Algorithmique et programmation en Python

I. — Variable et types

Une variable est un espace mémoire désigné par une étiquette (un nom). Lorsqu'on appelle une variable nommée A dans une instruction (par exemple dans l'addition A+3 ou dans la comparaison A>0), l'instruction s'effectue en fait sur la valeur affectée à la variable A c'est-à-dire à la valeur contenue dans l'espace mémoire étiqueté par A.

Affecter une valeur v à une variable A, c'est stocker la valeur v dans l'espace mémoire étiqueté par A. L'affectation d'une valeur v à une variable A se fait de la manière suivante :

en langage naturel	en Python
A ← v	`A=v`

Remarque. Lors qu'on affecte la variable A, soit elle existait déjà et alors son ancienne valeur est supprimée et remplacée par la nouvelle valeur affectée soit la variable A est créée avec la valeur affectée.

ATTENTION ! En Python, l'ordre est essentiel lors de l'affectation. Pour affecter la valeur `v` à la variable `A`, on ne peut pas écrire `v=A`.

En Python, les données sont typées c'est-à-dire que leurs valeurs sont associées à un certain type. Trois des principaux types sont : les entiers (`int`), les réels (`float`) et les chaînes de caractères (`str`). Lors de l'affectation d'une donnée v à une variable A, la variable A prend automatiquement le type de la donnée v.

Par défaut,

- les nombres entiers sont de type `int`. Ainsi, `A=3` affecte l'entier 3 à la variable `A`.
- les nombres réels écrits « en flottant » (c'est-à-dire avec un point) sont de type `float`. Ainsi, `A=3.0` ou `A=3.` affecte le réel 3 à la variable `A`.
- une chaîne de caractère encadrée par des apostrophes ou des guillemets est de type `str`. Ainsi, `A='année2020'` ou `A="année2020"` affecte la chaîne de caractère `année2020` à la variable `A`.

Il existe, de plus, dans Python trois fonctions `int`, `float` et `str` qui permettent dans une certaine mesure de transformer une donnée d'un certain type en une donnée d'un autre type. Par exemple, `float(145)` transforme l'entier 145 en le réel 145.0 et `str(145)` transforme l'entier 145 en la chaîne de caractères `'145'`. On peut de même transformer un réel en chaîne de caractères.

Par exemple, `A=int(3)`, `A=float(3)` et `A=str(3)` sont trois instructions qui affectent la valeur 3 à la variable A mais dans le premier cas 3 est considéré comme un entier, dans le second comme un réel et dans le troisième comme un caractère (et non pas un nombre).

Le typage a une importance cruciale lorsqu'on fait des opérations sur les variables car certaines opérations ne sont possibles que sur des variables du même type ou car une même opération donnera des résultats différents selon le type des variables.

Attention cependant,

- la fonction `int` appliquée à un réel arrondit celui-ci par défaut s'il est positif et par excès s'il est négatif. Ainsi, `int(1.999)` renvoie `1` et `int(-3.999)` renvoie `-3` ;
- on ne peut pas appliquer les fonctions `int` ou `float` à une chaîne de caractères contenant des lettres. On peut en revanche écrire `int('123')` qui renvoie l'entier `123` ou `float('123.45')` qui renvoie le réel `123.45`.

Remarque. Pour connaître le type d'une donnée ou d'une variable, on peut utiliser la fonction `type` qui renvoie `<class 'int'>`, `<class 'float'>` ou `<class 'str'>` selon les cas.

II. — Opérations élémentaires

1) addition et soustraction

Les opérateurs `+` et `-` correspondent à l'addition et à la soustraction habituelles sur les nombres (entiers ou réels). Ainsi, `3+4` renvoie l'entier `7` et `4.6+1.25` renvoie le réel `5.85`.

De plus, si on additionne ou soustrait un entier et un réel, l'entier est automatiquement converti en réel et le résultat est un réel. Par exemple, `3+2.6` renvoie le réel `5.6`.

Même si le résultat est entier, une addition ou une soustraction contenant un réel renvoie toujours un réel. Ainsi, `3.5+1.5` renvoie le réel `5.0`.

L'opérateur `+` est également défini sur les chaînes de caractères. Il revient à accoler les deux chaînes l'une derrière l'autre : c'est ce qu'on appelle la concaténation. Ainsi, `'bonne'+'année'` renvoie la chaîne `bonneannée`.

Remarque. On ne peut pas additionner un nombre (de type `int` ou `float`) avec une chaîne de caractères.

2) Multiplication

L'opérateur `*` correspond à la multiplication habituelle sur les nombres (entiers ou réels). Ainsi, `3*4` renvoie l'entier `12` et `4.3*2.5` renvoie le réel `10.75`.

Comme pour l'addition, on peut multiplier un entier et un réel et alors l'entier est automatiquement converti en réel et le résultat est un réel. Par exemple, `3*2.5` renvoie le réel `7.5`.

On peut également multiplier une chaîne `s` de caractères par un entier n. Le résultat est une chaîne vide si $n \leqslant 0$ et la chaîne $\underbrace{\texttt{ss...s}}_{n \text{ fois}}$ si $n \geqslant 1$. Ainsi, `'ab'*3` renvoie la chaîne `ababab`.

