# Corrigés des exercices d'algorithmique et de programmation en Python

## 1) Variables et opérations

**Exercice 1.**

1) `print(4+3)` est correct et affiche l'entier `7`.
2) `print(4+3.)` est correct est affiche `7.0`
3) `print(4+a)` n'est pas correct car `a` est le nom d'une variable qui n'est pas définie.
4) `print('4'+'a')` est correct et affiche la chaîne de caractère `4a`
5) `print(4+'a')` n'est pas correct car `4` est de type `int` et `'a'` est de type `str`
6) `print(2-3)` est correct et affiche l'entier `-1`
7) `print(2.-3)` est correct et affiche le réel `-1.`
8) `print('2-a')` est correct et affiche la chaîne de caractères `2-a`
9) `print(4*3)` est correct et affiche l'entier `12`
10) `print(4*3.)` est correct et affiche le réel `12.0`
11) `print(4*a)` n'est pas correct car `a` est le nombre d'une variable qui n'est pas définie.
12) `print(4*'a')` est correct et affiche la chaîne de caractère `aaaa`
13) `print(2**3)` est correct et affiche l'entier `8`
14) `print('a'**3)` n'est pas correct car `'a'` est une chaîne de caractères.
15) `print('ab'+'bc')` est correct et affiche la chaîne de caractère `abbc`
16) `print('ab'*'bc')` n'est pas correct car `'ab'` et `'bc'` sont des chaînes de caractères.

**Exercice 2.** Pour chacun des algorithmes, on peut dresser une table d'exécution.

1)

| $a$ | $b$ |
|---|---|
| 2 | |
| 2 | 3 |
| 3 | 3 |
| 3 | 6 |

À la fin de l'exécution, $a$ vaut 3 et $b$ vaut 6.

2)

| $a$ | $b$ |
|---|---|
| 2 | |
| 2 | 3 |
| 6 | 3 |
| 6 | 36 |

À la fin de l'exécution, $a$ vaut 6 et $b$ vaut 36.

3)

| $a$ | $b$ |
|---|---|
| 0 | |
| 0 | 1 |
| 1 | 1 |
| 1 | 1 |

À la fin de l'exécution, $a$ et $b$ valent 1.

4)

| $a$ | $b$ |
|---|---|
| 1 | |
| 1 | 2 |
| 3 | 2 |
| 6 | 2 |
| 36 | 2 |

À la fin de l'exécution, $a$ vaut 36 et $b$ vaut 2.

**Exercice 3.** Dressons, là aussi, une table d'exécution.

1)

| a | b |
|---|---|
| '2' | |
| '2' | 3 |
| '21' | 3 |
| '21' | '21b' |

À la fin de l'exécution, `a` contient la chaîne de caractère `21` et `b` la chaîne de caractères `21b`.

2)

| A | B |
|---|---|
| 'a' | |
| 'a' | 'b' |
| 'aa' | 'b' |
| 'aa' | 'aab' |

À la fin de l'exécution, `A` contient la chaîne de caractère `aa` et `B` la chaîne de caractères `aab`.

3)

| a | b |
|---|---|
| 0 | |
| 0 | 1 |
| 1 | 1 |
| 1 | 'ab' |

À la fin de l'exécution, `a` contient l'entier `1` et `b` la chaîne de caractères `ab`.

4)

| a | b |
|---|---|
| 'b' | |
| 'b' | 'a' |
| 'bb' | 'a' |
| 'bb' | 'bba' |

À la fin de l'exécution, `a` contient la chaîne de caractère `bb` et `b` la chaîne de caractères `bba`.

