

Introduction à la programmation avec Python 3

Micha Hersch

17 août 2020

1 Introduction

Python est un langage de programmation, c'est à dire que c'est un langage qui permet à un humain (le programmeur) d'expliquer à un ordinateur ce qu'il doit faire. Un programme est une série d'instructions écrites de façon à ce que l'ordinateur les comprenne et qu'il peut exécuter une à une. Toutes les applications, sites web, et appareils numériques, fonctionnent grâce à des programmes qui sont écrit soit en Python (comme par exemple youtube ou dropbox) soit en d'autres langages. Certains petit programmes ont juste quelques instructions, alors que d'autres en ont des millions.

Le langage Python, comme la plupart des langages de programmation, est basé sur l'anglais, donc il faut savoir un peu d'anglais pour se rappeler de la significations des instructions.

Voici un exemple d'un tout petit programme en Python qui ne contient qu'une seule instruction :

Exemple 1

```
print("bonjour")
```

 1

En anglais, "print" signifie "imprime". En Python, l'instruction print fait que l'ordinateur écrit à l'écran le contenu de la parenthèse qui vient après.

Exercice 1 Ecrire et exécuter le programme 1.

Vérifier que l'ordinateur suit bien l'instruction qui lui est donnée. Changer le texte pour que l'ordinateur écrive autre chose, par exemple "au revoir !"

Il est souvent utile de mettre des commentaires dans un programme, pour expliquer ce qu'il fait. En Python un commentaire est introduit par le caractère #. Tout ce qui vient après et jusqu'à la fin de la ligne, n'est pas lu par l'ordinateur. Cela sert uniquement à l'humain qui va lire le programme.

Exemple 2

```
# un tout petit programme
print("bonjour") # salutations
```

 1
2

Exercice 2 Ajouter un commentaire dans le programme de l'exercice précédent. Vérifier qu'il ne change pas le déroulement du programme.

Parfois, l'ordinateur ne comprend pas les instructions qui sont données dans le programme. Cela signifie qu'il y a une erreur dans le programme, et que l'ordinateur ne peut pas l'exécuter. C'est un peu comme si on lui disait des mots qu'il ne comprend pas.

Exercice 3 Enlever les guillemets dans le programme 1 et l'exécuter. Qu'est-ce qui se passe ?

Le chapitre suivant explique pourquoi l'ordinateur a besoin des guillemets pour comprendre cette instruction.

2 Les variables simples

2.1 Types d'objet simples

De la même manière qu'une recette de cuisine indique au cuisinier comment manipuler des ingrédients pour obtenir un plat, un programme indique au processeur comment manipuler des objets pour obtenir une application. En Python, les types d'objets de base sont les suivants :

1. les nombre entiers, appelés `int` : 2, 45, -4, 56
2. les nombres à virgules, appelés `float` : 2.3, 6.5, 78.9, -89.0. La décimale est indiquée par un point (et non une virgule) comme cela est l'usage dans les pays anglo-saxons.
3. les chaînes de caractères, appelés `str` (pour *string*, chaîne) : "bonjour", "au revoir", "3432", "232.543". Il s'agit simplement d'une suite de caractères (lettre, chiffre, signe de ponctuation, espace) placées entre guillemets ou (de manière équivalente) entre apostrophes.
4. Les booléens, appelés `bool` : True, False. Ce type d'objet ne peut prendre que deux valeurs True (vrai) et False (faux), à écrire avec une majuscule.

Exercice 4 Indiquer lesquels des objets suivants sont valides en Python et le cas échéant de quel type d'objet il s'agit. :

- a) "rewr"
- b) 34
- c) gdru
- d) 5
- e) 5.0
- f) 'julien34'
- g) 18gd
- h) "65.5"
- i) True
- j) "False"

Pour vérifier les réponses, entrer les objets ci-dessus dans un terminal Python. Il est possible de vérifier le type d'objet en utilisant la fonction `type()`, par exemple `type("rewr")`.

2.2 Opérateurs

Pour chaque type d'objet, plusieurs opérateurs sont définis qui permettent de manipuler ces objet. Pour le nombre entiers, les opérateurs suivants sont les plus courant :

- + (addition),
- - (soustraction)
- * (multiplication)
- / (division)
- // (division entière)
- % (modulo ou reste de la division entière)

Toutes les opérations ci-dessus retournent nombre entier, sauf la division (/) qui retourne un nombre à virgule. Les opérateurs similaires sont définis pour les nombres à virgules.

Il existe également les opérateurs de comparaison qui retournent des booléens (**True** ou **False**) :

- > (plus grand que)
- < (plus petit que)
- >= (plus grand ou égal à)
- <= (plus petit ou égal à)
- == (égal à)
- != (non égal à)

Enfin, les opérateur logiques suivants sont définis sur les booléens :

- **not** (non)
- **and** (et)
- **or** (ou)
- == (égal à)
- != (non égal à)

Exercice 5 Prédire le résultat des opérations suivantes en Python et vérifier en les entrant dans un terminal Python :

- a) 3+2
- b) 2.3*2
- c) 3/2
- d) 3//2
- e) 3.0/2.0
- f) 3.0//2.0
- g) 34-65
- h) 3 > 2
- i) -2 < -5.0
- j) 3 == 3.0
- k) **True and False**
- l) **True or False**
- m) **True != False**
- n) (2>3) == (5>10)

Comme en mathématique, les opérateurs ont un ordre de priorité. Ainsi si le calcul $4+2*3$ correspond à $4+(2*3)$ parce que la multiplication a priorité sur l'addition. Si on veut d'abord effectuer $4+2$ et multiplier le résultat par 3, alors il faut l'indiquer avec des parenthèses : $(4+2)*3$. De manière générale, la multiplication et la division ont priorité sur l'addition et la soustraction, et les opérateurs de calcul ont la priorité sur les opérateurs de comparaison. Si deux opérateurs ont le même niveau de priorité, alors celui de gauche est effectué en premier.

Exercice 6 Prédire le résultat des instructions suivantes en Python

- a) $3+2*5$
- b) $3*5+3$
- c) $6*2/4$
- d) $6/4*2$
- e) $6.0/4*2$
- f) $3*4<5*2$
- g) $2*4!=5*2$
- h) $2+4*2 == (2+3)*2$
- i) $2==1+1$ or $2==3$
- j) $2==1+1$ and $2<=3$

Entrer ces expressions dans un terminal Python pour vérifier les réponses.

2.3 Assignation de variables

Afin de manipuler des objets, il est utile de leur donner des noms, c'est ce qu'on appelle une assignation. Ceci se fait en utilisant l'opérateur `=` (à ne pas confondre avec l'opérateur `==` et qui est différent du `=` mathématique). Les noms de variable, s'écrivent sans guillemets, mais il ne peuvent pas commencer par un chiffre, ni contenir d'accent, d'apostrophe ou de caractères spéciaux.

Exemple 3

```
a = "Bonjour"
print(a)
```

1
2

Dans l'exemple ci-dessus, on donne le nom `a` à la chaîne de caractère `"bonjour"`. On peut dire que la variable `a` contient la chaîne de caractère `"bonjour"`. Une fois un objet nommé, on peut le manipuler en l'appelant par son nom. Dans cet exemple on demande à l'ordinateur d'écrire le contenu de `a`, c'est-à-dire `"bonjour"`.

Chaque nom ne peut correspondre qu'à un seul objet, donc si on redonne un nom déjà utilisé, l'ancien objet désigné par ce nom est oublié. Autrement dit, le contenu de cette variable est remplacé avec le nouvel objet.

Exemple 4

```
a = "Bonjour" 1
a = "Au revoir" 2
print(a) 3
```

Dans le programme ci-dessus, la variable **a** désigne d'abord l'objet "Bonjour", puis elle désigne l'objet "Au revoir". C'est cette dernière expression qui apparaît à l'écran avec la dernière ligne du programme.