3) Divisions

L'opérateur `/` correspond à la division habituelle sur les nombres (entiers ou réels). Il renvoie toujours un réel. Ainsi, `12/3` renvoie le réel `4.0`, `3/4` renvoie le réel `0.75` et `4.5/1.5` renvoie le réel `3.0`.

Les opérateurs `//` et `%` correspondent au quotient et au reste dans la division euclidienne des entiers. Ainsi, `14//3` renvoie l'entier `4` et `14%3` renvoie l'entier `2` car $14 = 4 \times 3 + 2$.

4) Puisssance

L'opérateur `**` correspond à l'exposant. Ainsi, `2**3` renvoie l'entier `8` (c'est-à-dire 2^3) et `2.5**2` renvoie le réel `6.25` (c'est-à-dire $2{,}5^2$).

5) À propos des calculs sur les nombres réels en Python

Les calculs sur les nombres réels en Python se font en fait sur des représentations de ces nombres qui ne sont pas toujours exactes. De ce fait, les calculs sur les réels sont en fait des calculs approchés est produisent parfois des résultats étonnants. Ainsi, un calcul aussi simple que `0.1+0.2` renvoie `0.30000000000000004` !

Nous ne détaillerons pas ici les raisons de ce résultat mais il faut garder à l'esprit que les calculs sur les réels en Python ne sont le plus souvent pas exacts...

III. — Saisie et affichage en Python

Dans un éditeur Python, si on écrit `2*0.1` et qu'on exécute, nous ne verrons rien. Pour voir le résultat de l'opération, il faut préciser qu'on souhaite qu'il soit affiché. Pour cela, on utilise la fonction `print`.

Ainsi, l'instruction `2*0.1` effectue le calcul mais ne l'affiche pas à l'écran alors que `print(2*0.1)` effectue le calcul et affiche à l'écran `0.2`.

De même, si on veut afficher le type d'une variable A, il faudra écrire `print(type(A))`.

Il se peut que dans un programme, on souhaite demander des valeurs à l'utilisateur. Dans ce cas, on utiliser la fonction `input`. Sans précision, la valeur saisie par l'utilisateur est, par défaut, considérer comme étant de type `str`.

Pour demander à l'utilisateur de saisir un entier et affecter la valeur saisie à une variable A, on écrit `A=int(input())`. On peut, si on le souhaite, rajouter un texte à afficher, par exemple `A=int(input('Saisir un entier A'))` ou `A=int(input('A= ?'))`

IV. — Instruction conditionnelle

Une instruction conditionnelle est une instruction du type

Si (condition C) alors (bloc d'instructions 1) sinon (bloc d'instructions 2)

ce qui signifie : si la condition C est remplie, effectuer le bloc d'instructions 1 et si elle n'est pas remplie, effectuer le bloc d'instructions 2.

Remarque. La partie « sinon (bloc d'instructions 2) » n'est pas obligatoire. Si on l'omet alors aucune action n'est effectuée si la condition C n'est pas remplie.

La syntaxe d'une instruction conditionnelle est la suivante.

en langage naturel	en Python

Si (Condition C)
  bloc d'instructions 1
Sinon
  bloc d'instructions 2
Fin Si
	if (Condition C):
	  bloc d'instructions 1
	else:
	  bloc d'instructions 2

Remarque ESSENTIELLE. Pour que la syntaxe soit correcte en Python, il est indispensable que :

1. la ligne commençant par `if` et celle contenant `else` se termine par deux points (`:`)
2. les blocs d'instructions 1 et 2 soient écrits en retrait par rapport à rapport aux autres lignes. Ce retrait est appelé l'indentation.

Voici un exemple. Que fait cet algorithme/ce programme ?

en langage naturel	en Python

Si $x \geqslant 0$
  A $\leftarrow x$
Sinon
  A $\leftarrow -x$
Fin Si
	`if (x>=0):`
	`  A=x`
	`else:`
	`  A=-x`

Dans la condition C, on aura souvent besoin de faire des tests. Pour cela, on peut utiliser les syntaxes suivantes :

Tester si	$a = b$	$a \neq b$	$a < b$	$a > b$	$a \leqslant b$	$a \geqslant b$
syntaxe Python	`a==b`	`a !=b`	`a<b`	`a>b`	`a<=b`	`a>=b`

V. — Boucles

Une boucle est la répétition d'un certain nombre d'actions identiques sur des variables (qui, elles, peuvent changer de valeurs) formant un bloc d'instructions.

1) Boucles bornées (boucle « Pour »)

Une boule bornée (ou boucle « Pour ») est une boucle dans laquelle les actions identiques sont répétées un nombre de fois déterminé à l'avance.

Pour répéter 10 fois un bloc d'instructions, la syntaxe est la suivante.

en langage naturel	en Python

Pour k variant de 0 à 9
  bloc d'instructions
Fin Pour
	`for k in range(10):`
	  bloc d'instructions

Dans cette syntaxe, `k` est une variable dont le nom peut être choisi arbitrairement.

Remarque ESSENTIELLE. Comme pour l'instruction conditionnelle, les deux points à la fin de la première ligne et l'indentation sont indispensables.

