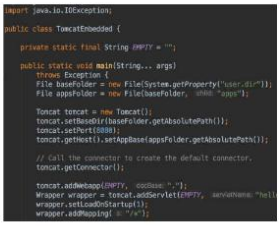
Chapitre 1 – Programmation en Python

1. Qu'est-ce-que la programmation ?

La **programmation** (ou codage) est l'activité consistant à écrire des programmes exécutés par un ordinateur, afin qu'il puisse réaliser différentes tâches.

Ici, le mot « ordinateur » est à prendre au sens large : les téléphones portables, les consoles de jeu, les appareils connectés… sont aussi des ordinateurs.

Il existe de nombreux langages de programmation, parmi lesquels :
- le langage **assembleur** est un des langages « bas niveau », compréhensible directement par les machines
- les langages **C et C++** ont été très utilisés dans les années 90, mais sont assez complexes
- le langage **HTML**, assez facile, sert pour la conception de sites internet
- le langage **Java**, plus récent, sert pour la création d'applications mobile, et en entreprise
- le langage **Python**, un des plus simples et des plus puissants, est apparu plus récemment et prend de l'ampleur. C'est celui que nous allons utiliser !



2. Instructions

Un programme est une suite d'instructions données à l'ordinateur. Ces instructions sont d'abord écrites dans une « console », puis exécutées quand le programme est prêt.

La première instruction à connaître est **`print`**, pour écrire un message à l'écran.
Exemple : **`print("Bonjour !")`** affichera « Bonjour ! » à l'écran.

Toute instruction doit être suivie de parenthèses, et dans le cas de **`print`**, le message doit être écrit entre guillemets.

Exemple 1 Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1 print("Un message")
2 print("Un autre message")
```

b.
```
1 print("Bonjour !")
2 print(ça va ?)
```

c.
```
1 print("aaa")
2 print("bbb")
3 print("")
4 print("ccc")
```

d.
```
1 print("Voici")
2
3 print("un autre exemple")
```

e.
```
1 m = "Un message"
2 print(m)
```

f.
```
1 m = "aaa"
2 n = "bbb"
3 print(m + n)
```

Exemple 2 Écrire un programme qui affiche exactement le texte suivant :

```
Une première ligne_
 une deuxième  ligne
```

Exemple 1

a. Un message
Un autre message

b. *Ce programme ne fonctionne pas : il manque les guillemets à la ligne 2*

c. aaa
bbb

ccc

d. Voici
Un autre exemple

e. Un message

f. aaabbb

Exemple 2

```
print("Une première ligne_")
print(" une deuxième  ligne")
```

3. Boucles for

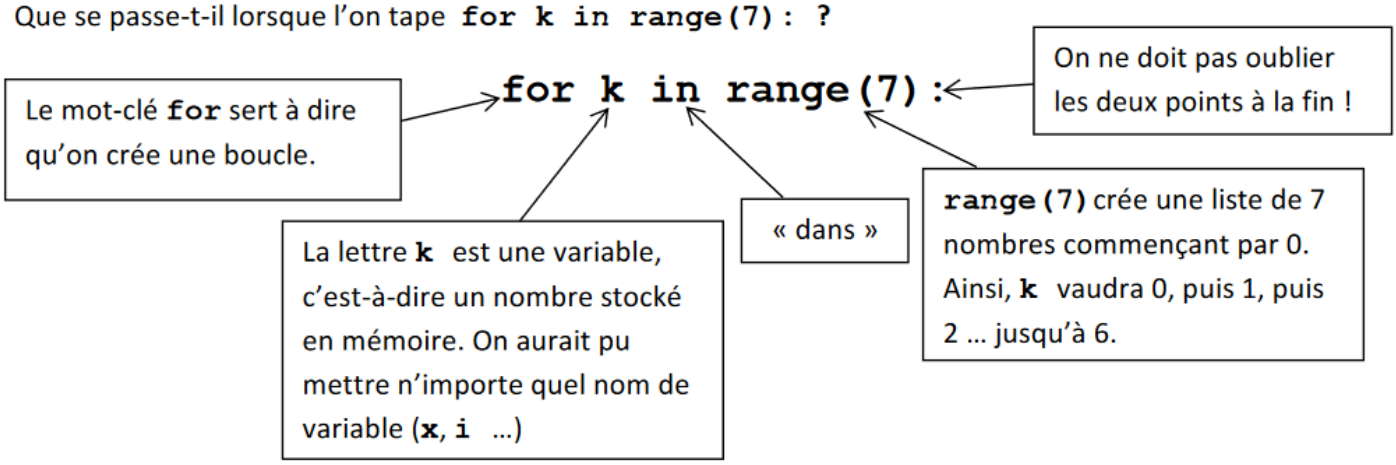
3a. Répétitions

Le moyen le plus simple de répéter des instructions est la boucle **`for`**.
Le code suivant :

