Chapitre 1
Algorithmique et programmation 2

I. Les fonctions en Python

R

1. Comment un informaticien/mathématicien fait bouillir de l'eau avec une casserole vide, un robinet et une plaque chauffante ?
   Il remplit la casserole, il fait chauffer la casserole jusqu'à ébullition.
2. Comment un informaticien/mathématicien fait bouillir de l'eau avec une casserole vide, un robinet et une plaque chauffante ?
   Il vide la casserole et ré-applique la réponse de la question 1.

En informatique, on essaie d'être assez fainéant mais productif, c'est à dire lorsque l'on a fait quelque chose on essaie de le réutiliser. Pour ce faire on utilise des fonctions. Comme en mathématiques, *une fonction c'est juste quelque chose en entrée et qui donne quelque chose en sortie.* En informatique parfois ce quelque chose c'est rien.

1. Généralités

En Python, il est possible de définir des ***fonctions***. Les fonctions en programmation permettent d'utiliser de multiple fois une séquence d'instructions qui dépendent d'un certain nombre de ***paramètres*** (on parle aussi d'***arguments***). La notion de fonction en informatique diffère légèrement de la notion de fonction en mathématiques qui est présentée la semaine suivante. D'ailleurs, elle peut varier d'un langage de programmation à l'autre. On se limite ici uniquement au cas des fonctions en Python.

En Python, la définition d'une fonction passe par la définition :

— du nom de la fonction ;
— de sa liste de paramètres en entrée (qui peut être vide) ;
— de la séquence d'instruction dans la fonction ;
— de la sortie de la fonction (qui est facultative en Python).

Elle prend forme suivante :

```
1 def nom_fonction(a,b,c):
2     instructions
3     return sortie
```

Ici, la première ligne comporte :

— la commande `def` qui indique qu'une fonction va être définie ;

— `nom_fonction` qui désigne le nom de la fonction (choisie par l'utilisateur) ;
— des variables `a`, `b` et `c` qui sont les paramètres de la fonction (dont le nom est choisi par l'utilisateur), ils sont placés entre parenthèses et séparés les uns des autres par des virgule (il peut y en avoir n'importe quel nombre entier, y compris 0, dans ce cas on écrit `nom_fonction()`) ;
— un deux points.

Cette première ligne est suivie, après indentation, de la séquence d'instructions de la fonction qui se termine généralement (mais pas nécessairement) par un ***renvoi*** de la fonction indiqué par la commande `return`. Le renvoi permet de définir une valeur renvoyée en sortie. Le code d'une fonction peut contenir plusieurs renvois correspondant à différents cas (en utilisant des instructions conditionnelles, par exemple). Toutefois, l'exécution de la fonction n'effectue jamais qu'un seul envoi.

▪ **Exemple 1.1.**
La fonction `f` prend un paramètre `x` et renvoie la valeur obtenue par l'instruction `2*x+3`. En mathématiques, elle correspond à une fonction affine de la forme $f : x \mapsto 2x+3$.

```
1 def f(x):
2     return 2*x+3
```

▪

▪ **Exemple 1.2.**
La fonction `puissance` prend deux paramètres `a` et `n` et renvoie en sortie `a` à la puissance `n`.

```
1 def puissance(a,n):
2     p = 1
3     for k in range(n):
4         p = p * a
5     return p
```

On remarque que l'indentation de la boucle s'ajoute à l'indentation de la définition de la fonction. ▪

▪ **Exemple 1.3.**
La fonction `mini` prend deux paramètres `a` et `b` et renvoie le plus petit des deux. Ici le renvoi dépend d'une instruction conditionnelle.

```
1 def mini(a,b):
2     if a<b:
3         return a
4     else:
5         return b
```

▪

■ **Exemple 1.4.**
La fonction `bonjour` affiche `Bonjour!` à l'écran. Elle ne prend aucun paramètre et ne fait aucun renvoi [1].

```
def bonjour():
    print("Bonjour!")
```

■

2. Fonctions prédéfinies

Dans Python, un certain nombre de fonctions sont déjà prédéfinies. Elles peuvent donc être utilisées directement sans avoir à les redéfinir. Quelques unes de ces fonctions ont déjà été évoquées dans ce cours (`print`,`input`). Le tableau suivant contient une liste de fonctions prédéfinies en Python qui pourront vous servir dans le cadre de ce cours. Cette liste n'est bien évidemment pas exhaustive et vous pourrez découvrir les autres fonctions prédéfinies en Python en consultant la documentation disponible via le lien suivant : https://docs.python.org/3/.