**Exercice 4.**

1) Dans l'affectation `A=a`, la lettre `a` désigne une variable qui n'est pas définie. Une correction possible est :

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

2) L'affectation `'A'=A` n'est pas écrite dans le bon ordre. Une correction possible est :

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

3) L'opération `a**b` est incorrecte car la variable `a` est de type `str` (elle vaut `'33'`). Une modification possible est :

```
a=2
b=3
a=a*b
b=a**b
```

4) L'opération `A*b` est incorrect car `A` et `B` sont de type `str`. Une modification possible est

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

**Exercice 5.** On peut modifier le programme de la manière suivante :

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

**Exercice 6.** On peut modifier l'algorithme en introduisant une troisième variable $c$ pour stocker la valeur de $a$.

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

Ainsi, à la fin de l'exécution, $a$ vaut 1 et $b$ vaut 0.

**Exercice 7.**

```
c=input("Saisir un caractère :")
n=int(input("Saisir un entier positif :"))
print(c*n)
```

**Exercice 8.**

```
c=input("Saisir un caractère :")
d=input("Saisir un caractère :")
n=int(input("Saisir un entier positif :"))
m=int(input("Saisir un entier positif :"))
print(c*n+d*m)
```

## 2) Instructions conditionnelles

**Exercice 9.** En langage naturel,

Saisir un entier $a$
Saisir un entier $b$
Si $(a < b)$
    Afficher $a$
Sinon
    Afficher $b$

En Python,

```
a=int(input("Saisir un entier :"))
b=int(input("Saisir un entier :"))
if (a<b):
  print(a)
else:
  print(b)
```

**Exercice 10.**

```
a=int(input("Saisir un entier :"))
if (a>=0):
  print('positif')
else:
  print('négatif')
```

**Exercice 11.** L'idée est de tester si l'entier saisi `a` est égal ou non à `int(a)`. Si c'est le cas alors `a` est entier et non `a` n'est pas entier.

```
a=float(input("Saisir un réel :"))
if (a==int(a)):
  print('entier')
else:
  print('non entier')
```

**Exercice 12.** L'idée est d'utiliser l'opérateur %2 qui renvoie le reste dans la division par 2. Ainsi, si `a%2` vaut 0 alors `a` pair et sinon `a` est impair.

```
a=int(input("Saisir un entier :"))
if (a%2==0):
  print('pair')
else:
  print('impair')
```

**Exercice 13.**

1.

```
x=float(input("Saisir un réel :"))
if (x>-2 and x<=3):
  print('vrai')
else:
  print('faux')
```

2.

```
x=float(input("Saisir un réel :"))
if (x<=-3 or x>=5):
  print('vrai')
else:
  print('faux')
```

**3.**

```
x=float(input("Saisir un réel :"))
if ((x>-5 and x<=-3) or (x>=0 and x<2)):
  print('vrai')
else:
  print('faux')
```

**Remarque**. — Les parenthèses dans la condition

```
((x>-5 and x<=-3) or (x>=0 and x<2))
```

ne sont pas indispensables car en Python il y a priorité du `and` sur le `or` donc

```
x>-5 and x<=-3 or x>=0 and x<2
```

est automatique interprété comme `((x>-5 and x<=-3) or (x>=0 and x<2))`. Cependant, les parenthèses rendent le code plus lisible.

**4.**

```
x=float(input("Saisir un réel :"))
if ((x>0 and x!=1) or (x<0 and x!=-1)):
  print('vrai')
else:
  print('faux')
```

**Exercice 14.**

```
x=int(input("Saisir un entier :"))
if (x>0):
  print('positif')
elif (x==0):
  print('nul')
else:
  print('négatif')
```

**Exercice 15.** Pour tester la divisibilité par 4 (respectivement 100 et 400), on peut examiner le reste dans la division par 4 (respectivement par 100 et 400) à l'aide de l'opérateur %.

```
x=int(input("Saisir une année :"))
if ((x%4==0 and x%100!=0) or (x%400==0)):
  print('bissextile')
else:
  print('non bissextile')
```

## 3) Boucles bornées

**Exercice 16.**

1. L'algorithme proposé affiche les entiers pairs de 0 à 10.
2. Le programme Python correspondant est

```
for k in range(6):
  print(2*k)
```