```
for k in range(7) :
    ...
```

répétera les instructions ... **`7`** fois.

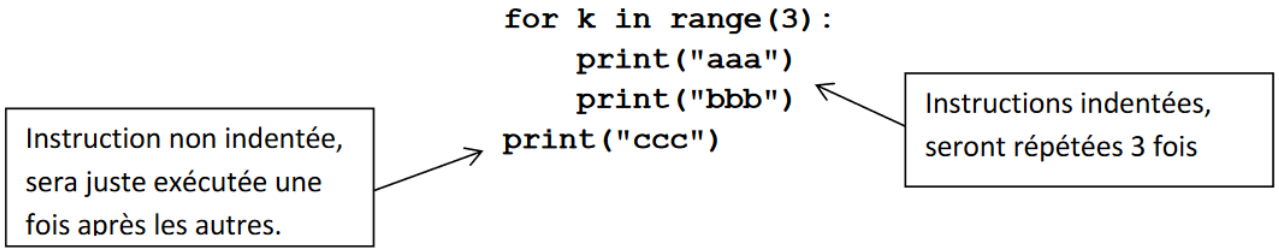
Que se passe-t-il lorsque l'on tape **`for k in range(7) :`** ?



En résumé, **`for k in range(7) :`** signifie « pour **`k`** allant de 0 à 6 ».
Ainsi, la boucle **`for`** crée une variable (nous parlerons des variables plus tard) et répète les instructions, avec la variable qui prend successivement les valeurs de la liste donnée, généralement créée avec **`range`**.

3b. Indentation

Les instructions à répéter dans une boucle doivent être **indentées**, c'est-à-dire précédées d'un espace, qu'on appelle alors l'**indentation**.



L'indentation permet de créer des « blocs » de code. Elle sera très souvent utilisée en Python. Ainsi, contrairement au saut de ligne, les **espaces précédant une ligne** sont très importants, ils ont du sens en Python !

Exemple : écrire ce qu'affichera le code ci-dessus.

Exemple
```
aaa
bbb
aaa
bbb
aaa
bbb
ccc
```

Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1 for x in range(3):
2     print("a")
3 print("b")
```

b.
```
1 for x in range(3):
2 print("a")
3 print("b")
```

c.
```
1 for i in range(3):
2     print("a")
3     print("b")
```

d.
```
1 for x in range(3):
2     print("a")
3 for y in range(3):
4     print("b")
```

e.
```
1 for x in range(250)
2     print("a")
```

f.
```
1 for k in range(3):
2     print("a")
3       print("b")
```

g.
```
1 for k in range(2):
2     print("a")
3
4     print("b")
5 print("c")
```

h.
```
1 for x in range(3):
2     print("a")
3 print("b")
4     print("c")
```

a.
a
a
a
b

b. *Ce programme ne fonctionne pas : les instructions sont mal indentées.*

c.
a
b
a
b
a
b

d.
a
a
a
b
b
b

e. *Ce programme ne fonctionne pas : il manque les deux points après range(250)*

f. *Ce programme ne fonctionne pas : les instructions sont mal indentées (ligne 3)*

g.
a
b
a
b
c

h. *Ce programme ne fonctionne pas : les instructions sont mal indentées (ligne 4)*

4. Variables

4a. Opérations

Il est possible de faire des calculs en Python, et d'afficher leur résultat avec **`print`**.

Outre les 4 opérations **`+`** **`-`** **`*`** et **`/`**, Python propose deux opérateurs liés aux divisions :

`//` donnera le quotient d'une division euclidienne, par exemple **`38 // 5`** vaut **7**.
`%` donne le reste d'une division euclidienne, par exemple **`38 % 5`** vaut **3**.

