



L'essentiel

- De nombreuses fonctions sont regroupées par thème dans des **modules**.
- Dans le module **random**, on trouve les fonctions :
 - random()** : renvoie un nombre flottant aléatoire entre 0 et 1. (Notez que les parenthèses vides sont obligatoires.)
 - randint(n,m)** : renvoie un entier aléatoire dans l'intervalle $[n,m]$ avec équiprobabilité.
- Différentes manières d'importer un module pour utiliser les fonctions :

```
from random import randint
de = randint(1,6)
print("Tirage :", de)
```

```
from random import *
de = randint(1,6)
print("Tirage :", de)
```

```
import random
de = random.randint(1,6)
print("Tirage :", de)
```

```
import random as rd
de = rd.randint(1,6)
print("Tirage :", de)
```

- ◇ Dans la première version : on importe la fonction `randint` que l'on pourra ensuite utiliser comme n'importe quelle autre fonction.
 - ◇ Dans la seconde version : même principe, mais avec `*` on importe toutes les fonctions du module.
 - ◇ Troisième méthode : on importe le module et non directement la fonction. Il faut alors faire précéder le nom de la fonction du nom du module. (Utile lorsque 2 fonctions ont le même nom dans 2 modules différents.)
 - ◇ Dernière méthode : même principe, mais on donne un nom au module. (Pratique si le nom du module est long.)
- Le module **math** contient (entre autres) :
 - ◇ **pi** : la constante π ;
 - ◇ **sqrt** : la fonction racine carrée;
 - ◇ **floor** : la fonction partie entière;
 - ◇ **sin**, **cos** et **tan** : les fonctions trigonométriques, ainsi que leurs réciproques : **asin**, **acos** et **atan**;
 - ◇ **exp** : la fonction exponentielle, et sa réciproque **log** la fonction logarithme népérien.
- Pour connaître la liste des fonctions d'un module, il suffit de taper dans la console : `help("math")`.
- Pour obtenir des informations sur une fonction, même principe : `help("math.gcd")`.
- Les variables créées à l'intérieur d'une fonction sont des **variables locales** : une fois que l'on sort de la fonction, elles n'existent plus :

```
def f(x) :
    y = x + 1
    return y

x, y = 3, 7
z = f(x)
print(x, y, z)
```

Affiche

3 7 4

```
def f(x) :
    x = x + 1
    return x

x, y = 3, 7
z = f(x)
print(x, y, z)
```

Affiche

3 7 4

```
def f(x) :
    x = y + 1
    return x

x, y = 3, 7
z = f(x)
print(x, y, z)
```

Affiche

3 7 8

```
def f(x) :
    y = y + 1
    return y

x, y = 3, 7
z = f(x)
```

Affiche

```
Traceback (most recent call last):
  File "<module3>", line 7, in <module>
    File "<module3>", line 3, in f
UnboundLocalError:
  local variable 'y' referenced before assignment
```

- À retenir :
 - ◇ Les **variables globales** (celles du script) sont accessibles en lecture, mais pas en écriture dans une fonction : on peut lire leurs valeurs, mais pas les modifier.
 - ◇ Les **variables locales** sont détruites au moment de la sortie de la fonction, il n'y a donc pas de conflit avec une variable globale qui porterait le même nom.

Exercices du jour

Exercice 1 : Créer une fonction `lancer` qui ne reçoit pas de paramètre, mais simule le lancer de deux dés et renvoie la somme des deux faces.

Exercice 2 : On souhaite modéliser le lancer d'une pièce truquée qui tombe sur Pile avec une probabilité de $\frac{1}{4}$ et sur Face avec une probabilité de $\frac{3}{4}$. Écrire une fonction `piece` qui ne reçoit pas de paramètre, mais renvoie "Pile" ou "Face" en respectant les probabilités demandées.

Exercice 3 : Pour le script suivant, dire à chaque ligne ce que valent les variables locales et globales `a`, `b` et `c` si elles existent. Préciser ce qui est affiché dans la console en dernière ligne. Vous pouvez vérifier avec

<https://pythontutor.com/>.

```
1 def f(a) :  
2     if a > b :  
3         return a  
4     else :  
5         return a + b  
6  
7 a, b = 1, 2  
8 c = f(b)  
9 b = f(a)  
10 a = f(c)  
11 print(a, b, c)
```

Ligne	Globales			Locales	
	a	b	c	a	b
7					
8					
9					
10					

Exercices en autonomie

Exercice 4 : Deux exercices de tirage aléatoire :

- Créer une fonction `aleatoire(N)` qui reçoit un entier naturel non nul `N` et renvoie au hasard un nombre entier de l'intervalle $[1; N]$.
- Créer une fonction `aleatoire_multiple(m, N)` qui reçoit deux entiers naturels `m` et `N` non nuls et renvoie au hasard un nombre entier multiple de `m` de l'intervalle $[1; N]$.

Exercice 5 : On considère l'expérience aléatoire qui consiste à tirer un entier dans $\{0, 1, 2\}$ selon la distribution de probabilité suivante :

k	0	1	2
$P(A_k)$	a	b	c

avec $a + b + c = 1$ et où A_k est l'événement « Tirer l'entier k ».

Écrire une fonction qui reçoit en paramètre les 3 réels a , b et c , et retourne une valeur aléatoire en respectant la distribution de probabilités.

Aide pour les exercices

Solutions des exercices

Correction 4 1.

```
from random import randint

def aleatoire(N) :
    return randint(1, N)
```

2. Une solution à l'aide de la fonction précédente :

```
def aleatoire_multiple(m, N):
    return m * aleatoire(N//m)
```

Correction 5 Faites une figure pour mieux comprendre ou demandez de l'aide :

```
def tirage(a, b, c) :
    x = random()
    if x < a :
        return 0
    if x < a + b :
        return 1
    return 2
```