Cours

P6 : LES FONCTIONS

En informatique, une **fonction** est un ensemble d'instructions regroupées sous un **nom** et s'exécutant à la demande. Les fonctions sont utiles pour :

- éviter la répétition (philosophie DRY « Don't Repeat Yourself ») ;
- permettre la réutilisation ;
- décomposer une tâche complexe en tâches plus simples (« diviser pour mieux régner »).

Python possèdent un certain nombre de fonctions prédéfinies telles que `print`, `range`, `len`. De très nombreuses fonctions sont également disponibles via des **modules** (comme `numpy`) et il est possible de définir ses propres fonctions.

I Définir une fonction

La syntaxe générale de définition d'une fonction en Python est :

```
def nom_fonction(arg1, arg2,..., argn):
    bloc.................................
    ........................d'instructions
# ici, on est hors de la définition de la fonction.
```

où `arg1, arg2,..., argn` sont les **arguments (ou paramètres) formels** d'entrée et le bloc d'instructions le **corps** de la fonction.

Entrée [1]:
```
def carre(x):
    return x**2
```

Entrée [2]:
```
def bonjour(prenom, classe):
    print('Bonjour', prenom, ', bienvenue en', classe)
```

!! Attention !! le rôle d'une définition de fonction n'est pas d'exécuter les instructions qui en composent le corps, mais uniquement de mémoriser ces instructions en vue d'une exécution ultérieure (facultative !), provoquée par une expression faisant **appel** à la fonction. Par exemple, définir la fonction qui suit ne provoque pas d'erreur :

Entrée [3]:
```
def ChuckNorris():
    print(1/0)
```

En revanche, l'appeler en provoque une, on va le voir plus loin !

II Utiliser une fonction

II.1 Appel d'une fonction

Une fois la fonction définie, on l'appelle à l'aide de la syntaxe :

```
nom_fonction(exp1, exp2,..., expn)
```

où `exp1, exp2,..., expn` sont des **expressions**. Lors de l'appel de la fonction `exp1, exp2,..., expn` sont évaluées, puis les résultats sont affectés à `arg1, arg2,..., argn` (dans cet ordre !) juste avant l'exécution du corps de la fonction. [1]

1. L'exécution de la fonction commence donc par l'instruction `arg1, arg2,..., argn = exp1, exp2,..., expn`.

Entrée [4]:
```
carre(3)
```

Out [4]:

Entrée [5]:
```
carre(2+3)
```

Out [5]:

Entrée [6]:
```
bonjour('Alice', 'ECG1')
```

Out [6]:

!! Attention !! S'il manque un argument lors de l'appel, Python génère une erreur :

Entrée [7]:
```
bonjour('Bob')
```

Out [7]:
```
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: bonjour() missing 1 required positional argument: 'classe'
```

Si une fonction n'a pas d'argument, son appel doit quand même comporter des parenthèses :

Entrée [8]:
```
ChuckNorris()
```

Out [8]:
```
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "<input>", line 2, in ChuckNorris
ZeroDivisionError: division by zero
```

Sans parenthèses, la fonction n'est pas appelée :

Entrée [9]:
```
ChuckNorris
```

Out [9]:
```
<function ChuckNorris at 0xc40d00>
```

Remarque : la fonction `rd.random()` du module `numpy.random` (ici avec l'alias `rd`) est également une fonction sans argument. Elle renvoie un nombre aléatoire de l'intervalle $[0, 1]$.

II.2 Valeur renvoyée

Une fonction peut éventuellement **renvoyer** (ou **retourner**) une valeur. Pour ce faire, il faut utiliser l'instruction `return` :

Entrée [10]:
```
def cube(x):
    return x**3
```

Entrée [11]:
```
cube(3)
```

Out [11]:

Le résultat renvoyée peut alors être stocké dans une variable :

Entrée [12]:
```
res = cube(2)
print(res)
```

Out [12]:

ou utilisé dans une expression :

Entrée [13]:
```
3*cube(2)+1
```

Out [13]:

!! Attention !! Lors de l'appel d'une fonction et de l'éxécution des instructions la composant, la fonction s'arrête dès qu'elle rencontre un return.

Entrée [14]:
```
def multiplie(a, b):
    x = a*b
    return x
    x = a+b
    return x
```

Entrée [15]:
```
multiplie(2, 3)
```

Out [15]:

Ceci sera pratique pour arrêter l'exécution d'une boucle.

Entrée [16]:
```
# Version 1 avec un for
def possede_un_1(liste):
    '''renvoie True si le nombre 1 est dans la liste et False sinon'''
```

Entrée [17]:
```
# Version 2 avec un while
def possede_un_1(liste):
    '''renvoie True si le nombre 1 est dans la liste et False sinon'''
```

II.3 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.

```
# Voici trois fonctions qui semblent faire la même chose, vérifions.
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
```

Entrée [18]:
```
f(2)
```

Out [18]:

Entrée [19]:
```
f(2) + 5
```

Out [19]:

Entrée [20]:
```
g(2)
```

Out [20]:

Entrée [21]:
```
g(2) + 5
```

Out [21]:

Entrée [22]:
```
h(2)
```

Out [22]:

Entrée [23]:
```
h(2) + 5
```

Out [23]:

II.4 Portée des variables

Dans un programme, la **portée d'une variable** définit les endroits du programme où la variable est accessible. En Python, la portée d'une variable est définie par l'endroit dans le code où la variable est définie.

Une variable créée à l'intérieur d'une fonction est **locale** : elle n'est accessible que dans le bloc d'instructions de cette fonction et est détruite une fois sorti de cette dernière.

Entrée [24]:
```
def afficher_42():
    n = 42
    print(n)
```

Entrée [25]:
```
afficher_42()
```

Out [25]:

Entrée [26]:
```
n
```

Out [26]:
```
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'n' is not defined
```

Autre exemple :

Entrée [27]:
```
def fonction_f(x):
    y = 3*x**2 - 2*x + 1
    return y
```

Entrée [28]:
```
x = 3
y = 4
fonction_f(5)
print("x =", x)
print("y =", y)
```

Out [28]:

Le fait que les variables des fonctions soient locales nous autorise à utiliser les mêmes noms de paramètres ou de variables dans des fonctions différentes.

III Exercice

1. Écrire une fonction `factorielle(n)` qui renvoie la valeur de $n!$ pour $n \in \mathbb{N}$.

2. En déduire une fonction `binomial(k,n)` qui renvoie $\binom{n}{k}$ pour $k, n \in \mathbb{N}$, $k \leqslant n$.