Il existe différentes syntaxes pour `for k in range(...):`

- `for k in range(n):` : `k` est une variable de type `int` qui prend successivement les valeurs 0, 1, ..., $n-1$.
- `for k in range(a,b):` : `k` est une variable de type `int` qui prend successivement les valeurs a, $a+1$, ..., $b-1$.
- `for k in range('chaîne_1', 'chaîne_2', ..., 'chaîne_n'):` : `k` est une variable de type `str` qui prend successivement les valeurs `'chaîne_1'`, `'chaîne_2'`, ..., `'chaîne_n'`.

Voici un exemple. Que fait cet algorithme/ce programme ?

en langage naturel	en Python

S $\leftarrow$ 0
Pour k allant de 0 à 10
  S $\leftarrow S + 2 \times k$
Fin Pour
	`S=0`
	`For k in range(11):`
	  `S=S+2*k`

2) Boucles non bornées (boucle « Tant que »)

Une boule non bornée (ou boucle « Tant que ») est une boucle dans laquelle un bloc d'instructions est répété tant qu'une certain condition C est vérifiée. La boucle s'arrête dès que la condition C n'est plus remplie.

La syntaxe est la suivante.

en langage naturel	en Python

Tant que (condition C)
  bloc d'instructions
Fin Tant que
	`while` (condition C):
	  bloc d'instructions

Remarque ESSENTIELLE. Comme pour une boucle bornée, les deux points à la fin de la première ligne et l'indentation sont indispensables.

Voici un exemple. Soit un réel $x \geqslant 1$. Que fait cet algorithme/ce programme ?

en langage naturel	en Python

A $\leftarrow$ 1
Tant que A $< x$
  A $\leftarrow 2 \times$ A
Fin Tant que
	`A=1`
	`while (A<x):`
	  `A=2*A`

VI. — Fonctions

Une fonction est une sorte sous-programme d'un programme Python dédié à réaliser une tâche particulière et auquel on peut faire appel dans le programme principal.

L'intérêt d'une fonction est d'éviter de réécrire plusieurs fois le même code d'instructions à différents endroits d'un programme.

La syntaxe est la suivante.

`def` nom_de_la_fonction(liste de paramètres):   bloc d'instructions

Si on veut que la fonction renvoie un résultat, on utilise la fonction `return`. L'exécution d'une fonction s'arrête dès le premier `return` rencontré. Il ne faut pas confondre la fonction `print` qui affiche un résultat à l'écran et la fonction `return` qui renvoie un résultat utilisable par la suite mais sans l'afficher.

Remarque ESSENTIELLE. Comme pour l'instruction conditionnelle et les boucles, les deux points à la fin de la première ligne et l'indentation sont indispensables.

Voici un exemple. Que fait cette fonction ?

```
def lpg(a,b):
  if (a>=b):
    return(a)
  else:
    return(b)
```

Écrire un programme en Python utilisant la fonction `lpg` qui demande 3 entiers à l'utilisateur et affiche en sortie le plus grand des 3.

VII. — Les modules

Il existe en Python de nombreuses fonctions prédéfinies qui ne sont pas automatiquement chargés mais qui sont stockés dans des fichiers appelés modules. Pour utiliser de telles fonctions, il faut au préalable importer le module qui les comporte.

Pour charger un module, on utilise la syntaxe suivante :

```
from nom_du_module import *
```

Il existe de très nombreux modules. Nous allons en évoquer seulement deux.

1) Le module `math`

Le module `math` contient toutes les constantes et les fonctions (au sens mathématiques) dont nous aurons besoin. On le charge à l'aide de la syntaxe suivante :

```
from math import *
```

Il contient en particulier

- `pi` : renvoie une valeur approchée de π à 10^{-16} près ;
- `sqrt(x)` : renvoie la racine carrée du nombre x ;
- `abs(x)` : renvoie la valeur absolue du nombre x ;

et, pour les élèves de 1ère,

- `cos(x)` : renvoie le cosinus du nombre x (en radians) ;
- `sin(x)` : renvoie le sinus du nombre x (en radians) ;
- `e` : renvoie une valeur approchée du nombre e à 10^{-16} près ;
- `exp(x)` : renvoie l'image de x par la fonction exponentielle (i.e. le nombre e^x).

2) Le module `random`

Le module **random** est un module qui permet d'engendrer des nombres aléatoires. On le charge à l'aide de la syntaxe suivante :

```
from math import *
```

Il contient en particulier les fonctions :

- `random()` : renvoie un nombre aléatoire entre 0 et 1 ;
- `randint(a,b)` : renvoie un entier aléatoire compris (au sans large) entre les deux entiers a et b.

VIII. — Les listes

1) Définition

Une liste est une suite ordonnée d'éléments. Une liste est représentée par des crochets et les éléments sont séparés par des virgules.

Ainsi, `L=[1,2,3,4]` affecte la liste formée des nombres 1, 2, 3 et 4 à la variable L. La variable ainsi définie est de type `list` (nouveau type qui se rajoute donc aux types déjà connus : `int`, `float` et `str`).

Remarque. Les éléments d'une liste peuvent être de types différents. Ainsi, Python accepte les listes comme `L=['a', 2, 0.24, 'abc', -2, '-2']` qui mêlent des entiers, des flottants et des chaînes de caractères.