**Exercice 17.**

1. L'algorithme proposé calcule la somme des carrés des entiers compris entre 1 et 10.
2. Le programme Python correspondant est

```
S=0
for i in range(1,11):
  S=S+i**2
```

**Exercice 18.**

1. Il y a deux erreurs de syntaxe : il manque les deux points à la fin de la deuxième ligne et il faut remplacer `^` par `**` dans la troisième ligne pour calculer la puissance. Il y a, de plus, une erreur dans le `range` : la variable `j` doit varier de 4 à 11 donc il faut écrire `range(4,11)`.
2. Le programme corrigé (et augmente d'un affichage) est donc

```
S=0
for j in range(4,11):
  S=S+3**j
print(S)
```

À l'affichage, on obtient 88533.

**Exercice 19.** Attention, pour aller de 1 à $n$, il faut écrire `range(1,n+1)`.

```
n=int(input('Saisir un entier :'))
S=0
for k in range(1,n+1):
  S=S+k
print(S)
```

**Exercice 20.**

1.

```
s=input('Saisir une chaîne de caractère :')
n=int(input('Saisir un entier :'))
for k in range(n):
  print(s)
```

2. Pour afficher le numéro de la ligne suivi d'un point, d'un espace et de la chaîne `s`, il faut transformer l'entier `k` en chaîne de caractère à l'aide de la fonction `str` puis concaténer (à l'aide de `+`) avec `'. '` et avec `s`.

```
s=input('Saisir une chaîne de caractère :')
n=int(input('Saisir un entier :'))
for k in range(1,n+1):
  print(str(k)+'. '+s)
```

**Exercice 21.** Pour répéter `i` fois un caractère `c`, on utiliser `c*i`.

```
c=input('Saisir un caractère :')
n=int(input('Saisir un entier :'))
for i in range(1,n+1):
  print(c*i)
```

**Exercice 22.**

1. Pour tester si un nombre $k$ est divisible par 7, on teste si `k%7` est nul ou non. On crée un variable `compteur` qui compte le nombre d'entiers divisible par 7. On ajoute `1` à `compteur` à chaque fois qu'on trouve un nombre divisible par 7.

   ```
   compteur=0
   for k in range(1,101):
     if (k%7==0):
       compteur=compteur+1
   print(compteur)
   ```

   À l'affichage, on obtient 14.

2. 

   ```
   compteur=0
   for k in range(1,101):
     if (k%7==0 and k%3!=0 and k%5!=0):
       compteur=compteur+1
   print(compteur)
   ```

   À l'affichage, on obtient 8.

**Exercice 23.** On teste, pour chaque entier entre 1 et 1000 s'il est parfait ou non. Pour cela, on crée une variable `compteur` initialisée à 0 et, pour tout entier $k$ entre 1 et 1000, on crée une variable `somme` initialisée à 0 et on regarde, pour tout entier $d$ entre 1 et $k$ si $d$ divise $k$. Si c'est le cas, on augmente la variable `somme` de $d$. Ainsi, on obtient dans la variable `somme` la somme des diviseurs positifs de $d$. Ensuite, on teste si `somme` est égale ou non à $2k$. Si c'est le cas, on augmente le compteur de 1.

```
compteur=0
for k in range(1,1001):
  somme=0
  for d in range(1,k+1):
    if (k%d==0):
      somme=somme+d
  if (somme==2*k):
    compteur=compteur+1
print(compteur)
```

À l'affichage, on obtient 3. (Il n'y a que 3 nombres parfaits inférieurs à 1000 : 6, 28 et 496 !)

