 2 - Différence entre print et return

print(...) est une fonction permettant d'**afficher** des valeurs ou des messages.

return est un mot-clé utilisable **uniquement dans une fonction** qui permet de **renvoyer** une valeur.

Attention : impossible d'utiliser return hors de la définition d'une fonction.

Voyons la différence, exécutez les cellules suivantes et observez.

[10]:
```
# Voici trois fonctions qui semblent faire la même chose, on va vérifier.
def f(x):
    y = 2*x**2 + 1

def g(x):
    y = 2*x**2 + 1
    print(y)

def h(x):
    y = 2*x**2 + 1
    return y
```

[11]:
```
f(2)
```

[12]:
```
f(2)+5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-e5eeb1bc5d27> in <module>
----> 1 f(2)+5

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

[13]:
```
g(2)
```

```
9
```

[14]:
```
g(2)+5
```

```
9
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-9e1d57568627> in <module>
----> 1 g(2)+5

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

[15]:
```
h(2)
```

[15]: 9

[16]:
```
h(2)+5
```

[16]: 14

1.3 3 - Instruction if

Nous allons voir comment écrire un programme python comportant une instruction condition- nelle du type **si ... alors ...**

Algorithme en français

- **Si** condition 1,
  - Faire l'instruction 1
- **Sinon, si** condition 2,
  - Faire l'instruction 2
- ...
- **Sinon**
  - Faire l'instruction n
- **Fin du Si**

Syntaxe python

```
if condition 1 :
    instruction 1
elif condition 2 :
    instruction 2
elif condition 3 :
    instruction 3
...
else :
    instruction n
```

Remarques :

- Les **conditions** sont des booléens ;
- les deux points indiquent la fin de chaque condition;
- ne pas oublier l'indentation des blocs d'instructions par rapport à la ligne if, elif ou else dont ils dépendent
- la fin de l'indentation indique la fin de l'instruction conditionnelle (pas de "fin du si" explicite).

[17]:
```
# Exemple
x = 17  # tester avec différentes valeurs de x

if x > 15:
```

```
    print("x est strictement supérieur à 15")
elif x >= 11:
    print(" x est entre 11 et 15")
else :
    print(" x est inférieur ou égal à 10")
```

```
x est strictement supérieur à 15
```

1.3.1 Exercice 2

Écrire une suite d'instructions en langage Python qui, étant donnés des nombres a et b, affiche lequel est le plus petit des deux (par exemple "a est le plus petit") ou s'il y a égalité.

[18]:
```
# Compléter avec des valeurs de a et b puis changer pour tester tous les cas de␣
 ↪figure.
a = 5
b = 2
# votre programme
if a < b :
    print("a est le plus petit")
elif b < a :
    print("b est le plus petit")
else:
    print("a=b")
```

```
b est le plus petit
```

1.3.2 Exercice 3

Écrire une fonction prenant en entrée trois nombres a,b,c (avec $a \neq 0$) et renvoyant les éventuelles solutions *réelles* de l'équation $ax^2 + bx + c = 0$.

[19]:
```
def second_degre(a,b,c):
    Delta = b**2 - 4*a*c # discriminant
    if Delta > 0 :
        x1 = (-b-Delta**(1/2))/(2*a)
        x2 = (-b+Delta**(1/2))/(2*a)
        return x1,x2
    elif Delta == 0 :
        x0 = -b/(2*a)
        return x0
    else:
        return "pas de solution"
```

[20]:
```
#Tester votre fonction.
second_degre(2, -6,4)
```

[20]:
```
(1.0, 2.0)
```

[21]:
```
second_degre(2,12,18)
```

[21]:
```
-3.0
```

[22]:
```
second_degre(1,1,1)
```

[22]:
```
'pas de solution'
```

1.3.3 Exercice 4

Écrire une fonction qui, étant donnée une année, renvoie **True** si celle-ci est bissextile et renvoie **False** sinon.
Une année est bissextile (366 jours) si l'un des deux cas suivants se produit : - l'année est divisible par 4 et non divisible par 100 ; - l'année est divisible par 400.

Sinon, elle est non-bissextile (365 jours).

[29]:
```
def bissextile(annee):
    if (annee % 4 == 0 and annee %100 != 0) or (annee % 400 == 0):
        return True
    else:
        return False
```

[30]:
```
bissextile(2020)
```

[30]:
```
True
```

[31]:
```
bissextile(2021)
```

[31]:
```
False
```

[32]:
```
bissextile(2000)
```

[32]:
```
True
```

[33]:
```
bissextile(2100)
```

[33]:
```
False
```