Fonction	Description
`print(x)`	Affiche la valeur de `x` dans la console.
`print(txt,x)`	Affiche le texte de la chaîne de caractères `txt` suivi de la valeur de `x` dans la console.
`input(txt)`	Affiche le texte de la chaîne de caractères `txt` dans la console. L'utilisateur doit alors taper une valeur dans la console qui sera renvoyée par la fonction `input` dès que l'utilisateur appuie sur la touche *Entrée* du clavier.
`int(x)`	Renvoie la valeur de `x` convertie en entier lorsque cela est possible.
`float(x)`	Renvoie la valeur de `x` convertie en flottant lorsque cela est possible.
`len(txt)`	Renvoie le nombre de caractères dans la chaîne de caractères `txt`.
`abs(x)`	Renvoie la valeur absolue de `x`.
`round(x,n)`	Renvoie la valeur arrondie de `x` à 10^{-n} près.

■ **Exemple 1.5.**
Dans le code Python suivant, on utilise les fonctions `float`, `input`, `print` et `round`.

```
x = float(input("Veuillez rentrer une valeur pour x"))
print("La valeur approchee de x a 0,01 pres vaut",round(x,2))
```

1. attention à ne pas confondre affichage et renvoi!

Remarquez l'encapsulation de la fonction `input` dans la fonction `float`. Ceci est nécessaire car la fonction `input` ne permet que de récupérer le texte taper par l'utilisateur (même si ce texte comporte des nombres). Ainsi, si l'on n'utilise pas la fonction `float` et que l'utilisateur rentre la valeur `123.3` avant de valider par la touche Entrée, c'est la chaîne de caractère `"123.3"` qui sera stockée dans la variable `x`. La fonction `float` permet alors de convertir la chaîne de caractère `"123.3"` en `123.3` qui est un nombre flottant. ■

Bibliothèques

En plus des fonctions prédéfinies qui sont accessibles directement, Python propose également un certain nombres de ***bibliothèques*** (aussi appelés ***modules***) qui contiennent de nombreuses fonctions déjà programmées (ainsi que d'autres objets prédéfinis). Pour utiliser les fonctions d'une bibliothèque, il faut d'abord ***importer*** la bibliothèque. La manière la plus simple pour ce faire est d'utiliser la commande `import` suivi du nom de la bibliothèque.

■ **Exemple 1.6.**
L'instruction suivante permet d'importer la bibliothèque standard `math`.

```
1 import math
```

■

Pour utiliser une fonction d'une bibliothèque (ou n'importe quel autre type d'objet d'une bibliothèque), il suffit alors d'utiliser la syntaxe `nom_bibliotheque.nom_fonction`.

■ **Exemple 1.7.**
Le code suivant permet d'importer la bibliothèque `math` dans Python pour utiliser la fonction `sqrt` qui permet de calculer une racine carrée[2]. Ici, on affiche la valeur de $\sqrt{2}$.

```
1 import math
2 print(math.sqrt(2))
```

■

Une autre façon d'importer les fonctions et objets d'une bibliothèque est d'utiliser une instruction suivant la syntaxe ci-dessous :

```
1 from nom_bibliotheque import *
```

Cette instruction permet d'importer toutes les fonctions et objets d'une bibliothèque de manière à ce qu'ils puissent être utilisés sans rappeler le nom de la bibliothèque.

■ **Exemple 1.8.**
Ici, on importe `math` pour utiliser la valeur approchée `pi` de π qui est enregistrée dans la bibliothèque `math`.

2. `sqrt` est une abréviation de *square root* qui signifie *racine carrée* en anglais.

```
from math import *
print(pi)
```

■

Il existe un certain nombre d'autres façons d'importer des bibliothèques en Python (en entier ou partiellement) qui ne seront pas détaillées ici. Vous pourrez les retrouver dans la documentation Python.

En plus de la bibliothèque `math`, on utilise également dans ce cours la bibliothèque `random` (pour faire des simulations aléatoires) et la bibliothèque `mathplotlib.pyplot` (pour faire des représentations graphiques). Les fonctions des bibliothèques seront présentées au fur et à mesure de leur utilisation dans le cours ou les exercices.