Une variable peut apparaître des deux côtés d'une assignation (d'un signe =). En ce cas, l'ordinateur calcule d'abord le côté droit, puis assigne le résultat à la variable apparaissant à gauche du signe =.

Exemple 5

```
b = 5 1
b = b*2 2
print(b) 3
```

Dans cet exemple, la première instruction assigne 5 à la variable **b**. Pour la seconde instruction, il multiplie **b** par 2 (ce qui donne 10), puis le résultat est assigné à **b** qui contient maintenant la valeur 10. La troisième instruction affiche cette valeur.

Exercice 7 Mettre l'objet 6 dans une variable appelée **a**, l'imprimer à l'écran puis le diviser par trois et le remettre dans **a**. Imprimer la nouvelle valeur de **a** à l'écran.

Exercice 8 Qu'impriment les trois programmes (6,7,8) suivants? Vérifiez vos réponses.

Exemple 6

```
x = 2
y = 3
x = y
print(x,y)
```

Exemple 7

```
x = 2
y = 3
y = x
print(x,y)
```

Exemple 8

```
x = 2 1
y = 3 2
x == y 3
print(x,y) 4
```

3 Les fonctions

Python, comme tout autre langage de programmation, contient tout une série de *fonctions*, c'est-à-dire des instructions déjà définies qui font faire quelque chose au programme. Nous en avons déjà utilisés deux au chapitre précédent, la fonction **print** qui affiche quelque chose à l'écran et la fonction **type** qui retourne le type d'un objet. L'appel d'une fonction s'effectue en indiquant la nom de la fonction, suivi d'une paire de parenthèses. Ces parenthèse contiennent les éventuels *arguments* de la fonction, c'est à dire les objets nécessaires pour que la fonction puisse être exécutée, séparés par des virgules.

Exemple 9

```
print("Hello!") 1
type(56.8)      2
a = "rouge"    3
print(a)       4
type(a)        5
```

Dans l'exemple ci-dessus, chaque appel de fonction de fait en fournissant un objet en argument. Cet objet peut être donné directement (comme dans les deux première ligne), soit par une variable (ligne 4-5).

Certaines fonctions prennent plus qu'un argument comme la fonction `pow` qui calcule la puissance de deux nombres, et qui a donc besoin de deux arguments :

Exemple 10

```
a = pow(2,3) # calcule 2 puissance 3 1
print(a)    2
```

Si vous ne donnez qu'un seul argument à la fonction `pow`, python vous indiquera une erreur. Souvent, les fonctions retournent une valeur, qui contient le résultat de la fonction. Dans l'exemple ci-dessus ce résultat (ici 8) est stocké dans la variable `a` puis affiché.

La fonction `print` a ceci de spécial qu'elle peut accueillir zéro, un, ou plusieurs arguments.

Exemple 11

```
a = 3 1
b = 5 2
print("a vaut", a, "b vaut", 4) # print a ici 4 arguments 3
```

Exercice 9 Ecrire un programme qui calcule 3.5 à la puissance 5 et qui affiche le résultat à l'aide d'une phrase commençant par "3.5 à la puissance 5 vaut ...".

Python contient un grand nombre de fonctions, et la plupart d'entre elles sont organisées au sein de *modules* ou bibliothèques, qui ne sont rien d'autre qu'une collection de fonctions. Par exemple, le module `math` contient beaucoup de fonctions mathématiques, comme par exemple la fonction `sqrt` qui calcule la racine carrée (*square root* en anglais) d'un nombre. Pour utiliser ces fonction, il faut d'abord importer le module.

Exemple 12

```
from math import * 1
a = sqrt(9) ## calcule la racine caree 2
```

La première ligne de l'exemple ci-dessus indique que l'on va utiliser les fonctions du module `math`. La seconde ligne utilise la fonction `sqrt` pour calculer la racine carrée de 4. La description des fonctions du module `math` est disponible ici : <https://docs.python.org/fr/3/library/math.html>.

Exercice 10 Trouver dans le lien ci-dessus, la fonction permettant de calculer le sinus d'un nombre. Ecrire un programme calculant le sinus de 1 radian et affichant le résultat à l'écran.

3.1 Les fonctions d'entrées

Une des fonctions les plus utiles est la fonction `input(phrase)` qui affiche `phrase` dans le terminal et retourne la chaîne de caractères que l'utilisateur écrit dans le terminal. Cela permet à l'utilisateur de donner des informations au programme, et le résultat du programme pourra ainsi dépendre des indications de l'utilisateur.

Exemple 13

```
nom = input("Quel est votre nom?") 1  
print("Bonjour", nom) 2
```

Dans cet exemple, le programme va demander à l'utilisateur d'écrire son nom dans le terminal, est assignera la chaîne de caractère entrée par l'utilisateur à la variable `nom`. Il écrit ensuite "Bonjour" puis le nom donné par l'utilisateur.

Exercice 11 Ecrire un programme demandant d'abord le nom, puis le prénom de l'utilisateur et qui salue ensuite l'utilisateur avec son prénom et son nom.

Si l'on souhaite que l'utilisateur rentre un nombre, il faudra convertir la chaîne de caractère rentrée par l'utilisateur soit en un nombre entier avec la fonction `int`, soit en nombre à virgule avec la fonction `float`.

Exemple 14

```
annee = int(input("Quel est votre annee de naissance")) 1  
age = 2020 - annee 2  
print("Vous avez", age, "ans") 3
```

Si l'on n'appelle pas la fonction `int` dans la première ligne, `annee` sera une chaîne de caractère et la seconde ligne retournera une erreur car python ne sait pas comment soustraire une chaîne de caractère à un nombre.

Exercice 12 Ecrire un programme qui demande un nombre à l'utilisateur et affiche le carré de ce nombre dans le terminal.

4 Les structures de contrôle

Les structures de contrôle sont un élément central de la programmation. C'est cela qui permet de moduler l'exécution du programme en fonction des étapes intermédiaires. Les structures de contrôle se retrouvent de façon presque identique dans les autres langages de programmation.

4.1 L'instruction if

L'instruction `if` (si, en anglais) permet d'effectuer une liste d'instructions uniquement si une valeur booléenne (appelée la condition) est `True` (vraie) et une autre liste d'instruction sinon.

Exemple 15

```
ok = True
if ok:
    print("la variable ok est vraie")
else:
    print("la variable ok est fausse")
```

1
2
3
4
5

Dans l'exemple ci-dessus, on définit la variable booléenne `ok` à `True` (vrai). L'instruction `if` teste cette variable. Si elle est vraie, le bloc d'instructions (décalé à droite) qui vient après les deux points est exécuté. sinon, le bloc d'instructions (aussi décalé à droite) qui vient après l'instruction `else:` est exécuté. L'indentation du texte (c'est-à-dire où la ligne commence) est importante. Elle permet à l'ordinateur de savoir quelles sont les instructions qui appartiennent au bloc du `if` et quelles sont celles qui appartiennent au bloc du `else`. (Il est conseillé d'utiliser la touche de tabulation du clavier, plutôt que la touche espace, pour régler les indentations.) De même, les deux points sont nécessaires pour que l'ordinateur puisse différencier la condition des blocs d'instructions.

Exercice 13 Ecrire un programme qui demande son âge à l'utilisateur. Si l'utilisateur donne un nombre inférieur à 20, le programme écrit "Comme vous êtes jeune!", sinon il écrit "Comme vous êtes vieux!".

On peut mettre plus qu'une instruction dans un block `if` ou `else`, comme le montre l'exemple suivant :

Exemple 16

```
age = input("Quel est votre age?")
if age<18:
    print("Vous etes encore mineur.")
    print("Vous ne pouvez pas encore voter.")
else:
    print("Vous etes majeur.")
    print("Vous pouvez voter.")
print("Nous vous recontacterons.")
```

1
2
3
4
5
6
7
8

Dans cet exemple, la condition que l'on teste est `age<18`. C'est une valeur booléenne que l'on peut tester même si elle n'est pas explicitement mise dans une variable.