Les éléments d'une liste sont numérotés à partir de 0. Ainsi, dans l'exemple suivant 1 est l'élément 0, 2 est l'élément 1 et ainsi de suite.

Pour obtenir la valeur de l'élément i d'une liste L, on saisit `L[i]`. Ainsi, pour la liste `L=[1,2,3,4]`, on a `L[0]=1`, `L[1]=2`, `L[2]=3` et `L[3]=4`.

Remarque. Le dernier terme d'une liste `L` peut être obtenue à l'aide de `L[-1]`.

ATTENTION ! Lorsqu'on écrit `L[i]`, il faut être sûr que la liste `L` contient au moins `i+1` éléments sinon on obtient un message d'erreur.

Le nombre d'éléments d'une liste est appelé la longueur de la liste. La fonction `len` permet d'obtenir la longueur (*length* en anglais) d'une liste. Ainsi, si `L=[2,4,7,2]`, alors `len(L)` renvoie 4.

Remarque. La liste vide qui ne contient aucun élément s'écrit `[]`. Ainsi, `len([])` renvoie `0`.

Pour définir une liste, il y a, en Python, plusieurs moyens :

- par énumération, comme nous l'avons précédemment, en écrivant explicitement les éléments de la liste : `L=[2,4,'a',2.4,3]` ;
- en compréhension, c'est-à-dire à l'aide d'une instruction conditionnelle (`if`) ou d'une boucle (`for` ou `while`) permettant de décrire tous les éléments de la liste dans l'ordre. Par exemple, la liste des entiers pairs de 0 à 10 peut s'obtenir de la manière suivante : `L=[2*k for k in range(4)]`.
- par répétition si on souhaite créer une liste dont tous les éléments sont identiques. Par exemple, $[1] * 10$ crée une liste de longueur 10 dont tous les éléments sont égaux à 1.

2) Opérations sur les listes

On dispose en Python d'un certain nombre de fonctions permettant d'agir les listes. En voici une liste non exhaustive.

- `append` permet d'ajouter une valeur à une liste. Par exemple, si `L=[2,4,6]` alors `L.append(8)` ajoute la valeur 8 à la liste `L` et ainsi la nouvelle valeur de `L` est `[2,4,6,8]`. Le nouvel élément est ajouté en fin de liste.
- `insert` permet d'ajouter une valeur dans une liste à une place donnée. Par exemple, si `L=[2,4,6]` alors `L.insert(2,8)` ajoute la valeur 8 à la liste `L` de telle sorte que 8 devienne l'élément numéro 2 de `L`. Ainsi, la nouvelle valeur de `L` est `[2,4,8,6]`.
- `remove` permet de supprimer une valeur d'une liste. Par exemple, si `L=[2,4,6]` alors `L.remove(4)` supprime la valeur 4 de la liste `L` et ainsi la nouvelle valeur de `L` est `[2,6]`. Si la valeur apparaît plusieurs fois dans la liste, seule la première occurrence est supprimée. Ainsi, si `L=[1,2,3,2,4]` alors `L.remove(2)` supprime le premier 2 de la liste et ainsi la nouvelle valeur de `L` est `[1,3,2,4]`. Il faut par ailleurs être certain que la valeur à supprimer fait bien partie de la liste au risque d'obtenir une erreur.
- `min`, `max` et `sum` permettent d'obtenir respectivement le plus petit, le plus grand et la somme des éléments d'une liste d'entiers ou de flottant. Ainsi, si `L=[1,5,4,2]` alors `min(L)` renvoie `1`, `max(L)` renvoie `5` et `sum(L)` renvoie `12`.

- `sort` permet de modifier une liste d'entiers ou de flottant en ordonnant ses éléments par ordre croissant. Ainsi, si `L=[3,4,7,2]` alors `L.sort()` transforme `L` en `[2,3,4,7]`.
- `sorted` permet d'otenir une liste ordonnée sans modifier la liste initiale. Ainsi, si `L=[3,4,7,2]` alors `sorted(L)` affiche `[2,3,4,7]` mais ne modifie pas la liste `L` (contrairement à `sort`).
- `+` permet de concaténer deux listes. Ainsi, si `L1=[3,4,7,2]` et `L2=[4,7,1]` alors `L1+L2` renvoie `[3,4,7,2,4,7,1]`.

3) Itération sur les éléments d'une liste

Il est possible de répéter une action sur les éléments d'une liste à l'aide d'une boucle `for`.

Par exemple, si $L = [-2, 3, 5, 2, -1, -3]$ qu'on veut afficher les carrés des éléments de L, on peut

```
for k in L:
  print(k**2)
```

4) Liste de listes

On l'a dit, on peut faire les listes avec des objets de différents types. Il est également possibles de faire des listes de listes c'est-à-dire des listes donc les éléments sont eux-mêmes des listes.