La **puissance** s'obtient avec deux symboles **`*`** : ainsi **`2**3`** vaut $2^3 = 8$.

La **virgule** des nombres décimaux s'écrit avec un point comme dans les pays anglo-saxons.
En Python, le symbole virgule a une autre utilité, il sert de séparateur entre plusieurs valeurs.

Le symbole **`+`** sur des chaînes de texte permet de les **concaténer**, c'est-à-dire de les coller.
Si on veut concaténer du texte avec des nombres, il faut d'abord appliquer la fonction **`str`**.
Par exemple, **`print("a"+str(5+2))`** affichera **`a7`** .

Exemple Dans chaque cas, écrire ce qu'affichera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1 print(151.3 + 40.9)
```
b.
```
1 print(3+5*4)
```
c.
```
1 print("7.1 + 5")
```
d.
```
1 print(13/2)
```
e.
```
1 print(13//2)
```
f.
```
1 print(13 % 2)
```
g.
```
1 print("ab"+"cd")
```
h.
```
1 print(ab + cd)
```
i.
```
1 print("ab + cd")
```

a. 192.2

b. 23

c. 7.1 + 5

d. 6.5

e. 6

f. 1

g. abcd

h. *Ce programme ne marche pas : il manque les guillemets*

i. ab + cd

4b. Variables

Une variable est un nombre ou un texte qu'on stocke dans la mémoire du programme en lui donnant un nom, pour s'en resservir ensuite.

En Python, on crée des variables en leur affectant une valeur avec le symbole **=**, qui n'a donc pas tout à fait le même sens qu'en mathématiques. Ainsi, si on écrit **`x = 3`** cela signifie que la variable **x** prendra la valeur **3**. On appelle cela une **affectation**. On peut ensuite l'afficher ou faire des calculs avec.

Exemple 1 Dans chaque cas, écrire ce qu'affichera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1  x = 4
2  y = 7
3  print(x * y)
```

b.
```
1  nombreA = 50.8
2  print(nombreA)
3  nombreB = 2
4  print(nombreA/nombreB)
```

c.
```
1  nombre A = 50.8
2  7 = x
3  print(3x)
```

Une fois affectée, une variable peut être modifiée : on peut lui réaffecter une nouvelle valeur. On peut même utiliser la valeur précédente de la variable pour cela. Ainsi, le code ci-contre affichera **4**. En effet, **a** vaut d'abord **3**, puis le code **`a = a + 1`** augmente sa valeur de **1**.

```
a = 3
a = a + 1
print(a)
```

Exemple 2 Dans chaque cas, écrire ce qu'affichera le programme, s'il est valide. Sinon, indiquer pourquoi.

d.
```
1  x = 7.1
2  print(x)
3  x = 6
4  print(x)
```

e.
```
1  x = 5
2  print(x)
3  x = x + 3
4  print(x)
```

f.
```
1  x = 10
2  x = x + 3
3  x = 2*x - 5
4  print(x)
```

g.
```
1  x = 5
2  y = 6
3  x = x * y
4  print(x)
5  print(y)
```

h.
```
1  x = 3
2  for k in range(6):
3      x = x * 2
4      print(x)
```

i.
```
1  x = 5
2  y = y + 3
3  print(x + y)
```

a. 28

b. 25.4

c. *Ce programme ne marche pas : il y a une erreur à chaque ligne.*

d. 7.1
6

e. 5
8

f. 21

g. 30
6

h. 6
12
24
48
96
192

i. *Ce programme ne marche pas : à la ligne 2, la variable y n'a pas été définie avant d'être modifiée.*

4c. Variables et boucles for

Le code permettant de créer une boucle, **`for x in range(7):`** crée une variable **x**.
Lors de la première exécution de la boucle, cette variable vaut **0**, et cette valeur augmente de **1** à chaque exécution de la boucle.
Ainsi, le code suivant :

```
for x in range(7):
    print(x)
```

affichera les nombres entiers de 0 à 6.