Exercice 14 Dans l'exemple ci-dessus, indiquer ce que le programme écrit si l'utilisateur indique un âge inférieur à 18, et ce qu'il écrit dans le cas contraire. Considérer en particulier la dernière ligne et vérifier sa réponse en exécutant le programme.

Il peut arriver que le programme n'ait rien à faire si la condition est fausse. En ce cas, on peut écrire l'instruction `if` sans la partie `else`.

Exemple 17

```
ok1 = True
ok2 = False
if ok1:
    print("ok1 est vraie")
if ok2:
    print("ok2 est vraie")
```

1
2
3
4
5
6

Dans l'exemple ci-dessus, on utilise successivement deux instructions `if`, une qui teste la variable `ok1` et l'autre qui teste la variable `ok2`. Comme seule la variable `ok1` est vraie, le programme exécute les instructions du premier `if`, mais pas celle du second.

4.2 L'instruction `while`

L'instruction `while` (qui signifie "tant que" en anglais) permet de répéter un bloc d'instructions tant qu'une condition est remplie.

Exemple 18

```
a = 1
while a < 100:
    print(a)
    a = a * 2
print("maintenant a vaut", a)
```

1
2
3
4
5

Dans cet exemple, on initialise la variable `a` à 1. Puis tant qu'elle est inférieure à 100, on l'affiche à l'écran et on la double. Ainsi, le `a` passera de 1 à 2 à 4 à 8, etc. Le programme imprime donc toutes les puissances de deux inférieures à 100.

Exercice 15 Modifier le programme ci-dessus pour qu'il imprime toutes les puissances de 10 inférieures ou égales à 10000.

Il est bien entendu possible d'imbriquer des structures de contrôle les unes dans les autres.

Exemple 19

```
from random import *
nombre = randint(1,20) # randint est une fonction du module
random
dev = int(input("Je pense à un nombre entre 1 et 20. Devinez
lequel:"))
while dev != nombre:
    if dev > nombre:
        dev = int(input("trop grand, essayez encore. "))
```

1
2
3
4
5
6

```

else:
    dev = int(input("trop petit, essayez encore. "))
print("Bravo, vous avez trouve!")

```

L'exemple ci-dessus est un jeu de devinette. Il utilise la fonction `randint(a,b)` qui retourne un entier aléatoire entre `a` et `b`. Cette fonction se trouve dans le module `random`. Pour pouvoir utiliser cette fonction, il faut importer ce module, ce qui est fait à la première ligne du programme. La fonction `abs(a)` qui retourne la valeur absolue de `a` est aussi utilisée.

Exercice 16 Exécuter le programme de l'exemple ci-dessus et vérifier son déroulement.

Ajouter une condition pour que le programme dise en plus "vous y êtes presque!" si l'utilisateur devine un nombre proche du nombre cherché (par exemple si la différence est inférieure à 3).

4.3 Les instructions break-else et continue

Il est parfois utile d'interrompre une boucle avant qu'elle ne soit entièrement réalisée. C'est ce que fait l'instruction `break`, qui interrompt l'exécution de la boucle. Le programme continue donc après la boucle.

Exemple 20

```

from random import *
nombre = randint(1,20) # randint est une fonction du module
random
dev= int(input("Je pense a un nombre entre 1 et 20. Devinez
lequel:"))
while True:
    if dev == nombre
        print("Bravo, vous avez trouve")
        break
    else:
        dev = input("Faux, essayez encore")
print("On peut passer a la suite")

```

L'instruction `continue` placée dans une boucle indique au processeur de passer directement à la prochaine itération de la boucle, tandis que l'instruction `else` placée à la fin d'une boucle est exécutée uniquement si les boucles n'ont pas été interrompues par un `break`.

5 Les types séquentiels

Certains programmes nécessitent l'utilisation d'un grand nombre de valeurs. Il serait impraticable de les mettre chacune dans une variable. Comme d'autres langages de programmation, Python offre la possibilité de stocker des séquences de valeurs dans une variable. Nous allons voir trois types de séquences disponibles en python, les listes, les ranges et les tuples.

5.1 Les listes

Pour définir une liste, on met simplement la liste des objets que contient la liste dans des crochets séparés par des virgules. Ensuite on peut accéder au contenu de la liste en indiquant entre crochets quel élément de la liste on souhaite. Attention, la numérotation des éléments d'une liste commence à zéro (et pas un), il faut donc indiquer 0 pour avoir le premier élément de la liste, 1 pour le deuxième, etc.

Exemple 21

```
# deux listes
jours = ["lundi", "mardi", "mercredi", "jeudi",
        "vendredi", "samedi", "dimanche"]
visiteurs = [200, 120, 345, 256, 123, 765, 644]
# on s'intresse au premier jour...
j = jours[0]
v = visiteurs[0]
print("le", j, "il y a eu", v, "visiteurs")
# ... et au dernier jour
print("le", jours[6], "il y a eu", visiteurs[6], "visiteurs")
```

Dans l'exemple ci-dessous, on définit deux listes, une contenant les jours de la semaine, et une contenant sept nombres entiers. Ensuite, on va chercher le premier élément de chaque liste et on les imprime. On va ensuite chercher les sixièmes éléments de chaque liste et on les imprime.

Exercice 17 Faire une liste contenant le nom des quatre saisons et une autre contenant les valeurs suivantes : 14, 22, 15, 5 qui représentent les températures moyennes. Imprimer les quatre saisons avec les quatre températures moyennes correspondantes.

Il est aussi possible de faire des listes contenant des éléments de types différents et même des listes de listes.

Exemple 22

```
objets = ["Julie", 6, "Pierre", 4, "Paul", 4]
print("Le second element est", objets[1])
```

Exemple 23

```
branches = ["math", "allemand", "anglais"]
notes = [[4,5], [4,4.5,6], [3,5,4]]
print("branche:", branches[2], "notes:", notes[2])
print("J'ai aussi fait un ", notes[1][2], "en", branches[1])
```

Exercice 18 Modifier le programme ci-dessus pour qu'il affiche la première note en anglais (le 3).

5.1.1 Les opérateurs

Tout comme les variables simples, les listes ont aussi leurs opérateurs. Les plus utilisés sont les suivant :

- `+` (concaténation) : si `a` et `b` sont des listes, `a+b` retourne les deux listes `a` et `b` mises bout-à-bout dans une seule liste
- `*` (répétition) : si `a` est une liste et `b` est un nombre entier, `a*b` retourne la liste `a` répétée `b` fois.
- `[]` (indexation) : si `a` est une liste et `b` est un nombre entier, `a[b]` retourne le `b`-ième élément de la liste, comme décrit ci-dessus (en commençant à 0). Il est possible d'utiliser cet opérateur pour extraire des tranches de la liste. Par exemple `a[2:6]` retourne une sous-liste avec les éléments 2 à 5 de `a`. On peut omettre une borne s'il s'agit de l'extrémité de la liste : `a[:6]` retourne tous les éléments de `a` jusqu'au 5^e et `a[6:]` retourne tous les éléments de `a` à partir du 6^e.

Exemple 24

```
l1 = [0,1] 1
l2 = l1 + [3,4,5] 2
l3 = l1*4 3
print(l2) 4
print(l3) 5
print(l3[4]) 6
print(l3[2:5]) 7
print(l3[3:]) 8
```

Exercice 19 Définir une liste qui contient 20 fois le nombre 3 puis 10 fois le nombre 5 et la mettre dans une variable `l`. Afficher le contenu de cette liste pour vérifier. Afficher ensuite uniquement les éléments 17 à 23 de cette liste.

5.1.2 Quelques fonctions utiles

len La fonction `len(a)` retourne la longueur de la liste `a`, c'est-à-dire le nombre d'éléments qu'elle contient.

sum La fonction `sum(a)` ("somme" en anglais) retourne la somme des éléments de la liste `a`. Elle ne peut s'appliquer que si la liste contient uniquement des nombres.