Par exemple, `L=[[1,2],['a', 'b', 'c'],[0.1,1.1,3.4]]` est une liste de longueur 3 dont les éléments sont les listes `[1,2]`, `['a','b','c']` et `[0.1,1.1,3.4]`. Ainsi, l'instruction `L[2]` renvoie `[0.1,1.1,3.4]` (puisque les listes sont numérotées à partir de 0). Si on veut accéder à l'élément d'une de ces trois listes, il faut utiliser une syntaxe du type `L[i][j]` qui permet d'obtenir l'élément numéro j de la liste numéro i. Ainsi, dans l'exemple précédent, pour obtenir 2, il faut saisir `L[0][1]` car 2 est le deuxième élément (donc numéro 1) de la première liste (donc numéro 0).

Les listes de listes permettent notamment de représenter les tableaux en Python.

Ainsi, si on veut représenter en Python le tableau suivant :

1	-1	0	3
6	3	-15	2
-1	-1	8	2

on peut créer la liste suivante :

`L=[[1,-1,0,3],[6,3,-15,2],[-1,-1,8,2]]`.

L'élément -15 qui se trouve à l'intersection de la deuxième ligne et de la troisième colonne du tableau s'obtiendra alors par `L[1][2]` (liste numéro 1, élement numero 2 de cette liste, tout étant numéroté à partir de 0).

IX. — Exercices

1) Variables et opérations

Exercice 1. Pour chaque exemple suivant, on écrit dans un éditeur Python uniquement la ligne indiquée. Déterminer lesquelles sont correctes et, pour celles-ci, l'affichage obtenu.

1) `print(4+3)`	2) `print(4+3.)`	3) `print(4+a)`	4) `print('4'+'a')`
5) `print(4+'a')`	6) `print(2-3)`	7) `print(2.-3)`	8) `print('2-a')`
9) `print(4*3)`	10) `print(4*3.)`	11) `print(4*a)`	12) `print(4*'a')`
13) `print(2**3)`	14) `print('a'**3)`	15) `print('ab'+'bc')`	16) `print('ab'*'bc')`

Exercice 2. Pour chacun des algorithmes suivants, déterminer la valeur des différentes variables à la fin de l'exécution. Traduire ensuite l'algorithme en un programme Python, ajouter l'affichage des variables et vérifier les réponses à la première question.

1)

$a \leftarrow 2$
$b \leftarrow 3$
$a \leftarrow a + 1$
$b \leftarrow a + b$

2)

$a \leftarrow 2$
$b \leftarrow 3$
$a \leftarrow a \times b$
$b \leftarrow a^2$

3)

$a \leftarrow 0$
$b \leftarrow 1$
$a \leftarrow b$
$b \leftarrow a$

4)

$a \leftarrow 1$
$b \leftarrow 2$
$a \leftarrow a + b$
$a \leftarrow a \times b$
$a \leftarrow a^b$

Exercice 3. Pour chacun des programmes suivants, déterminer la valeur des différentes variables à la fin de l'exécution. Implémenter les programmes et vérifier les réponses à la question précédentes en ajoutant un affichage.

1)
```
a='2'
b=3
a=a+'1'
b=a+'b'
```

2)
```
A='a'
B='b'
A=A*2
B=A+B
```

3)
```
a=0
b=1
a=a+b
b='a'+'b'
```

4)
```
a='b'
b='a'
a=a*2
b=a+b
```

Exercice 4. Chacun des programmes suivants contient une erreur. Identifier cette erreur et proposer une correction.

1)
```
A=a
B=4
A=A*B
B=2*B
```

2)
```
A=2
B='x'
'A'=A
B=A+B
```

3)
```
a=2
b='3'
a=a*b
b=a**b
```

4)
```
A='a'
B='b'
A=A*2
B=A*B
```

Exercice 5. Modifier le programme 2) de l'exercice 4 de telle sorte qu'à la fin de l'exécution la valeur de `B` soit la chaîne de caractères `2x`.

Exercice 6. Modifier l'algorithme 3) de l'exercice 2 de telle sorte qu'à la fin de l'exécution les valeurs initiales de `a` et `b` soient échangées.

Remarque. Python dispose en fait de la syntaxe particulière `a,b=b,a` qui permet d'échanger directement les valeurs des variables `a` et `b`.

Exercice 7. Écrire un programme en Python qui demande à l'utilisateur un caractère c et un entier n et qui affiche la chaîne de caractère composée du caractère c répété n fois.

Exercice 8. Écrire un programme en Python qui demande à l'utilisateur deux caractères c et d et deux entiers n et m et qui affiche la chaîne de caractères composée du caractère c répété n fois suivie du caractère d répété m fois.

2) Instructions conditionnelles

Exercice 9. Écrire un algorithme en langage naturel puis un programme en Python qui demande à l'utilisateur de saisir deux entiers et qui affiche le plus petit des deux.

Exercice 10. Écrire un programme en Python qui demande à l'utilisateur de saisir un entier et qui affiche `positif` si l'entier est positif ou nul et `négatif` si l'entier est strictement négatif.

Exercice 11. Écrire un programme en Python qui demande à l'utilisateur de saisir un réel et qui affiche `entier` si le réel est un entier et `non entier` sinon. Ainsi, le programme doit afficher `entier` si on saisit `3` ou `3.0` et `non entier` si on saisit `4.23`.

Exercice 12. Écrire un programme en Python qui demande à l'utilisateur de saisir un entier et qui affiche `pair` si l'entier est pair et `impair` sinon.

