ALGORITHMIQUE ET PROGRAMMATION : EXERCICES D'INTRODUCTION À PYTHON
CORRECTION

Exercice 1

1.

```
a=2
b=3
print(a+b)
```

← Comme on peut s'y attendre, le programme affiche « 5 ».

```
a,b = 2,3
print(a+b)
```

← C'est le même programme que le premier, mais la première ligne permet de donner des valeurs à `a` et `b` en une seule ligne. Affichage : « 5 ».

```
a,b = 2,3
a=b
b=a
print(a,b)
```

← En lisant « `a=b` » puis « `b=a` », on pourrait s'attendre à ce que le programme échange les valeurs de `a` et `b`, en affichant « 3,2 ». Or l'affichage est « 3,3 ».
En effet, `a` vaut 2 et `b` vaut 3. En écrivant « `a=b` », `a` prend la valeur 3.
Puis en écrivant « `b=a` », b prend la valeur contenue dans `a`, c'est-à-dire 3.
Finalement, `a` vaut 3 et `b` vaut 3.

```
a,b = 2,3
a,b = b,a
print(a,b)
```

← C'est le même programme que le précédent, sauf que les deuxième et troisième lignes sont codées en une seule : « `a,b = b,a` ». En Python, cela permet d'échanger les valeurs de `a` et `b` de façon instantanée.
Nous n'avons donc pas le problème précédent, l'affichage est « 3,2 ».

2.

```
nombre=17
print(nombre)
```

← Affichage attendu et observé : « 17 ».

```
nombre=17
print("nombre")
```

← Affichage : « nombre ». Python distingue le texte « nombre » et la variable nommée `nombre`.

```
nombre=17
print("nombre=",nombre)
```

← Affichage : « nombre= 17 ». Même commentaire que ci-dessus.

```
nombre=17
print(nombre+1)
```

← Affichage : « 18 ». Même commentaire que ci-dessus.

```
nombre="17"
print(nombre+1)
```

← Affichage d'un message d'erreur. En effet, la variable `nombre` contient le texte « 17 ». On ne peut donc pas demander d'afficher le résultat de `nombre+1`, puisque cela revient à ajouter du texte et l'entier 1.

```
nombre="17"
print(nombre+"1")
```

← Affichage : « 171 ». En effet, la variable `nombre` contient le texte « 17 ». Python comprend `nombre+"1"` comme la concaténation du texte « 17 » et du texte « 1 ».

Exercice 2

```
from math import sqrt
print(sqrt(3)**2)
print(sqrt(22)**2)
```

← La première ligne permet d'importer une fonction Python, nommée `sqrt`, qui permet de calculer une racine carrée (*sqrt* signifie ***sq**uare **root***).

La deuxième ligne calcule une valeur approchée de $\sqrt{3}^2$, ce qui devrait afficher 3 : pourtant cela affiche « 2.9999999999999996 ».
La deuxième ligne calcule une valeur approchée de $\sqrt{22}^2$, ce qui devrait afficher 22 : cela affiche « 22.0 ».
Cela vient du fait que, lorsque nous écrivons `sqrt(3)`, Python prend une valeur approchée de $\sqrt{3}$.
Idem pour `sqrt(22)`. En mettant ces valeurs approchées au carré, il obtient une valeur *très* approchée…

Conclusion : ne jamais oublier que Python permet la plupart du temps de faire des calculs approchés.

Remarque : en réalité, le problème est lié à la façon dont Python code les nombres. Quand j'écris « 0.3 », Python code ce nombre décimal en nombre binaire (on appelle cela des *nombres flottants* ou *float numbers* en anglais). Or, certains nombres décimaux ont une écriture illimitée quand on les convertit en binaire. C'est le cas par exemple du nombre décimal 0.1. Par conséquent, lorsqu'on demande à Python de calculer `0.1+0.2`, celui-ci n'affiche pas `0.3` mais `0.30000000000000004`.

Exercice 3

```
from math import pi
lune=1737
print(4/3*pi*lune**3)
```

← La première ligne permet d'importer une variable `pi` qui contient une valeur approchée du nombre π.
Ce programme affiche une valeur approchée du volume de la lune (en km^3), dont le rayon est d'environ 1737 km.

Exercice 4

```
a=2840
b=112
r=a%b
q=a//b
print(a,"=",b,"x",q,"+",r)
```

← Affichage : « 2840 = 112 x 25 + 40 ».
La troisième ligne calcule le reste de `a` par `b`.
La quatrième ligne calcule le quotient de `a` par `b`.
La dernière ligne alterne l'affichage de variables et de textes.
Ce programme affiche donc la division euclidienne de `a` par `b`.

Exercice 5

Avant le programme, on rajoute si besoin "`from math import sqrt`".