5.2 Les tuples

Les tuples sont comme des listes, à la différence qu'ils ne peuvent pas être modifiés. Ils sont initialisés avec des parenthèses plutôt que des crochets, comme dans l'exemple ci-dessous. Les opérateurs de liste décrits ci-dessus fonctionnent également avec les tuples.

Exemple 25

```
tup1 = ("a", "b") 1
tup2 = (0,1) 2
tup3 = tup1 + tup2 3
tup4 = tup3*2 4
print(len(tup4)) 5
print(sum(tup2)) 6
print(tup4[3:]) 7
```

5.3 Les ranges

Les ranges sont surtout utilisés pour stocker des intervalles de nombres entiers. C'est comme des listes, mais ils prennent beaucoup moins de place dans la mémoire car seuls le début et la fin de l'intervalle est gardée en mémoire. Une variable de type `range` peut être créée avec la fonction `range(a,b)` qui retourne un `range` contenant tous les entiers de `a` à `b-1`. L'opérateur d'indexation ainsi que les fonction `len` et `sum` peuvent aussi être utilisées avec des ranges. Un range peut être convertit en liste avec la fonction `list`

Exemple 26

```
nombres = range(0,10) 1
n = len(nombres) 2
print("Le second element est", nombres[1]) 3
print("le range contient", n, "elements") 4
print("le range est", nombres) 5
li = list(nombres) 6
print("la liste contient aussi", len(li), "elements") 7
```

Exercice 20 Ecrire un programme qui demande un nombre à l'utilisateur et qui imprime la somme des nombres de 1 à au nombre donné. Le tester en entrant 5, le programme devrait afficher 15 (1+2+3+4+5).

5.4 L'instruction for pour l'itération sur liste

L'instruction `for` permet de répéter un block d'instructions en donnant à chaque fois une autre valeur à une variable donnée.

Exemple 27

```
presidents = ["Bush", "Clinton", "Bush", "Obama"] 1
for name in presidents: 2
    print (name, "fut president") 3
```

Dans l'exemple ci-dessus, la variable `name` prendra successivement les valeurs de la liste `presidents` et l'instruction `print` sera à chaque fois exécutée. L'instruction `for` peut également être utilisée avec un tuple ou un range.

Exemple 28

```
for i in range(1,11):  
    print(i)
```

1
2

Le programme ci-dessus affiche les nombres de 1 à 10.

Exercice 21 Ecrire un programme qui affiche dix fois “cela se repete” en utilisant une boucle `for`

Exercice 22 Ecrire un programme qui demande un nombre n à l'utilisateur et affiche la somme des nombres de 1 à n .

Le modifier pour qu'il affiche le produit des nombres de 1 à n .

Exercice 23 Ecrire un programme qui demande un nombre à l'utilisateur et détermine s'il s'agit d'un nombre premier.

Indice : Utiliser l'opérateur modulo (%) ainsi que l'instruction `for-break`.

Exercice 24 - La suite de Fibonacci La suite de Fibonacci est une suite de nombres qui s'obtient de la façon suivante : les deux premiers nombres sont 0 et 1, puis les nombres suivants s'obtiennent en additionnant toujours les deux derniers nombres de la suite.

La suite commence donc ainsi 0, 1, 1, 2, 3, 5, 8, 13, 21 (= 8+13), ...

On peut montrer mathématiquement qu'en divisant deux nombres consécutifs de la suite on obtient une approximation du nombre d'or $\frac{1+\sqrt{5}}{2}$. Plus les nombres sont avancés dans la suite, meilleure est l'approximation.

— Ecrire un programme qui affiche la suite de Fibonacci jusqu'à 200.

Indice : Initialiser deux variables `a` et `b` qui représentent le dernier et l'avant-dernier nombre de la suite. Dans une boucle `while`, calculer le nouveau dernier et le nouvel avant-dernier nombre de la suite et les mettre dans `a` et `b`. La boucle continue tant que le dernier nombre est plus petit que 200.

— Modifier le programme écrit en a) pour vérifier que la ratio de deux nombres consécutifs de la suite se rapproche du nombre d'or. Le programme doit donc afficher le ratio des nombres consécutifs de la suite de Fibonacci.

Indice : Utiliser la fonction `sqrt(a)` qui retourne la racine carrée de `a`. Cette fonction se trouve dans le module `math` et peut donc s'utiliser après avoir été importée avec l'instruction `from math import *`

5.5 Les listes en compréhension

6 La définition de fonction

Il est souvent utile d'écrire ses propres fonctions, afin de ne pas avoir à écrire plusieurs fois la même liste d'instruction. En programmation, le concept de fonction n'est pas exactement le même qu'en mathématiques. Il faut plutôt le voir comme un sous-programme auquel on fournit des objets et qui en retourne

d'autres.

Pour définir une fonction, il faut indiquer les éléments suivants :

1. Le nom de la fonction, qui ne doit pas déjà être utilisé dans ce programme.
2. Les arguments, ou variables d'entrées, qui indiquent quels sont les objets à fournir à la fonction pour que le programme puisse l'exécuter.
3. La liste des instructions de la fonction, autrement dit, le sous-programme effectué par la fonction. La liste des instructions est *indentée* par rapport au programme principal, c'est-à-dire qu'elle est décalée à droite. De plus, la liste d'instruction se termine par l'instruction **return**. Une liste d'instruction est aussi appelée un *bloc* d'instruction.
4. Le résultat, ou variable de sortie, qui indique quel objet est retourné par la fonction est donné par la dernière instruction de la liste d'instruction, l'instruction **return**.

Ces quatre éléments constituent la *définition* de la fonction. Une fois une fonction ainsi définie, on peut l'utiliser autant de fois que l'on désire dans un programme. La fonction **print**, est un exemple de fonction que l'on a déjà utilisée. Elle est définie par dans les bibliothèques de base de python.

Exemple 29

```
def aucube(n): 1
    cube = n * n * n 2
    return cube 3
4
a = aucube(n=2) 5
b = aucube(n=5) 6
print("les cubes de 2 et 5 sont", a, "et", b) 7
```

Dans l'exemple ci-dessus le mot-clé **def** indique que l'on définit une fonction. Toute définition de fonction commence avec ce mot-clé. Le mot qui suit est le nom de la fonction (ici **aucube**) suivi d'une paire de parenthèses indiquant les arguments de la fonction. La fonction de cet exemple n'a qu'un argument, appelé **n**. La première ligne de la définition de la fonction se termine par deux points (:), indiquant que la liste d'instructions va commencer. La fonction se termine avec l'instruction **return** qui indique ce que la fonction retourne.

Après avoir définie cette fonction, le programme ci-dessus l'utilise dans les lignes 5 et 6. Ce n'est qu'à ce moment-là que les instructions de la fonction sont exécutées. La première fois (ligne 5), la variable **n** prend la valeur 2 (ligne 5) et le résultat de la fonction (ce qui est retourné par l'instruction **return**) est assigné à la variable **a**. La seconde fois (ligne 6), **n** prend la valeur 5 et le résultat de la fonction est assigné à **b**.

Exercice 25 Modifier l'exemple ci-dessus pour avoir une fonction **aucarre** qui calcule le carré d'un nombre. Utiliser cette fonction pour calculer le carré des nombres 6, 18 et 573.

Exemple 30

```
def saluer(nom):
    print("Bonjour", nom)
    print("Bienvenue")
    return

saluer(nom="Pierre")
saluer("Jeanne")
```

L'exemple ci-dessus est montrée la définition d'une fonction qui ne retourne aucune valeur : l'instruction `return` n'est pas suivie par une variable. La fonction s'exécute mais ne retourne rien.

On remarque également que l'appel de fonction de la dernière instruction omet le nom de l'argument (`nom=`) qui est facultatif. En ce cas, il faut respecter l'ordre des arguments dans la fonction, comme dans l'exemple ci-dessous.