Indication. On pourra utiliser l'opérateur `%`.

Exercice 13. Dans une instruction conditionnelle, les conditions peuvent être composées de plusieurs sous-conditions comme « Si ($x > 0$ et $x < 3$) » ou « Si ($x > 3$ ou $x < -3$) ». Dans ce cas, on utilise les opérateurs logique `and` et `or`. Ainsi, les conditions précédentes s'écrivent respectivement en Python `if (x>0 and x<3):` et `if (x>3 or x<-3):`.

1. Écrire un programme en Python qui demande à l'utilisateur de saisir un réel x et qui affiche `vrai` si x appartient à l'intervalle $]-2\,;3]$ et `faux` sinon.
2. Écrire un programme en Python qui demande à l'utilisateur de saisir un réel x et qui affiche `vrai` si x appartient à l'ensemble $]-\infty\,;-3] \cup [5\,;+\infty[$ et `faux` sinon.
3. Écrire un programme en Python qui demande à l'utilisateur de saisir un réel x et qui affiche `vrai` si x appartient à l'ensemble $]-5\,;-3] \cup [0\,;2[$ et `faux` sinon.
4. Écrire un programme en Python qui demande à l'utilisateur de saisir un réel x et qui affiche `vrai` si x est strictement positif et différent de 1 ou si x est strictement négatif et différent -1 et `faux` sinon.

Exercice 14. Lorsqu'on a plus de deux cas possibles dans une instruction conditionnelle, on utilise la syntaxe suivante :

```
if (condition 1):
   bloc d'instructions 1
elif (condition 2):
   bloc d'instructions 2
else (condition 3):
   bloc d'instructions 3
```

On peut ajouter autant de conditions intermédiaires que l'on veut en ajoutant `elif` devant chacun d'elles.

Écrire un programme en Python qui demande à l'utilisateur de saisir un entier et qui affiche `positif` si l'entier est strictement positif, `nul` si l'entier est nul et `négatif` si l'entier est strictement négatif.

Exercice 15. Une année est bissextile si elle est divisible par 4 mais pas par 100 ou bien si elle est divisible par 400. Par exemple, 2020 est bissextile car 2020 est divisible par 4 mais pas par 100. De même, 2000 est bissextile car 2000 est divisible par 400. En revanche, 2100 n'est pas bissextile car 2100 est divisible par 100 mais pas par 400.

Écrire un programme en Python qui demande à l'utilisateur de saisir une année et qui affiche `bissextile` si l'année est bissextile et `non bissextile` sinon.

3) Boucles bornées

Exercice 16. On considère l'algorithme suivant.

```
Pour k allant de 0 à 5
    Afficher 2k
Fin Pour
```

1. Quelle est la fonction de cet algorithme ?
2. Traduire cet algorithme en un programme Python puis implémenter ce programme.

Exercice 17. On considère l'algorithme suivant.

```
S ← 0
Pour i allant de 1 à 10
    S ← S + i²
Fin Pour
```

1. Quelle est la fonction de cet algorithme ?
2. Traduire cet algorithme en un programme Python puis implémenter ce programme.

Exercice 18. Le programme suivant est censé calculer la somme des puissances de 3 de 3^4 jusqu'à 3^{10}.

```
S=0
for j in range(4,10)
  S=S+3^j
```

1. Il y a plusieurs erreurs dans ce programme. Les trouver et les corriger.
2. Implémenter ce programme en rajoutant une ligne pour afficher le résultat.

Exercice 19. Écrire une programme Python qui demande à l'utilisateur la saisie d'un entier n et qui calcule puis affiche la somme des entiers de 1 à n.

Exercice 20.

1. Écrire un programme Python qui demande à l'utilisateur la saisie d'une chaîne de caractères `s` et d'un entier n et qui affiche n fois la chaîne s sur n lignes successives.
2. Modifier le programme précédent de telle sorte que les lignes soient de plus numérotées de 1 à n (la première sera donc de la forme : `1. s`, la seconde de la forme : `2. s` et ainsi de suite, le numéro de la ligne étant suivi d'un point et le point suivi d'un espace puis de la chaîne `s`).

Exercice 21. Écrire une programme Python qui demande à l'utilisateur la saisie d'un caractère `c` et d'un entier n et qui affiche n lignes successives de telle sorte que la première ligne contient `c`, la seconde `cc`, la troisième `ccc` et, de manière générale, la $i-$ème ligne contient la chaîne formée du caractère `c` répété i fois.

Exercice 22.

1. Écrire une programme Python qui affiche le nombre d'entiers compris entre 1 et 100 qui sont divisibles par 7.
2. Modifier le programme précédent pour qu'il affiche seulement le nombre d'entiers compris entre 1 et 100 qui sont divisibles par 7 mais ni 3 ni par 5.

Exercice 23. On dit qu'un entier naturel n est parfait si la somme de ses diviseurs positifs est égale $2n$. Par exemple, 6 est parfait car les diviseurs positifs de 6 sont 1, 2, 3 et 6 et $1 + 2 + 3 + 6 = 12 = 2 \times 6$.

Écrire un programme Python qui calcule le nombre d'entiers parfaits compris entre 1 et 1000.