Exemple Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
for k in range(4):
    print(k)
```

b.
```
for x in range(3):
    print(k*2)
```

c.
```
for i in range(3):
    print(i+5)
```

d.
```
for x in range(5):
    print(x)
    print(x*3+1)
```

e.
```
x = 1
for k in range(4):
    x = x + k
    x = x * 3
    print(x)
```

f.
```
a = 10
for k in range(5):
    a = a + k
print(a)
```

a.
0
1
2
3

b. *Ce programme ne marche pas : la variable k n'existe pas.*

c.
5
6
7

d.
0
1
1
4
2
7
3
10
4
13

e.
3
12
42
135

f.
20

5. Fonctions

5a. Définitions

Les fonctions permettent de réutiliser du code plusieurs fois.
Le mot def sert à définir une fonction, qui pourra ensuite être appelée dans le reste du code.

Une fonction se définit de la façon ci-contre :
(**`ma_fonction`** est le nom de la fonction, vous pouvez choisir celui que vous voudrez)

```
def ma_fonction():
    ...
    ...
```

code de la fonction

Une fois la fonction définie, il suffit d'utiliser l'instruction **`ma_fonction()`** pour l'exécuter.
Ne pas oublier les **deux points** et l'**indentation**, comme pour les boucles for !

Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1 def ma_fonction():
2     print("aaa")
3     print("bbb")
4
5 ma_fonction()
6 print("ccc")
7 ma_fonction()
8 ma_fonction()
```

b.
```
1 print("a")
2 une_fonction()
3
4 def une_fonction():
5     print("b")
6     print("c")
7
8 une_fonction()
```

c.
```
1 def test():
2     for k in range(3):
3         print("a")
4
5 print("b")
```

d.
```
1 def une_fonction():
2     print("a")
3     print("b")
4
5 for k in range(3):
6     une_fonction()
7 print("a")
```

a.
aaa
bbb
ccc
aaa
bbb
aaa
bbb

b. *Ce programme ne marche pas : à la ligne 2, la fonction une_fonction n'est pas encore définie.*

c. b

La fonction test n'est jamais utilisée, elle est juste définie.

d.
a
b
a
b
a
b
a

5b. Arguments

Il est possible de créer des fonctions avec des paramètres (aussi appelés arguments), dont le nom est entre parenthèses au moment de la définition.

Lors de la définition, on peut indiquer un **nom de variable** entre parenthèses. Lors de l'**appel** de la fonction, il faudra alors préciser la **valeur de cette variable** entre parenthèses.

```
def ma_fonction(k):
    ...
    ...
```

code de la fonction

Une fonction peut attendre plusieurs argument, exemple : `ma_fonction(x,y)`

Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.

```
1 def incremente(k):
2     print(k+1)
3
4 incremente(5)
5 incremente(100)
6 incremente(1.3)
```

b.

```
1 def double_triple(y):
2     print(2 * y)
3     print(3 * y)
4
5 double_triple(5)
6 double_triple(10)
```

c.

```
1 def direBonjour(nom):
2     print("Bonjour, " + nom)
3
4 direBonjour("Alphonse")
5 direBonjour("Léontine")
6 direBonjour("")
```

d.

```
1 def parle(f):
2     for k in range(f):
3         print("bla")
4
5 parle(2)
6 print("bli")
7 parle(3)
```

e.

```
1 def operation(x,y):
2     print(x+5*y)
3
4 operation(3,2)
5 operation(-7,2)
```

f.

```
1 def fonctionA(x):
2     print(3*x)
3
4 def fonctionB(x):
5     print(x+2)
6
7 fonctionB(7)
8 fonctionA(5)
9 fonctionB(10)
```

a.
6
101
2.3

b.
10
15
20
30

c.
Bonjour, Alphonse
Bonjour, Léontine
Bonjour,

d.
bla
bla
bli
bla
bla
bla

e.
13
3

f.
9
15
12

5c. Valeurs de retour

Une fonction peut renvoyer une valeur avec le mot return.
Cela leur permet de fonctionner comme les fonctions en mathématiques.