Exemple 31

```
def volume_cylindre(rayon, hauteur):
    vol = 3.1415 * rayon**2 * hauteur
    return vol

v1 = volume_cylindre(rayon=2.3, hauteur=10)
v2 = volume_cylindre(1.2,5) ## le rayon=1.2 et la hauteur=5
print("Le volumes des cylindres est de ", v1, "et ", v2)
```

6.1 Visibilité des variables

Un élément à prendre en compte dans l'utilisation de fonction est la visibilité des variables. Lorsqu'une variable est utilisée dans une fonction, elle n'est visible (ou définie) qu'à l'intérieur de cette fonction. Dans l'exemple 29, la variable `cube` n'est pas définie en dehors de la fonction `aucube` et ne peut donc pas être utilisée. Par contre, si l'on souhaite utiliser une variable globale (c'est-à-dire définie dans le script principal, en dehors d'une fonction), alors il faut l'indiquer avec le mot-clé `global`, comme dans l'exemple ci-dessous. Mais en cas la variable globale doit être initialisée avant le premier appel de la fonction.

Exemple 32

```
def saluer(nom):
    global nbpers
    nbpers = nbpers + 1
    print("Bonjour", nom)
    print("Bienvenue")
    return

nbpers = 0
saluer("Pierre")
saluer("Jeanne")
print("J'ai pu saluer ", nbpers, "personnes")
```


Dans l'exemple ci-dessus le programme garde le compte du nombre de personnes saluées dans la variable globale `nbpers` qui est mise à jour à chaque appel de fonction. Grâce au mot-clé `global` de la ligne 2, python va chercher la valeur de cette variable à l'extérieur de la fonction. Elle a été initialisé à la ligne 8, qui est la première ligne exécutée par le programme après la définition de la fonction `saluer`.

Exercice 26 Qu'impriment les deux programmes 33 et 34 suivants? Vérifiez vos réponses.

Exemple 33

```
def fonction(x):
    a = 3
    a = a+1
    return a*x
a = 5
b = fonction(10)
print(b)
b = fonction(10)
print(b)
```

Exemple 34

```
def fonction(x):
    global a
    a = a+1
    return a*x
a = 5
b = fonction(10)
print(b)
b = fonction(10)
print(b)
```

Exercice 27 Reprendre l'exemple 31.

1. Ajouter une variable globale `compte` qui compte le nombre de fois que la fonction `volume_cylindre` est utilisée.
2. Vérifier que votre programme fonctionne en affichant le nombre de fois que cette fonction est appelée.
3. Ajouter une autre variable globale `somme` qui fait la somme de tous les volumes calculés.

Exercice 28 L'indice de masse corporelle (IMC) d'une personne est donnée par son poids (en kg) divisé par le carré de sa taille (en mètres). Ecrire une fonction qui prends le poids et la taille en argument et retourne l'IMC Utiliser cette fonction dans un programme qui demande son poids et sa taille (en mètres) à l'utilisateur et affiche son IMC dans le terminal.

7 Les chaînes de caractères

7.1 Extraction de caractères

Comme les listes, il est possible d'extraire des caractères des chaînes de caractères grâce aux crochets. Les indices commencent également à 0 pour le premier caractères.

Exemple 35

```
nom = 'Julie' 1
prenom = 'Dupont' 2
print ('Bonjour', nom[0], '. ', prenom) 3
```

Comme dans les listes, il est également possible de sélectionner une “tranche” de plusieurs caractères consécutifs en utilisant `s[a:b]` qui retourne les caractères `a` à `b-1` de la chaîne de caractères `s`. Si `a` n’est pas spécifié, on sélectionne depuis le début de la chaîne de caractère, et si `b` n’est pas spécifié on sélectionne jusqu’à la dernière lettre de la chaîne de caractère.

Exemple 36

```
mot = 'Bonjour' 1
a = mot[3:6] 2
print(a) 3
print (mot[:3]) 4
print (mot[1:]) 5
```

Dans l’exemple ci-dessous, la variable `a` contient les caractères 3 à 5 du mot ‘Bonjour’, c’est-à-dire ‘jou’. Toutefois, à la différence des listes, il n’est pas possible en Python de modifier une chaîne de caractère.

Exemple 37

```
l = ['a', 'b', 'c', 'd'] 1
l[1] = 'e' # possible car l est une liste 2
a = 'abcd' 3
## on ne peut pas ecrire a[1] = 'e' 4
## car a est une chaine de caractere. 5
## par contre on peut ecrire 6
a = a[0]+'e'+a[2:] 7
print(a) 8
```

7.2 Quelques fonctions et opérateurs utiles

opérateurs Les opérateurs les plus utiles pour les chaînes de caractères sont sans doute le `+` et le `*`. Le `+` permet de concaténer deux chaînes de caractères, c’est-à-dire de les mettre bout-à-bout, comme dans l’exemple ci-dessus. Le `*` permet de répéter une chaîne de caractères un certain nombre de fois

Exemple 38

```
a = 'sos'*4 1
print(a) 2
```

Dans l’exemple ci-dessus, la variable `a` contient ‘sossossossos’, autrement dit les lettres ‘sos’ répétées quatre fois.

list La fonction `list(a)`, prend la chaîne de caractères `a` et retourne une liste ayant pour éléments tous les caractères de la chaîne.

len La fonction `len(a)` retourne le nombre de lettres contenues dans la chaîne de caractères `a`.

str La fonction `str(a)` crée une chaîne de caractère contenant le contenu de `a`. Cette fonction retourne donc la conversion de `a` en chaîne de caractère. Par exemple `str(5)` retourne la chaîne de caractère "5".

str.find La fonction `str.find(a, str, beg)` détermine si la chaîne de caractères `a` contient le caractère (ou la chaîne de caractères) `str` en commençant la recherche à partir de l'indice `beg`. Si ce n'est pas le cas, la fonction retourne -1, sinon elle retourne l'indice de `a` où `str` commence. Si on ne spécifie pas l'argument `beg`, la recherche se fait à partir du début de la chaîne de caractères.

str.join La fonction `str.join(s, seq)` concatène les chaînes de caractères de la liste `seq` en insérant la chaîne de caractère `s` entre chacune d'elle et retourne la chaîne de caractères résultante.

Exercice 29 Tester les opérateurs et fonctions ci-dessus avec des chaînes de caractères de votre choix

Exercice 30 Ecrire une fonction `efface(mot, lettre)` qui retourne la chaîne de caractère `mot` dans laquelle les occurrences de la lettre `lettre` ont été effacées. Par exemple `efface("coucou", "u")` doit retourner "coco".

Indice : Convertir `mot` en une liste `li` puis modifier la liste et la reconvertir en chaîne de caractères à l'aide de `str.join(",li)`.

Exercice 31 Ecrire une fonction `echange(mot, lettre1, lettre2)` qui échange, dans `mot`, toutes les occurrences de la lettre `lettre1` avec `lettre2` et vice-versa. Par exemple `echange("tare", "r", "t")` doit retourner "rate"

Exercice 32 - Le jeu du pendu Le but de cet exercice est de programmer le jeu du pendu. Les programmes vus jusqu'à présent étaient si simples qu'ils pouvaient facilement s'écrire de façon linéaire, de la première à la dernière ligne. Cependant ce n'est généralement pas ainsi que les programmeurs écrivent des programmes. De même qu'on écrit un plan avant de rédiger un texte, ou qu'un dessinateur commence avec des traits principaux avant de figurer les détails, un programmeur n'écrit généralement pas son programme de la première à la dernière ligne. Ici, nous allons procéder en implémentant d'abord les fonctionnalités principales puis nous allons étoffer le programmes pour le rendre plus complet et amusant.