4) Boucles non bornées

Exercice 24. On considère l'algorithme suivant.

```
A ← 1
Tant que A < 10
   Afficher A
   A = A + 2
Fin Tant que
```

1. Quelle est la fonction de cet algorithme ?

2. Traduire cet algorithme en un programme Python puis implémenter ce programme.

Remarque. En Python, l'affectation `A=A+2` peut aussi s'écrire `A+=2`.

Exercice 25. On considère l'algorithme suivant.

```
k ← 0
Tant que k² < 1000
   k ← k + 1
Fin Tant que
```

1. Quelle est la fonction de cet algorithme ?

2. Traduire cet algorithme en un programme Python puis implémenter ce programme.

Exercice 26. Le programme suivant est censé calculer la somme des nombres impairs inférieurs à 50.

```
k=1
S=0
while (S<50)
  k+=2
  S=S+k
```

1. Il y a plusieurs erreurs dans ce programme. Les trouver et les corriger.

2. Implémenter ce programme en rajoutant une ligne pour afficher le résultat.

Exercice 27. Sur un compte en banque, on place 1000 €. Le compte rapporte 2% d'intérêt par an. Ainsi, au bout d'un an, il y aura sur le compte $1000 + \frac{2}{100} \times 1000 = 1020$ €.

Écrire un algorithme en langage naturel puis un programme en Python qui calcule le nombre d'années nécessaires pour que le montant sur le compte dépasse 1500 €. On suppose que le taux reste à 2% et qu'on n'ajoute rien d'autre sur le compte que les intérêts annuels.

Exercice 28. Écrire un programme en Python ayant le même fonctionnement que celui de l'exercice 19 mais en utilisant une boucle non bornée.

Exercice 29. Écrire un algorithme en langage naturel puis un programme en Python qui calcule le plus grand entier inférieur à 1000 qui est le carré d'un entier.

Exercice 30. Écrire un programme en Python qui calcule le plus petit entier n tel que la somme des carrés des entiers de 1 à n soit supérieure ou égale à 1000.

Exercice 31. Écrire un programme en Python qui calcule le nombre d'entiers naturels non nuls n tels que n^n est inférieur ou égale à 1000000.

Exercice 32. Reprendre l'exercice 21 en n'utilisant que des boucles non bornées.

5) Fonctions

Exercice 33. On considère le programme suivant.

```
def somme(a,b):
  c=a+b

print(somme(3,5))
```

1. Implémenter ce programme et l'exécuter. Expliquer l'affichage obtenu.
2. Modifier la fonction `somme` de telle sorte que l'exécution du programme modifié affiche 8.

Exercice 34. En s'inspirant de l'exercice précédent, écrire

1. une fonction `difference` telle que `difference(a,b)` renvoie `a-b` ;
2. une fonction `produit` telle que `produit(a,b)` renvoie `a*b` ;

Exercice 35. Écrire une fonction `min` telle que `min`(a, b) renvoie le minimum de `a` et `b`.

Exercice 36.

1. Écrire une fonction `est_isocele` qui prend en argument trois nombres entiers a, b et c est qui renvoie `isocèle` si le triangle de côtés a, b et c est isocèle et `non isocèle` sinon.
2. Écrire de même une fonction `est_rectangle` qui prend en argument trois nombres entiers a, b et c est qui renvoie `rectangle` si le triangle de côtés a, b et c est rectangle et `non rectangle` sinon.

Exercice 37.

1. Écrire une fonction `som_div` telle que `som_div(n)` renvoie la somme des diviseurs positifs de l'entier `n`.
2. Utiliser la fonction `som_div` pour réécrire le programme de l'exercice 23.

Exercice 38. Si n est un entier au moins égal à 1, on définit la nombre factorielle n, noté $n!$ par $n! = 1 \times 2 \times 3 \times \cdots \times n$. Ainsi, $1! = 1$, $2! = 1 \times 2$, $3! = 1 \times 2 \times 3 = 6$, $4! = 1 \times 2 \times 3 \times 4 = 24$.

Écrire une fonction `factorielle` telle que `factorielle(n)` renvoie `n!`.

Exercice 39. La suite de Syracuse est une suite de nombres entiers définie de la manière suivante :

- on part d'un nombre entier n ;
- si n est pair, le nombre suivant vaut $\frac{n}{2}$;
- sinon, le nombre suivant est $3n + 1$.
- on recommence avec le nouveau nombre ainsi calculé.

Par exemple, en partant du nombre 7, on obtient la suite :

$$7, \quad 7 \times 3 + 1 = 22, \quad \frac{22}{2} = 11, \quad 11 \times 3 + 1 = 34, \quad \frac{34}{2} = 17, ...$$

1. Écrire une fonction `syracuse` telle que `syracuse(n)` renvoie le nombre suivant le nombre `n` dans la suite de Syracuse.
2. Une conjecture (non encore démontrée à ce jour) postule que cette suite finit toujours par atteindre le nombre 1.

   En utilisant la fonction `syracuse`, écrire un programme qui demande la saisie d'un entier n à l'utilisateur et qui affiche tous les termes de la suite de Syracuse partant de n jusqu'à arriver pour la première fois à 1.

