Les fonctions.

Nous avons déjà vu beaucoup de fonctions : `print()`, `type()`, `len()`, `int(input())`, `range()`... Ce sont des fonctions prédéfinies.
Nous avons aussi la possibilité de créer nos propres fonctions !

Intérêt des fonctions.

Une fonction est une portion de code que l'on peut appeler au besoin (c'est une sorte de sous-programme).

L'utilisation des fonctions évite des redondances dans le code : on obtient ainsi des programmes plus courts et plus lisibles.

Par exemple, nous avons besoin de convertir à plusieurs reprises des degrés Celsius en degrés Fahrenheit :

```
>>> print 100.0*9.0/5.0+32.0
212.0
>>> print 37.0*9.0/5.0+32.0
98.6
```

La même chose en utilisant une fonction :

```
def F(DegreCelsius):
    print(DegreCelsius*9.0/5.0+32.0)

F(100)
F(37)
x=233
F(x)
```

L'instruction *def.*

```
def NomDeLaFonction(parametre1,parametre2,parametre3,...):
    """ Documentation
    qu'on peut écrire
    sur plusieurs lignes """    # docstring entouré de 3 guillemets (ou apostrophes)

    bloc d'instructions         # attention à l'indentation

    return resultat             # la fonction retourne le contenu de la variable resultat
```

Exemple n°1 : La fonction suivante simule le comportement d'un dé à 6 faces. Pour cela, on utilise la fonction `randint()` du module `random`.

```
def TirageDe():
    """ Retourne un nombre entier aléatoire entre 1 et 6 """
    import random
    valeur = random.randint(1,6)
    return valeur


print(TirageDe())
print(TirageDe())
help(TirageDe)
```

Le retour de valeurs.

ATTENTION !!! Ne pas confondre return et print.

• L'instruction print n'a pas de valeur, elle a pour seul effet d'afficher un texte à l'écran.

• L'instruction return, au contraire, n'affiche rien mais décide de ce que renvoie la fonction, et donc de la valeur de l'appel f(...).

Ainsi, si on commet l'erreur d'utiliser print à la place de return dans une fonction, celle-ci affichera la valeur calculée à l'écran mais ne la renverra pas (elle renverra None comme on l'a vu plus haut).

```
def puissance(x, n):
    if (x == 0):
        return 0
    r = 1
    for i in range(n):
        r = r * x
    return r

x=float(input())
n=int(input())
print(puissance(x,n))
```

Si la valeur de x est nulle, l'instruction return 0 interrompra le déroulement de la fonction et aucune des instructions suivantes (à partir de r = 1) ne sera exécutée.

Exemple n°2 : Une fonction qui affiche la parité d'un nombre entier.

Il peut y avoir plusieurs instructions `return` dans une fonction. **L'instruction `return` provoque le retour immédiat de la fonction.**

```
# définition de fonction
def Parite(nombre):
    """ Affiche la parité d'un nombre entier """
    if (nombre % 2) == 1:   # L'opérateur % donne le reste d'une division
        print(nombre,'est impair')
        return
    if (nombre % 2) == 0:
        print(nombre,'est pair')
        return

# début du programme
Parite(13)
Parite(24)
```

Exemple n°3 : Une fonction avec deux paramètres :

```
def TirageDe2(valeurMin,valeurMax):
    """ Retourne un nombre entier aléatoire entre valeurMin et valeurMax """
    import random
    return random.randint(valeurMin,valeurMax)

# début du programme
for i in range(5):
    print(TirageDe2(1,10))  # appel de la fonction avec les arguments 1 et 10
```

Exemple n°4 : Une fonction qui retourne une liste .

```
# définition de fonction
def TirageMultipleDe(NbTirage):
    """ Retourne une liste de nombres entiers aléatoires entre 1 et 6 """
    import random
    return [random.randint(1,6) for i in range(NbTirage)]

# début du programme
print(TirageMultipleDe(20))
help(TirageMultipleDe)
```

Portée de variables : variables globales et locales.

La portée d'une variable est l'endroit du programme où on peut accéder à la variable.

En Python, on distingue deux sortes de variables : les globales et les locales.

Observons le script suivant :

```
a = 10
def MaFonction():
    a = 20
    print(a)
    return(a)

print(a)
MaFonction()
print(a)
```

La variable ***a*** de valeur 20 est créée dans la fonction : c'est une variable locale à la fonction. Elle est détruite dès que l'on sort de la fonction.

L'instruction ***global*** rend une variable globale :

```
a = 10      # variable globale

def MaFonction():
    global a    # la variable est maintenant globale
    a = 20
    print(a)
    return(a)

print(a)
MaFonction()
print(a)
```

Remarque : il est préférable d'éviter l'utilisation de l'instruction `global` car c'est une source d'erreurs (on peut modifier le contenu d'une variable sans s'en rendre compte, surtout dans les gros programmes).

Exemple 5 : Quelles sont les variables locales et globales de la fonction f ? Qu'affiche le programme suivant ?

```
def f():
    global a
    a = a + 1
    c = 2 * a
    return a + b + c

a = 3
b = 4
c = 5
print(f())
print(a)
print(b)
print(c)
```

Ordre d'évaluation.

Considérons les deux scripts suivants :

```
n = 0
def g(x):
    global n
    n = n + 1
    return x + n

def somme(x, y):
    return x + y

print(somme(n, g(1)))
```

```
n = 0
def g(x):
    global n
    n = n + 1
    return x + n

def somme(x, y):
    return x + y

print(somme(g(1),n))
```

Quelles sont les valeurs affichées dans chaque cas ?

Comment remédier à ce problème ?

Exemple 6 : Qu'affiche le programme suivant ? Pourquoi ?

```
def f(x):
    global b
    b=10
    return(x + 1)

a=3
b=4
print(f(a)+b)
```

Proposer une adaptation de ce programme dans laquelle le résultat ne dépend pas de l'ordre d'évaluation.