La programmation de ce jeu est divisée en plusieurs étapes relativement simples. Les variables principales sont deux chaînes de caractères. D'une part la variable `mot` qui contient le mot que l'utilisateur doit deviner, et d'autre part la variable `resultat` qui contient le même nombre de lettre que `mot` mais qui ne montre que les lettres que l'utilisateur a devinées, les autres étant représentées

par des +.

Ainsi, il faut d'abord choisir aléatoirement un mot et le mettre dans la variable `mot`, puis initialiser la variable `resultat` avec une suite de + (autant que de lettres dans `mot`). Il faut ensuite demander à l'utilisateur de deviner une lettre et vérifier si elle se trouve dans `mot`. Si c'est le cas, il faut remplacer le (ou les) + correspondant par cette lettre dans `resultat`. On affiche ensuite `resultat` et on recommence jusqu'à ce que l'utilisateur ai deviné toutes les lettres du mot (ou qu'il ait atteint le nombre maximal d'erreur, auquel cas il est perdu). Pour des questions de lisibilité, dans ce programme nous placerons toutes les instruction `import` en tête de programme, puis les définitions de fonctions et enfin les instructions du programme.

1. Ecrire une fonction `remplace(mot,resultat,lettre)` qui trouve à quelle position dans `mot` le caractère `lettre` apparaît et remplace, dans `resultat`, le caractère à cette position par cette lettre. Par exemple `remplace('chemin','abcdef','i')` devrait retourner `'abcdif'` car le `'i'` se trouve en quatrième position de `'chemin'` et donc on met un `'i'` en quatrième position de `'abcdef'`. (Les positions commencent à 0.).
2. Modifier la fonction `remplace_tous` pour que cela fonctionne également si le caractère `lettre` est présent en plusieurs positions dans `mot`. *Indice* : Une manière de faire est de vérifier toutes les lettre de `mot`, les unes après les autres.
3. Initialiser la variable `mot` avec un mot de votre choix.
4. Initialiser la variable `resultat` avec autant de + qu'il a de lettre dans `mot`. *Indice* : Utiliser les opérateurs et fonctions définies ci-dessus.
5. Afficher un message de bienvenue au jeu.
6. Afficher la variable `resultat`, demander à l'utilisateur de deviner une lettre et utiliser `remplace` pour mettre à jour `resultat` (c'est-à-dire placer la lettre de l'utilisateur dans `resultat` si elle se trouve dans `mot`).
7. Utiliser une boucle `while` pour que les instruction du point précédent se répètent tant que le mot n'a pas été deviné. A la fin de la boucle, si le mot est deviné, féliciter l'utilisateur.
8. Modifier le programme pour permettre un maximum de dix erreurs. Pour ceci créer une variable qui compte les erreurs de l'utilisateurs. Si cette variable arrive à 10 interrompre la boucle, indiquer que le joueur a perdu et afficher le mot qu'il fallait deviner.
9. Ce programme va toujours proposer le même mot à deviner, il faut le modifier pour qu'il choisisse un mot de façon aléatoire.
 - Créer une liste appelée `touslesmots` qui contient tous les mots possibles que l'ordinateur peut proposer. Pour commencer, une liste de 10 mots est amplement suffisante.
 - Définir (en début de programme) une fonction appelée `choisi_mot(li)` qui retourne un élément choisi au hasard dans la liste `li`. Utiliser la fonction `randint(a)` qui choisi un nombre au hasard entre 0 et `a-1`. **Indication** : Mettre la ligne suivante en début de programme pour utiliser `randint` :

```
from random import *
```

- Modifier l'initialisation de la variable `mot` au début du programme pour qu'il choisisse aléatoirement un mot à deviner. Utiliser la fonction `choisi_mot` et la liste `touslesmots`.

10. Modifier le programme pour qu'il fasse le dessin du pendu à chaque étape. Pour ceci, définir une fonction `dessine_pendu(n)` qui dessine les `n` premières étapes du pendu.

Il est possible d'améliorer ce jeux de diverses manières. Par exemple, au lieu de définir soit même une liste de mots, on pourrait dire au programme d'aller en chercher une sur le web (ce qui permettrait à l'utilisateur de choisir sa langue). Ou alors on pourrait créer une interface graphique pour que le jeu se déroule dans une fenêtre dédiée avec un vrai dessin de pendu.

8 Les dictionnaires

En Python, un dictionnaire, aussi appelés "table de hachage" dans d'autres langages, est une structure qui permet stocker des couples d'éléments (en anglais *items*) constitués d'une "clef" (en anglais *key*) et d'une "valeur" (en anglais *value*). Dans l'analogie avec un dictionnaire réel, la "clef" représente le mot et la "valeur", la définition. On ne peut donc accéder à la valeur (la définition) qu'en utilisant la clef (le mot).

Pour initialiser un dictionnaire, on met la liste de couples `clef:valeur` dans une accolade et séparés par une virgule. La clef est séparée de la valeur par deux points. L'exemple ci-dessous utilise un dictionnaire Python pour réaliser un dictionnaire français-anglais.

Exemple 39

```
dico = {"livre": "book", "souris": "mouse", 1  
        "ordinateur": "computer", "voiture": "car"} 2  
print("En anglais, le mot 'souris' se dit", dico["souris"]) 3
```

Dans l'exemple ci-dessus, le dictionnaire `dico` ne peut être utilisé que pour traduire du français vers l'anglais, mais pas pour traduire de l'anglais vers le français car les éléments ne sont accessible que par leur clef (et non par leur valeur). Pour accéder à la valeur correspondant à une clef, on place la clef entre crochets juste après le nom du dictionnaire (`dico["souris"]` dans l'exemple ci-dessus).

Dans cet exemple, la clef est donnée par une chaîne de caractère, mais elle peut aussi être donnée par d'autres types de variable comme par exemple un entier :

Exemple 40

```
chiffres = {0: "zero", 1: "un", 2: "deux", 3: "trois", 4: "quatre", 1  
            5: "cinq", 6: "six", 7: "sept", 8: "huit", 9: "neuf"} 2  
print("En toutes lettres, le chiffre 4 s'écrit", chiffres[4]) 3
```

Il est aussi possible de mettre des listes comme valeurs :

Exemple 41

```
notes = {"Pierre": [5, 4.5], "Jean": [5,6,3.5], "Anna": [6,5,5]} 1
print("Les notes de Jean sont:", notes["Jean"]) 2
```

Il est bien entendu possible d'ajouter ou de modifier une valeur dans un dictionnaire avec l'opérateur =.

Exemple 42

```
moyennes = {"Laure": 5, "Pierre": 4.5} 1
moyennes["Mia"] = 5.5 2
moyennes["Pierre"] = 4 3
print("La moyenne de Pierre est", moyennes["Pierre"]) 4
print("La moyennen de Mia est", moyennes["Mia"]) 5
```

Exercice 33 Ecrire un programme qui demande un nombre à l'utilisateur d'entrer un nombre entier n et initialise un dictionnaire avec comme clef les nombres de 1 à n et comme valeur la liste des diviseurs de la clef.

8.1 Muabilité

Tout comme les listes, les dictionnaires en Python sont muables (*mutable* en anglais), contrairement aux variable de type simple (nombres, chaînes de caractères, booléens) et aux tuples. Cela signifie concrètement qu'un dictionnaire ou une liste qui sont passé comme argument d'une fonction peuvent être modifiés par la fonction comme dans l'exemple ci-dessous.

Exemple 43

```
def ajoute(dic,nom,age): 1
    age = int(age) # enleve la virgule a age 2
    dic[nom] = age 3
    return 4
5
dico = {} 6
a = 22.5 7
ajoute(dico,"Joelle", a) 8
a = 19.5 9
ajoute(dico,"Mike", a) 10
print("Le dictionnaire muable dico:", dico) 11
print("La variable immuable a:", a) 12
```

Dans l'exemple ci-dessus, le dictionnaire `dico` est modifié par la fonction `ajoute`, alors que la variable `a` qui est un nombre ne l'est pas. Ainsi, à la fin du programme, le dictionnaire contient les éléments `{"Joelle":22, "Mike": 19}`, alors que la variable `a` contient le nombre 19.5.