A la fin du code d'une fonction, si le mot **return** est utilisé, l'appel de la fonction produira une valeur, qui peut être utilisé dans le reste du code.

```
def ma_fonction(x):
    ...
    return ...
```

Exemple 1 On définit une fonction **f** ci-contre :
a. Qu'affichera `print(f(7))` ? et `print(f(-3))` ?
b. Mêmes questions avec la fonction **g** ci-contre.

```
def f(x):
    return 5*x+2
```

```
def g(a):
    a = a + 5
    return a*a
```

Exemple 2 On considère le programme de calcul ci-contre.
a. Tester ce programme pour la valeur 17, puis pour la valeur -3.
b. Créer une fonction **programmeA** en Python qui renvoie le résultat du programme de calcul.
c. En appelant x le nombre de départ, exprimer le résultat du programme de calcul en fonction de x.
d. En déduire une autre fonction **programmeB** qui renvoie le résultat du programme de calcul.

Choisir un nombre
Lui ajouter 4
Multiplier la somme par 5
Soustraire le nombre de départ.

Exemple 3 On veut définir la fonction Python **f** qui à un nombre associe son cube.
*(Pour calculer des puissances en Python, on utilise le symbole multiplication deux fois : **)*
Mais la définition ci-contre comporte trois erreurs. Lesquelles ?

```
def f(t)
    x = x**3
return x
```

Exemple 4 Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
1 def f(x):
2     return 2*x+1
3
4 print(f(5))
5 f(7)
6 print(f(9)+10)
```

b.
```
1 def g(x):
2     print("Je calcule...")
3     return x*x+1
4
5 print(g(10))
6 print(g(-4))
```

Exemple 1 **a.** `print(f(7))` affichera 37, et `print(f(-3))` affichera -13.
b. `print(g(7))` affichera 144, et `print(g(-3))` affichera 4.

Exemple 2 **a.** En choisissant 17, on obtient 88, et en choisissant -3, on obtient 8.
b.
```
def programmeA(x) :
    return (x + 4)* 5 - x
```
c. L'expression $(x + 4) \times 5 - x$ se développe pour devenir $4x + 20$.
d.
```
def programmeB(x) :
    return 4*x + 20
```

Exemple 3 La lettre entre parenthèses devrait être x, le return devrait être indenté et les deux points ont été oubliés.

Exemple 4

a. 11
29
La ligne 5 n'affiche rien, car il n'y a pas d'instruction print.

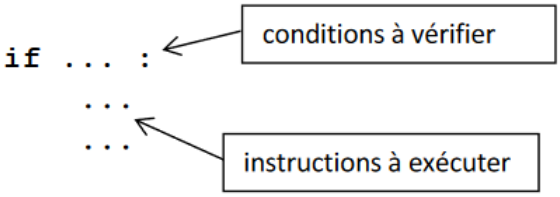
b. Je calcule...
101
Je calcule...
17

6. Conditions

6a. If et else

Le mot if (« si ») permet de n'exécuter du code que si une condition est vérifiée.
Le mot else (« sinon ») permet d'exécuter du code dans le cas contraire.

Les instructions indentées après un **if** ne s'exécutent que si la condition est réalisée.
Les conditions sont généralement des comparaisons de nombres, par exemple **if a > 5 :**



On peut utiliser les symboles **<** (inférieur), **>** (supérieur), **<=** (inférieur ou égal), **>=** (supérieur ou égal), **==** (égal) et **!=** (différent). Notez bien que le test d'égalité s'écrit avec deux symboles **==** !

Le mot **else** permet d'exécuter des instructions si la condition n'est pas réalisée.

```
if ... :
    ...
else :
    ...
```

Exemple 1 Dans chaque cas, écrire ce que fera le programme, s'il est valide. Sinon, indiquer pourquoi.

a.
```
def f(x):
    if x > 5 :
        print("Plus grand !")

f(8)
f(-7)
f(5)
f(5.01)
```

b.
```
def f(x):
    if x >= 10 :
        return x+3
    else :
        return x-2

print(f(4))
print(f(12))
print(f(10))
```

c.
```
def compare(x,y):
    if x < 2*y :
        return "a"
    else :
        return "b"

print(compare(10,7))
print(compare(10,5))
```

Exemple 2 Un service de location de véhicules pratique les tarifs suivants :
- pour un trajet de 100 km ou moins, la location coûte 30 €
- pour un trajet de plus de 100 km, la location coûte 0,30 € par kilomètre parcouru

Écrire une fonction **location** qui renvoie le prix d'une location en fonction du nombre de kilomètres.

Exemple 1

a.
Plus grand !
Plus grand !

b.
2
15
13

c.
a
b

Exemple 2

```
def location(x) :
    if x < 100 :
        return 30
    else :
        return 0.3*x
```

6b. Combinaisons

Le mot elif (contraction de « else if », « sinon si ») combine else et if.
On peut combiner des conditions avec and (et) et or (ou).
Avec ces mots, on peut créer des fonctions correspondant à des problèmes concrets.

Exemple 1 Dans chaque cas, écrire ce que fera le programme, s'il est valide.
Sinon, indiquer pourquoi.

a.