3. Fonctions aléatoires

Le langage Python permet également la définition de ***fonctions aléatoires***. En informatique, une fonction aléatoire est une fonction dont la valeur renvoyée est le résultat d'un tirage aléatoire. Cette valeur n'est donc pas déterminée de manière unique par la valeur des paramètres en entrée de la fonction. La bibliothèque `random` en Python contient un certain nombre de fonctions aléatoires prédéfinies et notamment la fonction `randint` qui renvoie un entier compris entre deux valeurs fournies en paramètres de manière aléatoire et selon une loi de probabilité uniforme[3].

■ **Exemple 1.9.**
Dans le code suivant, on importe les fonctions de la bibliothèque `random` afin de tirer au hasard soit 0, soit 1.

```
from random import *
print(randint(0,1))
```

■

Simulation

Une ***simulation*** est un processus qui calque un phénomène réel dans le but d'en prédire l'issue. Le résultat d'une simulation peut ainsi informer sur le résultat possible du phénomène. On utilise généralement les simulations en informatique pour recréer un phénomène qui coûterait trop cher à réaliser physiquement (comme pour une simulation de crash-test d'avion) ou qui est tout simplement impossible à réaliser physiquement (comme pour une simulation de l'évolution climatique du globe terrestre).

■ **Exemple 1.10.**
On peut utiliser la fonction `randint` comme précédemment pour simuler le lancer d'une pièce qui peut tomber sur pile (que l'on fait correspondre au 0) ou face (que l'on fait correspondre au 1). Si un lancer de pièce est facile à réaliser physiquement, le lancer d'un million de pièces par exemple devient plus compliqué alors qu'en simulation cela ne met que quelques secondes sur un ordinateur moderne. Dans le code Python

3. Comme on le verra au cours de la semaine **??**, cela signifie que toutes les valeurs peuvent sortir avec la même probabilité

suivant, une simulation permet de comptabiliser combien de fois une pièce tombe sur face en un million de lancers. C'est la variable p qui permet de stocker cette valeur.

```
from random import *
p=0
for k in range(1000000):
    p = p + randint(0,1)
```

■

Exercices non à soumettre

Exercice 1.1.
Déterminez les renvois des fonctions `f` et g définies ci-dessous pour les appels `f(2)`, `f(0)`, `g(1)` et `g(3)`

```
def f(x):
    y = x**2-3
    return y
```

```
def g(x):
    y = x**2-3
    z = 3*x-y
    y = z + x
    return z
```

Exercice 1.2.
On considère les trois fonctions Python suivantes.

```
def f1(x,y):
    z = 0
    while z<20:
        z = x+y
        x = 2*x
        y = 3*y
        z = y*z
    return z
```

```
def f2(x,y):
    z = 0
    while z<20:
        z = x+y
        x = 2*x
        y = 3*y
    z = y*z
    return z
```

```
def f3(x,y):
    z = 0
    while z<20:
        z = x+y
        x = 2*x
    y = 3*y
    z = y*z
    return z
```

Détaillez les calculs de `f1(5,1)`, `f2(5,1)` et `f3(5,1)`. Vous listerez l'ensemble de toutes les affectations qui sont réalisées en indiquant si celles-ci ont lieu :
- avant le début de la boucle ;
- dans la boucle et, dans ce cas, au combientième tour de la boucle ;
- après la sortie de la boucle.

Exercice 1.3.

Le tableau suivant donne la mention en fonction de la note N obtenue à la première session du baccalauréat.

Condition	Mention
$0 \leq N < 8$	AJOURNE(E)
$8 \leq N < 10$	RATTRAPAGE
$10 \leq N < 12$	SANS MENTION
$12 \leq N < 14$	ASSEZ BIEN
$14 \leq N < 16$	BIEN
$16 \leq N$	TRES BIEN

Définissez une fonction Python qui prend une note en paramètre et renvoie la mention correspondante sous forme de chaîne de caractère. ■

■ **Exercice 1.4.**
À quelle notion mathématique correspond la fonction Python définie ci-dessous ?

```
1 from math import *
2
3 def fonction_mystere(a,b,c,d):
4     return sqrt((a-c)**2+(b-d)**2)
```