Les listes sont également muables en Python, mais il existe également des listes immuables qui qu'appellent des tuples. On les utilisent lorsqu'on est sûr de ne pas vouloir les modifier. Les tuples sont initialisés en mettant les éléments séparés par des virgules dans une parenthèse.

Exemple 44

```
premiers = (2,3,5,7,11) 1
print("le plus petit nombre premier est", premiers[0]) 2
print("les trois premiers nombres premiers sont", premiers[0:3]) 3
```

Dans l'exemple ci-dessus, `premier` est un tuple car il est initialisé avec des parenthèses à la place des crochets comme on le ferait pour une liste. Comme c'est un tuple, il est immuable. L'instruction `premier[2]=4` générerait une erreur puisque l'on essaierait de modifier ce tuple. Par contre, il est possible d'accéder (sans les modifier) aux éléments du tuple, comme pour les listes.

Exercice 34 Déterminer ce qu'affiche le programme suivant et vérifier en exécutant le code.

Exemple 45

```
def f1(a,b): 1
    b = b+1 2
    a[0] = b 3
    return 4
5
x = 3 6
z = [2, 4] 7
y = ["b", "o", "n"] 8
u = {"k": "b"} 9
f1(z,x) 10
f1(y,z[1]) 11
f1(u,x) 12
print(x,y,z,u) 13
```

8.2 Quelques fonctions et opérateur utiles

L'opérateur `in` L'instruction `clef in dico` retourne `True` si `dico` contient la clef `clef`, `False` sinon.

L'opérateur `not in` L'instruction `clef not in dico` retourne `False` si `dico` contient la clef `clef`, `True` sinon.

`del dico[clef]` Efface l'élément `clef` du dictionnaire `dico`.

`length(dico)` Retourne la longueur du dictionnaire, c'est-à-dire le nombre de clefs et de valeurs qu'il contient.

`dict.clear(dico)` Vide le dictionnaire `dico`.

`dict.get(dico,clef)` Si celle-ci existe, retourne la valeur correspondant à la clef `clef` dans le dictionnaire `dico`, sinon retourne `None`.

`dict.keys(dico)` Retourne la liste des clefs du dictionnaire `dico`. Les clefs sont retournées dans un ordre aléatoire.

`dict.values(dico)` Retourne la liste des valeurs du dictionnaire `dico`.

`dict.items(dico)` Retourne une liste de tuples contenant la clef et la valeur correspondante. Un tuple est simplement une paire d'éléments entre parenthèses.

`dict.popitem(dico)` Ote (sans ordre particulier) une clef et sa valeur correspondante du dictionnaire et les retourne dans un tuple.

Remarque : Selon une syntaxe qui sera vue dans le chapitre 9, il est également possible d'appeler les fonction ci-dessus de la manière suivante : `dico.clear()`, `dico.get(clef)`, `dico.keys()`, `dico.values()`, `dico.items()` où `dico` est le nom de la variable de type dictionnaire.

Exercice 35 Initialiser un dictionnaire et tester les fonctions ci-dessus.

Il est courant de parcourir tous les éléments d'un dictionnaire de la manière suivante.

Exemple 46

```
capitales = {"Suisse":"Berne", "France":"Paris", "USA": "1  
Washington", "Italie":"Rome"}  
for (pays,cap) in dict.items(capitales): 2  
    print("La capitale de", pays, "est", cap) 3
```

Dans cet exemple, la boucle `for` se fait sur un tuple `(pays, cap)`, c'est à dire sur deux variables simultanément. A chaque itération de la boucle, `pays` prend la valeur de la clef et `cap` prend la valeur correspondante.

Exercice 36 Dans l'exemple ci-dessus, le déterminant du pays manque dans la phrase affichée ("La capitale de Suisse" au lieu de "la capitale de *la* Suisse"). A l'aide d'un dictionnaire supplémentaire, modifier cet exemple pour que la phrase affichée contienne le déterminant adéquat.

Dans l'exemple suivant, un dictionnaire est utilisé pour stocker les débuts et fins de mandat des président américains. Une boucle `while` est ensuite utilisée dans laquelle le programme demande à l'utilisateur d'entrer le nom d'un président et indique le début et la fin de son mandat.

Exemple 47

```
presidents = {"J. Carter":[1977,1980], "R. Reagan":[1981,1988], "G. 1  
H.W. Bush":[1989,1992], "B. Clinton":[1993,2000],  
"G.W. Bush":[2001,2007], "B. Obama":[2008,2016], "D. Trump" 2  
:[2017,2018]}  
while True: 3  
    pres = raw_input("Quel president vous interesse? ") 4
```



```

if pres=="":
    break
if pres in dict.keys(presidents):
    annees = presidents[pres]
    print(pres, "etait president entre", annees[0], "et",
          annees[1])
else:
    print("Je ne connais pas ce president.")

```

Exercice 37 Que se passe-t-il si l'utilisateur entre uniquement le nom de famille du président (par exemple "Clinton")? Modifier le programme pour qu'il puisse gérer ce cas de figure, par exemple en utilisant des fonctions vues au chapitre précédent (chaînes de caractères).

Exercice 38 Modifier le programme ci-dessus (sans changer le dictionnaire) pour qu'il demande à l'utilisateur d'entrer une année et affiche quel était le président cette année-là.

Exercice 39 Ecrire trois fonctions qui vident un dictionnaire, une qui utilise la fonction `dict.clear`, une qui utilise l'opérateur `del` et une qui utilise la fonction `popitem`.

Exercice 40 - Memory Ecrire un programme qui propose une version simple du jeu du memory pour un joueur. Le jeu contient 20 "cartes" représentant 20 objets (dans notre cas des nombres entiers de 0 à 9) de telle sorte que chaque objet (ou nombre) est représenté exactement sur deux cartes.

Au début du jeu, les cartes sont disposées face cachées et le joueur doit tenter de retrouver une paire en retournant deux cartes de son choix. Si les deux cartes représentent le même objet, alors la paire est sortie du jeu. Sinon elles sont retournées à nouveau (face cachée), et le joueur peut tenter de retourner une autre paire de cartes. Le but du joueur est de former toutes les paires avec le moins de tentatives possibles.

Il s'agit de programmer les éléments suivants :

1. une phase d'initialisation durant laquelle la position des cartes est aléatoirement distribuée parmi les 20 positions, afin que la position initiale change lorsqu'on relance le programme.
2. une phase de jeu au cours de laquelle à chaque tour, le joueur choisit les deux cartes qu'il retourne et le programme lui indique si les deux cartes forment une paire et affiche alors le nouveau plateau de jeu en utilisant une fonction `affiche` à définir plus tard.
3. A la fin du jeu, lorsque toutes les cartes sont découvertes, le programme doit indiquer au joueur combien d'essais non infructueux il a fait (c'est-à-dire le nombre de fois où les deux cartes qu'il a retournées ne faisaient pas la paire.).
4. la fonction `affiche` mentionnée ci-dessus qui affiche le plateau de jeu en indiquant clairement quelle sont les cartes à la face cachée (par exemple

avec un "X"), lesquelles sont sorties du jeu (par exemple avec un "-") et lesquelles sont apparentes (avec le nombre correspondant à la carte).

Indices :

1. Vous pouvez utiliser un dictionnaire dont chaque élément représente une carte du plateau de jeu (sa position et son contenu).
2. Vous pouvez utiliser les fonctions `randint` ou `shuffle(li)` du module `random` qui mélange les éléments de la liste `li`, c'est-à-dire qu'elle change de façon aléatoire l'ordre des éléments de cette liste.