6) Les modules

Exercice 40.

1. En utilisant le théorème de Pythagore, écrire une fonction `hypotenuse(a,b)` qui renvoie la longueur de l'hypoténuse d'un triangle rectangle dont les deux côtés adjacents à l'angle droit mesurent `a` et `b`.
2. Utiliser cette fonction pour écrire un programme qui demande `a` et `b` et affiche la longueur de l'hypoténuse.

Exercice 41. En utilisant la fonction `sqrt`, écrire une fonction `carreParfait(n)` qui affiche *vrai* si l'entier `n` est le carré d'un entier et *faux* sinon.

Exercice 42. Écrire une fonction `pileOuFace` qui affiche au hasard et de manière équiprobable *pile* ou *face*.

Exercice 43. Écrire une fonction `lancerDeDe` qui renvoie au hasard et de manière équiprobable un nombre entier en 1 et 6.

Exercice 44. Lorsqu'on jour au Monopoly, on lance deux dés au hasard et on calcule la somme des valeurs obtenues.

En utilisant la fonction `lancerDeDe` de l'exercice 43, écrire une fonction `tirageMonopoly()` qui renvoie le résultat d'un tel tirage.

Exercice 45. écrire un programme qui demande un entier `n` à l'utilisateur, qui simule `n` lancers successifs de dé et qui affiche la fréquence de 6 obtenus (c'est-à-dire le nombre de 6 obtenus divisé par le nombre total de lancers).

Exercice 46. En utilisant la fonction `lancerDeDe` de l'exercice 43, écrire un programme qui simule une répétition de lancer de dé et affiche le nombre de lancers nécessaires pour obtenir un 6 pour la première fois.

Exercice 47. On veut réaliser un programme d'entraînement au calcul mental, et plus précisément à la connaissance des tables de multiplication.

On veut que le programme réalise les opérations suivantes :

- tirer deux nombres au hasard `a` et `b` entre 2 et 10 inclus ;
- demander à l'utilisateur le résultat du produit de `a` par `b` ;
- afficher le résultat du produit de `a` par `b` et si le résultat donné par l'utilisateur est juste ou non.

Par exemple :

```
6*3 = 18. Votre réponse est juste.
```

1. Implémenter le programme en Python.
2. Modifier ce programme pour qu'il propose 10 multiplications consécutives et qu'il affiche, en plus de ce qui précède, le nombre total de bonnes réponses à la fin.

Exercice 48. Le jeu du « plus grand, plus petit » se déroule de la manière suivante :

- le programme choisit au hasard un nombre `a` entre 1 et 1000 ;
- le joueur propose un nombre `b` compris entre 1 et 1000 ;
- le programme indique au joueur si `b` est plus grand ou plus petit que `a` ;
- on continue ainsi jusqu'à ce que le joueur trouve le nombre `a`.

1. Implémenter ce jeu en Python.
2. Modifier le programme pour qu'il joue tout seul à ce jeu.

7) Les listes

Exercice 49. On considère la liste suivante :

```
L=[1,2.3,4.1,4,2.55,5,23]
```

Que renvoie chacune des instructions suivantes :

a) `L[1]`?	**b)** `L[5]`?	**c)** `L[7]`?	**d)** `len(L)`?
e) `max(L)`?	**f)** `sum(L)`?	**g)** `sorted(L)`	**h)** `L+L`?

Exercice 50. Pour chacune des listes suivantes, donner une syntaxe Python permettant de la définir en compréhension.

1. `[1,3,5,7,9,11,13,15,17,19]`
2. `[0,1,4,9,25,36,49,64,81,100]`
3. `[1,2,4,8,16,32,64,128,256,512,1024]`
4. `[3,6,12,15,21,24,30,33,39,42,48,51]`

Exercice 51. Écrire en Python une fonction `suite` qui prend en argument un entier naturel n et qui renvoie la liste des n premières valeurs de la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = u_n^2 + n$.

Exercice 52. Écrire en Python une fonction `produit` qui prend en argument une liste d'entiers ou de flottant et qui renvoie le produit des éléments de la liste.

Exercice 53. Écrire en Python une fonction `ajouter_et_trier` qui prend en argument une liste d'entiers `L` et un entier `a`, qui ajoute à `a` à `L` puis ordonne la nouvelle valeur de `L` dans l'ordre croissant.

Exercice 54. Écrire en Python une fonction `valeur_absolue` qui prend en argument une liste d'entiers `L` et remplace chaque élément de la liste par sa valeur absolue.

Exercice 55. Expliquer le rôle de la fonction `nb6` suivante :

```
from random import *

def nb6(n):
    c=0
    Liste=[randint(1,6) for k in range(n)]
    for j in Liste:
        if j==2:
            c+=1
    return c
```

Exercice 56. Écrire en Python une fonction `E_et_sigma` qui demande à l'utilisateur de rentrer les valeurs d'une variable aléatoire X puis les probabilités associées (dans le même ordre, évidemment) et qui renvoie l'espérance et l'écart-type de X.

On donnera deux solutions : une première en utilisant deux listes (une pour les valeurs de X et l'autre pour les probabilités) et une seconde en utilisant une liste de listes.