```
def f(x):
    if x <= 5 or x >= 10 :
        return x * 2
    else :
        return x + 3

print(f(12))
print(f(6))
print(f(5))
```

b.

```
def g(x,y):
    if x < y and x > 2 :
        return x * y
    else :
        return x + y

print(g(1,5))
print(g(3,7))
print(g(9,6))
```

Exemple 2 Écrire une fonction `f(x)` qui renvoie le double de `x` s'il est strictement inférieur à 5, le triple de `x` s'il est compris entre 5 et 20 inclus, et le quadruple de `x` dans les autres cas.

Exemple 3 Pendant les soldes, un magasin pratique les tarifs suivants :
- si le montant des achats dépasse 50 €, une remise de 10% est appliquée,
- si le montant des achats dépasse 100 €, la remise est de 25%,
- dans les autres cas, aucune remise n'est appliquée.

Écrire une fonction `prix_solde(x)` qui prend en argument le montant `x` des achats, et qui renvoie le prix une fois la remise éventuelle appliquée.

Exemple 1

a. 24
9
10

b. 6
21
15

Exemple 2

```
def f(x) :
    if x < 5 :
        return 2*x
    elif x <= 20 :
        return 3*x
    else :
        return 4*x
```

Exemple 3

```
def prix_solde(x) :
    if x > 100 :
        return 0.75*x
    elif x > 50 :
        return 0.9*x
    else :
        return x
```

7. Boucles while

Le mot while (« tant que ») permet d'exécuter du code répétitivement, tant qu'une condition est vérifiée. Attention à ne pas créer de boucles infinies !

Les instructions indentées après un **while** s'exécutent tant que la condition est réalisée.

```
while ... :
    ...
```

Il faut donc faire attention : si la condition reste toujours vraie, le programme s'exécutera sans s'arrêter, ce qui peut bloquer l'ordinateur dans certains cas. On parle alors de boucles infinies.

Les boucles while servent quand on veut répéter des instructions, mais que le nombre de répétitions n'est pas connu à l'avance.

Exemple 1 Dans chaque cas, écrire ce que fera le programme, s'il ne provoque pas de boucle infinie.

a.
```
x = 14
while x > 6 :
    x = x - 2
    print(x)
```

b.
```
y = 50
while y >= 100 :
    y = y - 10
    print(y)
```

c.
```
x = 30
while x < 50 :
    x = x - 3
    print(x)
```

d.
```
y = 30
n = 0
while y < 1000 :
    y = y * 2
    n = n + 1
    print(n)
```

Exemple 2 Baptiste dépose 3 000€ sur un compte bancaire. Chaque mois, il rajoute 50 €.
Écrire une fonction `f(x)` qui renvoie le nombre de mois qu'il lui faudra pour avoir `x` euros sur son compte.

Exemple 3 172 tigres vivent dans une réserve naturelle.
Cette espèce étant en voie de disparition, elle y est protégée. Il est prévu que le nombre de tigres vivant dans la réserve augmente de 6% par an.
Écrire une fonction `f(x)` qui renvoie le temps nécessaire (en années) pour que le nombre de tigres atteigne la valeur donnée par `x`.

Exemple 1

a.
12
10
8
6

b. *Ce programme n'affiche rien : la condition du while n'est jamais remplie.*

c. *Ce programme provoque une boucle infinie : la condition du while est toujours remplie.*

d.
1
2
3
4
5
6

Exemple 2

```
def f(x) :
    s = 3000
    mois = 0
    while s < x :
        s = s + 50
        mois = mois + 1
    return mois
```

Exemple 3

```
def f(x) :
    tigres = 172
    annees = 0
    while tigres < x :
        tigres = tigres * 1.06
        annees = annees + 1
    return annees
```