Valeur rentrée →	snt	2	3+4	sqrt(2)	2/3
n°1 : `print(x)`	snt	2	3+4	sqrt(2)	2/3
n°2 : `print(x+2)`	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**
n°3 : `print(x+"texte")`	snttexte	2texte	3+4texte	sqrt(2)texte	2/3texte

Conclusion : "`input("Valeur proposée ?")`" permet d'entre une valeur, qui est toujours interprétée par Python comme une chaîne de caractères. Les calculs ne sont donc pas possibles si on ne convertit pas ce texte en entier ou nombre (flottant).

Exercice 6

Avant les programmes, on rajoute si besoin "`from math import sqrt`".

Programme n°1, avec **`int`**`(input(...))` :

Valeur rentrée	snt	2	3+4	sqrt(2)	2/3
`print(x+2)`	**ERREUR**	4	**ERREUR**	**ERREUR**	**ERREUR**
`print(x+"texte")`	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**

Programme n°2, avec **`float`**`(input(...))` :

Valeur rentrée	snt	2	3+4	sqrt(2)	2/3
`print(x+2)`	**ERREUR**	4.0	**ERREUR**	**ERREUR**	**ERREUR**
`print(x+"texte")`	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**

Programme n°3, avec **`eval`**`(input(...))` :

Valeur rentrée	snt	2	3+4	sqrt(2)	2/3
`print(x+2)`	**ERREUR**	4	9	3.414213562373095	2.6666666666666665
`print(x+"texte")`	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**	**ERREUR**

Conclusions : – **`int`** convertit une chaîne en nombre entier : int("54") renvoie l'entier 54
– **`float`** convertit une chaîne en nombre (flottant) : float("54.27") renvoie le nombre 54.27
– **`eval`** permet d'interpréter une chaîne de caractères comme un calcul à effectuer :
c'est très utile lorsqu'on veut rentrer des fractions, des opérations, des racines carrées…

Exercice 7

1.

```
from math import *
xA = eval(input("Entrer l'abscisse de A :"))
yA = eval(input("Entrer l'ordonnée de A :"))
xB = eval(input("Entrer l'abscisse de B :"))
yB = eval(input("Entrer l'ordonnée de B :"))
xI, yI = (xA+xB)/2, (yA+yB)/2
print(xI, yI)
```

permet de pouvoir rentrer des racines carrées (par exemple) en tapant `sqrt(…)`

calcule l'abscisse `xI` et l'ordonnée `yI` du milieu du segment [AB]

2.

```
from math import *
xA = eval(input("Entrer l'abscisse de A :"))
yA = eval(input("Entrer l'ordonnée de A :"))
xB = eval(input("Entrer l'abscisse de B :"))
yB = eval(input("Entrer l'ordonnée de B :"))
dist = (xB-xA)**2+(yB-yA)**2
print("La distance AB est égale à la racine carrée de",dist,"soit environ",sqrt(dist),".")
```

Exercice 8

Ce programme demande un nombre à l'utilisateur ; un calcul ou une racine carrée est possible puisque `eval` est devant le `input`. Si ce nombre est strictement supérieur à 3, le programme l'indique. Idem dans le cas contraire (nombre inférieur ou égal à 3).

Exercice 9

```
a = int(input("Donnez l'âge du joueur :"))
if a<8:
    print("trop jeune")
elif 8<=a<10:
    print("U8")
elif 10<=a<12:
    print("U10")
elif 12<=a<14:
    print("U12")
elif 14<=a<16:
    print("U14")
else:
    print("ce joueur ne rentre dans aucune catégorie !")
```

Exercice 10

```
for i in range(3):
    print(i)
```

← affiche les entiers 0, 1 et 2

attention

```
for i in range(8,23):
    print(i)
```

← affiche les entiers de 8 à **22** (inclus)

```
for i in range(8,23,4):
    print(i)
```

← affiche les entiers de 8, 12, 16 et 20
autrement dit les entiers de 8 à 22 avec un pas de 4

Exercice 11

```
for i in range(0,101):
    print(i,"²=",i**2)
```

```
93 ² = 8649
94 ² = 8836
95 ² = 9025
96 ² = 9216
97 ² = 9409
98 ² = 9604
99 ² = 9801
100 ² = 10000
>>>
```

Remarque : on remarque que l'affichage obtenu n'est pas vraiment parfait. En effet, il y a des espaces entre l'entier qui varie et le symbole ². Pour remédier à ce problème, on peut utiliser une façon plus efficace d'utiliser `print` :

```
for i in range(0,101):
    print("{}² = {}".format(i,i**2))
```

Exercice 12

```
for i in range(2,21):
    print("")
    for j in range(1,11):
        print(i,"x",j,"=",i*j)
```

Exercice 13

Ce programme affiche une valeur approchée du volume d'une boule de rayon `r`, pour `r` qui varie de 1 à 100. L'intérêt de procéder ainsi (utilisation d'une fonction nommée `sphere_vol`) est de pouvoir ré-utiliser facilement toute une partie de code que l'on souhaite appliquer à plusieurs nombres.