■

■ **Exercice 1.5.**
On considère la fonction Python suivante qui prend comme paramètres six variables qui correspondent aux coordonnées des sommets d'un triangle ABC dans un repère orthonormé et renvoie un couple de valeurs (les couples sont des objets qui existent en Python et sont notés à l'aide de parenthèses comme en mathématiques).

```
1 def G(xA,yA,xB,yB,xC,yC):
2     xG=(xA+xB+xC)/3
3     yG=(yA+yB+yC)/3
4     return (xG,yG)
```

1. On appelle cette fonction en rentrant les paramètres correspondant aux points $A(2;1)$, $B(-2;5)$ et $C(0;-3)$. Que renvoie-t-elle ?
2. On considère maintenant que A, B et C sont trois points quelconques du plan. Montrez que le point G correspondant au renvoi de la fonction `G` en Python est défini par l'équation vectorielle suivante :

$$3\overrightarrow{OG} = \overrightarrow{OA} + \overrightarrow{OB} + \overrightarrow{OC}$$

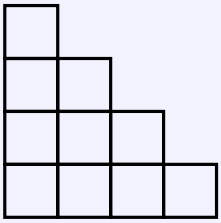
3. Montrez que cette équation vectorielle est équivalente à :

$$\overrightarrow{GA} + \overrightarrow{GB} + \overrightarrow{GC} = \overrightarrow{0}$$

4. À quoi sert cette fonction Python ?

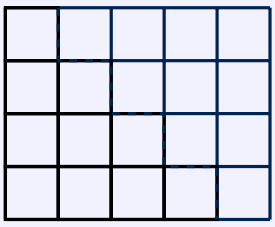
■

■ **Exercice 1.6.**
On considère un escalier (en deux dimensions) formés de carré comme dans la représentation graphique ci-dessous.



On appelle n le nombre de carrés situés à la base de l'escalier. Le but de l'exercice est de déterminer le nombre total m de carrés qui composent l'escalier en fonction de n. Dans l'exemple, $n = 4$ et $m = 10$.

1. Si on part du sommet pour arriver à la base, chaque étage possède un carré supplémentaire par rapport au précédent. En vous appuyant sur cette idée, écrivez une fonction Python qui permet de calculer m en fonction de n en parcourant une boucle.
2. On complète le schéma précédent en rajoutant un escalier inversé de la même forme que le premier de la manière suivante :



Déduisez-en une expression algébrique de m en fonction de n.
3. Quelle approche privilégieriez-vous entre les deux ? Justifiez votre réponse.

■

■ **Exercice 1.7.**
Algorithme : Première puissance de a inférieure (ou supérieure) à b. On cherche ici à définir un algorithme qui permette d'obtenir la première puissance entière positive d'un nombre réel positif a inférieure ou égale à un autre nombre réel positif b. Autrement dit, quelle est la plus petite valeur de $n \in \mathbb{N}^*$ telle que $a^n \leq b$.

1. Quelle valeur de n convient si $a \leq b$?
2. On considère par la suite que $b < a$.

(a) En considérant, $a = 4$ et $b = \frac{1}{2}$, montrez qu'un tel n n'est pas forcément défini.
(b) Discutez en fonction de la position relative de a par rapport à 1 de l'existence d'un tel n.
(c) On suppose de plus que $a < 1$. Complétez le programme Python suivant qui permet de déterminer la valeur souhaitée dans le cas où $a = 0,9$ et $b = 0,02$.

```
a = 0.9
b = 0.02

n = 1
p = a
while p > b:
    n = ...
    p = ...
```

3. En utilisant les différents éléments que vous avez trouvés, écrivez une fonction Python qui permettent de répondre à l'énoncé du problème.
4. Écrivez une fonction pour le cas où l'on recherche la première puissance de a supérieure à b.

■

■ Exercice 1.8.

Algorithme : Test de primalité. Le but de cet exercice est de déterminer si un entier $p > 1$ est premier ou non.