## 4) Boucles non bornées

**Exercice 24.**

1. Cet algorithme affiche les entiers naturels impairs strictement inférieurs à 10.
2. Le programme Python correspondant est :

```
A=1
while (A<10):
  print(A)
  A=A+2
```

**Exercice 25.**

1. Cet algorithme détermine le plus petit entier naturel dont le carré dépasse 1000.
2. Le programme Python correspondant est

```
k=0
while (k**2<1000):
  k=k+1
```

**Exercice 26.**

1. Il y a un erreur de syntaxe : il manque les deux points à la fin de la ligne `while` et deux erreurs de programmation : la condition est `k<50` et non pas `S<50`. De plus, il faut échanger les deux lignes après le `while` car on modifie la valeur de `k` avant de l'ajouter à `S` alors qu'il faut faire l'inverse.
2. Le programme corrigé et complété est donc

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

**Exercice 27.** On introduit une variable $M$ qui calcule le montant sur compte, année après année, tant que ce montant ne dépasse pas 1500 et une variable « compteur » $C$ qui compte le nombre d'années écoulées.

En langage naturel :

$M \leftarrow 1000$
$C \leftarrow 0$
Tant que $(M \leqslant 1500)$
    $M \leftarrow M + \frac{2}{100}M$
    $C \leftarrow C + 1$
Fin Tant que

En Python :

```
M=1000
C=0
while (M<=1500):
  M=M+(2/100)*M
  C+=1
```

**Exercice 28.**

```
n=int(input("Saisir un entier :"))
S=0
k=0
while (k<=n):
  S=S+k
  k+=1
print(S)
```

**Exercice 29.** On considère une variable `k` inialisée à 0. Tant que le carré de `k` est inférieur à 1000, on augment `k` de 1. À la fin de la boucle non bornée, `k` est donc le plus petit entier tel que $\texttt{k}^2 > 1000$. Il reste donc à enlever 1 à `k` pour trouver la valeur cherchée.

En langage naturel :

$k \leftarrow 0$
Tant que $(k^2 \leqslant 1000)$
    $k = k + 1$
Fin Tant que
$k \leftarrow k - 1$

En Python :

```
k=0
while (k**2<=1000):
  k+=1
k=k-1
```

**Exercice 30.**

```
n=0
S=0
while (S<=1000):
  n+=1
  S=S+n**2
```

**Exercice 31.** Le nombre $N$ d'entiers naturels $n$ non nuls tels que $n^n \leqslant 1000000$ n'est autre que le plus grand entier $n$ tel que $n^n \leqslant 1000000$. On calcule donc le plus petit entier $m$ tel que $m^m > 1000000$ et alors $N = m - 1$

```
n=1
while (n**n<=1000000):
  n+=1
n=n-1
```

**Exercice 32.**

```
c=input("Saisir un caractère :")
n=int(input("Saisir un entier :"))
k=1
while (k<=n):
  print(c*k)
  k+=1
```

## 5) Fonctions

La fonction `lpg` définie à la fin du cours renvoie le plus grand des entiers `a` et `b`.
Le programme suivant demande à l'utilisateur 3 entiers et affiche le plus grand des 3.

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

a=int(input("Saisir un premier entier :"))
b=int(input("Saisir un deuxième entier :"))
c=int(input("Saisir un troisième entier :"))
print(lpg(a,lpg(b,c)))
```

**Exercice 33.** On considère le programme suivant.

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

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

1. Il n'y a pas de **return** dans la fonction **somme** donc la fonction **print** n'a rien à afficher. On obtient donc **None** à l'affichage.
2. Le programme modifié est

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

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

ou bien

```
def somme(a,b):
  return(a+b)

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

**Exercice 34.**

1.

```
def difference(a,b):
  return(a-b)
```

2.

```
def produit(a,b):
  return(a*b)
```

**Exercice 35.**

```
def min(a,b):
  if (a<b):
    return(a)
  else:
    return(b)
```

**Exercice 36.**

1.

```
def est_isocele(a,b,c):
  if (a==b or a==c or b==c):
    return('isocèle')
  else:
    return('non isocèle')
```

2.

```
def est_rectangle(a,b,c):
  if (a**2+b**2==c**2 or a**2+c**2==b**2 or b**2+c**2==a**2):
    return('rectangle')
  else:
    return('non rectangle')
```

**Exercice 37.**

1.

```
def som_div(n):
  S=0
  for d in range(1,n+1):
    if (n%d==0):
      S=S+d
  return(S)
```