1. Rappelez la définition d'un nombre premier.
2. Que vaut `(p%k==0)` lorsque `k` est un diviseur de `p` ? Et sinon ?
3. Expliquez pourquoi le code suivant permet bien de déterminer si un nombre est premier.

```
def premier1(p):
    for k in range(2,p):
        if (p%k == 0):
            return False
    return True
```

4. Montrez que l'on peut arrêter la boucle après avoir testé tous les diviseurs potentiels de p inférieurs ou égaux à $\sqrt{p}$.
5. Complétez le code suivant pour intégrer cette information. On pourra utiliser la fonction `sqrt` du module `math` et faisant attention au fait que `range` doit prendre en entrée des paramètres de type `int`.

```
def premier2(p):
        n = ...
    for k in range(2,n):
        if (p%k == 0):
            return False
    return True
```

6. Testez chacune des fonctions pour déterminer si 999999937 est premier. Vous devriez trouver que c'est un nombre premier dans les deux cas : c'est le plus grand nombre premier à 9 chiffres.
7. Les deux fonctions ne mettent pas le même temps à répondre. Estimez le nombre de tours parcourus dans la boucle dans chacun des cas.
8. Testez maintenant les fonctions pour déterminer si 999999938 est premier. Cette fois-ci les fonctions ont mis à peu près le même temps. Pourquoi ?

■

■ **Exercice 1.9.**

1. Complétez le code suivant de la fonction `lancer` pour qu'elle simule le lancer d'un dé équilibré à six faces.

```
from random import *

def lancer():
    return randint(...,...)
```

2. On souhaite réaliser une simulation du lancer de 1000 dés. Complétez le code suivant et exécutez le à la suite du code précédent afin de pouvoir remplir le tableau des effectifs correspondant à la série statistique de la simulation.

```
n = 1000
n1 = 0
n2 = 0
n3 = 0
n4 = 0
n5 = 0
n6 = 0

for k in range(n):
    x = lancer()
    if x == 1:
        n1 = n1 + 1
    elif x == 2:
        n2 = n2 + 1
    ...
    else:
        n6 = n6 + 1
```

Face	1	2	3	4	5	6
Effectif						

3. Écrivez une fonction `moyenne` qui calcule la moyenne de la valeur obtenue lors d'un lancer à partir des valeurs obtenues lors de la simulation.

■

■ **Exercice 1.10.**

Dans cet exercice, on réalise la ***simulation du saut de puce***. C'est un problème classique en mathématiques qui appartient à une classe de problèmes très importants connus sous le nom de ***marches aléatoires***. Les marches aléatoires ont des applications multiples dans de nombreux domaines allant de l'informatique à l'économie en passant par la biologie. Elles sont beaucoup utilisées, par exemple, pour faire se déplacer des personnages dans des jeux vidéos. Les marches aléatoires peuvent être définies de manière très complexe mais on ne considère ici que la version la plus simple qu'est le saut de puce. On imagine qu'une puce se déplace en faisant des sauts le long d'un axe gradué dans un sens ou dans l'autre. On appelle x la position de la puce sur l'axe gradué. La puce saute toutes les secondes exactement et le saut de la puce mesure toujours exactement la même longueur d'unité 1. Si l'on suppose que la puce commence à la position $x = 0$ de l'axe gradué, après 1 seconde, elle sera soit à $x = -1$, soit à $x = 1$. Enfin, le sens dans lequel la puce saute à chaque fois est parfaitement aléatoire : la puce a autant de chance de sauter vers la droite que vers la gauche.

1. Complétez le code Python suivant où l'on définit une fonction `saut` qui prend en paramètre correspondant à la position de la puce avant un saut et renvoie la position de la puce après le saut.

```
from random import *

def saut(x):
    a = randint(0,1)
    if a == 1:
        return ...
    else:
        return ...
```

2. On suppose que la puce commence toujours à la position $x = 0$. Réalisez une fonction `deplacement` qui simule le déplacement de la puce pendant n secondes et renvoie sa position finale (c'est-à-dire après n sauts). Vous pourrez utiliser la fonction `saut` déjà définie.
3. Montrez qu'après 4 sauts, la position x de la puce appartient nécessairement à $\{-4;-2;0;2;4\}$.
4. Exécutez la commande `deplacement(4)` vingt fois de suite et relevez les effectifs correspondant à l'occurrence de chacune des positions finales possibles dans le tableau suivant.

x	-4	-2	0	2	4
Effectif					

5. Quelle est la valeur moyenne de la position finale donnée par cette série statistique?
6. On voudrait réaliser une série statistique similaire mais, cette fois-ci, avec un effectif total beaucoup plus important : $1,000,000$ par exemple. Faire un programme Python qui génère une telle série statistique et déterminez les fréquences obtenues dans ce cas.
7. Quelle est la valeur moyenne de cette nouvelle série statistique?