2.

```
C=0
for n in range(1,1001):
  if (som_div(n)==2*n):
    C+=1
```

**Exercice 38.**

```
def factorielle(n):
  P=1
  for k in range(1,n+1):
    P=P*k
  return(P)
```

**Exercice 39.**

1.

```
def syracuse(n):
  if (n%2==0):
    return(n//2)
  else:
    return(3*n+1)
```

2.

```
n=int(input("Saisir un entier :"))
print(n)
while (n!=1):
  n=syracuse(n)
  print(n)
```

## 6) Les modules

**Exercice 40.**

1.

```
from math import *

def hypotenuse(a,b):
  return(sqrt(a**2+b**2))
```

2.

```
from math import *

def hypotenuse(a,b):
  return(sqrt(a**2+b**2))

a=float(input("Saisir un entier a :"))
b=float(input("Saisir un entier b :"))
print("L'hypoténuse mesure", hypotenuse(a,b))
```

**Exercice 41.** Pour tester si $n$ est un carré parfait, on teste si $\sqrt{n}$ est un entier.

```
from math import *

def carreParfait(n):
  if (int(sqrt(n))==sqrt(n)):
    return("vrai")
  else:
    return("faux")
```

**Exercice 42.** Pour simuler un tirage de *pile* ou *face*, on effectue un tirage aléatoire de 0 ou 1 et on associe 0 à pile et 1 à face. (Ce choix est arbitraire, on pourrait très bien faire le contraire, évidemment).

```
from random import *

def pileOuFace():
  a=randint(0,1)
  if (a==0):
    return("pile")
  else:
    return("face")
```

**Exercice 43.**

```
from random import *

def lancerDeDe():
  return(randint(1,6))
```

**Exercice 44.**

```
from random import *

def lancerDeDe():
  return(randint(1,6))

def tirageMonopoly():
  return(lancerDeDe()+lancerDeDe())
```

**Exercice 45.**

```
from random import *

def lancerDeDe():
  return(randint(1,6))

n=int(input("Saisir un entier naturel n :"))
c=0
for k in range(n):
  if (lancerDeDe()==6):
    c+=1;
print("Le fréquence de 6 obtenus est", c/n)
```

**Exercice 46.**

```
from random import *

def lancerDeDe():
  return(randint(1,6))

c=1
while (lancerDeDe()!=6):
  c+=1;
print("Il a fallu", c, "lancers pour obtenir le premier 6.")
```

**Exercice 47.**

1.

```
from random import *

a=return(randint(2,10))
b=return(randint(2,10))
p=a*b
print("Quel est le produit de", a, "par", b,  "?")
r=int(input("Votre réponse :"))
if (r==p):
  print(a,"*",b,"=",p,". Votre réponse est juste.")
else:
  print(a,"*",b,"=",p,". Votre réponse est fausse.")
```

2.

```
from random import *

c=0
for k in range(10):
  a=randint(2,10)
  b=randint(2,10))
  p=a*b
  print("Quel est le produit de", a, "par", b,  "?")
  r=int(input("Votre réponse :"))
  if (r==p):
    print(a,"*",b,"=",p,". Votre réponse est juste.")
    c+=1
  else:
    print(a,"*",b,"=",p,". Votre réponse est fausse.")
print("Vous avez obtenu", c, "bonnes réponses.")
```

**Exercice 48.**

1.

```
from random import *

a=randint(1,1000)
r=0
while (r!=a):
  r=int(input("Saisir un entier entre 1 et 1000 :"))
  if (r>a):
    print("Le nombre cherché est plus petit.")
  elif(r<a):
    print("Le nombre cherché est plus grand.")
  else:
    print("Bravo ! Vous avez trouvé !")
```

2.

```
from random import *

a=randint(1,1000)
r=500
while (r!=a):
  if (r>a):
    r-=1
  else:
    r+=1
print("Le nombre cherché est", r)
```