9 Les objets

Les programmes vus jusqu'à maintenant manipulent des variables et définissent des fonctions. Cette manière de programmer est appelée la programmation procédurale puisque le code est structuré par des procédures (c'est-à-dire des fonctions). Une manière alternative de structurer un programme consiste à le modulariser par ce qu'on appelle des objets. Il s'agit de la programmation orientée objet. Certains langages (comme C) ne permettent pas la programmation orientée objet, d'autres (comme Java) l'imposent, et d'autres encore (comme Python) la permettent sans l'imposer.

9.1 Définition d'un objet

Comme dans la réalité, en programmation un objet se définit par des attributs et par ses fonctionnalités. Par exemple, dans la réalité pour définir une chaise avec ses attributs (quatre pieds, une assise et un dossier) et sa fonctionnalité (on peut s'asseoir dessus). En programmation, les attributs d'un objet sont des variables ou d'autres objets, alors que les fonctionnalités sont des fonctions (appelée dans ce cas des méthodes). On peut ainsi par exemple définir un compte bancaire ainsi :

Exemple 48

```
class Compte:
    solde = 0

    def imprimer_solde(self):
        print("Le solde est",self.solde)

    def ajouter(self,montant):
        self.solde = self.solde + montant

    def retirer(self,montant):
        self.solde = self.solde - montant
```

Dans l'exemple ci-dessus, la première ligne indique qu'on va définir un nouveau type d'objet qui s'appelle `Compte`, ou plus précisément une nouvelle *classe* d'objet appelée `Compte`. A la seconde ligne, on indique que ce type d'objet a un attribut appelé `solde` qui vaut initialement zéro. On définit ensuite trois fonctions (ou méthodes) : `imprimer_solde` affiche le solde du compte, `ajouter`

pour mettre de l'argent sur le compte et `retirer` pour retirer de l'argent du compte. La première méthode `imprimer_solde` ne comporte qu'un argument `self`, c'est-à-dire le compte en question. Les deux autres méthodes possèdent un autre argument, `montant` qui est le montant à ajouter ou retirer. De manière générale, toutes les méthodes ont l'argument `self` qui représente l'objet sur lequel s'applique la méthode. Une méthode peut accéder aux attributs de son objet grâce à l'opérateur "." (point). Ainsi `self.solde` représente l'attribut `solde` de l'objet représenté par `self`.

Une fois une classe définie il est possible de l'utiliser, par exemple ainsi :

Exemple 49

```
compte1 = Compte() 1
compte1.ajouter(100) 2
compte1.imprimer() 3
compte1.retirer(50) 4
compte1.imprimer() 5
6
compte2 = Compte() 7
compte2.ajouter(60) 8
compte2.imprimer() 9
compte2.retirer(30) 10
compte2.imprimer() 11
```

Dans l'exemple ci-dessus deux objets de classe `Compte` ont été créés, l'un appelé `compte1` et l'autre appelé `compte2`. On parle alors de deux *instances* de la classe `Compte`. La création d'un objet d'une classe donnée (on parle alors de l'instanciation d'un objet) se fait par une fonction qui a le même nom que la classe (cf. lignes 1 et 7 de l'exemple ci-dessus). Cette fonction, qui construit un objet d'une classe donnée est appelée un *constructeur*. Elle peut être définie implicitement comme dans l'exemple ci-dessus ou explicitement avec la fonction `__init__` comme dans l'exemple suivant :

Exemple 50

```
class Compte: 1
    solde = 0 2
3
    def __init__(self): 4
        print("Le compte est en voie de creation") 5
6
    def afficher_solde(self): 7
        print("Le solde est",self.solde) 8
9
    def ajouter(self,montant): 10
        self.solde = self.solde + montant 11
12
    def retirer(self,montant): 13
        self.solde = self.solde - montant 14
```

Avec l'exemple ci-dessus, chaque fois qu'un `Compte` sera ouvert avec l'appel de la fonction `Compte()`, c'est la méthode `__init__` qui sera exécutée.

Note : Dans l'exemple ci-dessus l'entier de la définition de la classe a été reproduit, bien que seul le constructeur nous intéresse afin d'avoir un exemple complet et de bien montrer le contexte. A partir maintenant, pour rester focalisé, seuls les éléments pertinents du programme seront indiqués dans les exemples et le contexte sera indiqué en commentaire du programme comme dans l'exemple suivant.

Le constructeur peut aussi prendre des arguments supplémentaires, comme dans l'exemple ci-dessous.

Exemple 51

```
class Compte:
    solde = 0

    def __init__(self, solde_initial):
        self.solde = solde_initial
        print("Le compte est en voie de creation")
        print("Solde initial:", self.solde)

    ## definition des methodes de la classe (omis) ##

compte1 = Compte(100)
compte1.retirer(20)
compte1.afficher_solde()
```

Exercice 41 Cet exercice reprend la classe `Compte` définie ci-dessus :

1. Définir la classe `Compte` et instancier trois comptes ayant respectivement, 100, 200, et 500 à leur solde.
2. Ajouter à la classe `Compte` une méthode `vider` qui vide le compte et retourne la valeur du solde.
3. Verser tout l'argent du premier et du deuxième compte sur le troisième compte, et afficher le solde des trois comptes.

Exercice 42 Définir une classe `Personne` ayant comme attributs le nom, le prénom ainsi que l'année de naissance. Ajouter une méthode `age()` qui calcule l'âge de cette personne le 31 décembre de cette année.

Indice : L'année en cours peut être obtenue avec l'instruction `date.today().year` du module `datetime`.

9.1.1 Variable de classe et variable d'instance

Dans les exemples ci-dessus, chaque compte a son propre solde, c'est-à-dire que chaque instance de compte a sa propre variable `solde` qui a une valeur potentiellement différente des autres comptes. C'est ce qu'on appelle une *variable d'instance*. Mais on peut aussi avoir des *variables de classe* qui sont communes à toutes les instances d'une classe donnée. Par exemple la variable `monnaie` dans l'exemple suivant :

Exemple 52

```
class Compte(object):
    solde = 0
    monnaie = "CHF"

    def __init__(self, solde_initial):
        self.solde = solde_initial
        print("Le compte est en voie de creation")
        print("Solde initial:", self.solde, Compte.monnaie)

    def afficher_solde(self):
        print("Le solde est",self.solde, Compte.monnaie)

    ## definition des methodes de la classe (omis) ###

compte1 = Compte(100)
compte2 = Compte(500)
compte1.afficher_solde()
compte2.afficher_solde()
Compte.monnaie = "EUR"
compte1.afficher_solde()
compte2.afficher_solde()
```

Dans cet exemple, la variable `monnaie` est la même pour tous les comptes instanciés, c'est à dire pour `compte1` et `compte2`, contrairement à la variable `solde`. La variable `monnaie` n'est jamais associée à une instance spécifique (à un `self`), mais à toute la classe donc on l'associe au nom de la classe (ici `Compte.monnaie`) comme aux lignes 8, 11, et 19.

Exercice 43 Cet exercice reprend la classe `Compte` définie ci-dessus.

1. Ajouter une variable de classe `nombre` qui compte le nombre de comptes instanciés. Autrement dit, à chaque fois que le constructeur est appelé, cette variable doit être incrémentés. Le mieux est donc de l'incrémenter à l'intérieur du constructeur.
2. Ajouter une variable d'instance `numero` qui donne un nouveau numéro de compte à chaque compte créé. Le plus simple de donner les numéros dans l'ordre, donc d'utiliser la variable `nombre`.

9.2 Héritage

Il est souvent utile de définir une nouvelle classe à partir d'une classe existante en y ajouter des attributs et des méthodes. Ici, nous allons définir un compte spécial, qui a une limite de découvert, c'est-à-dire dont le solde ne peut être inférieur à une valeur (négative) donnée. Il s'agit donc d'un `Compte` qui à en plus un attribut `limite_decouvert`. Pour ne pas avoir à redéfinir toutes la classe, on utilise le principe d'*héritage* en définissant une *sous-classe* de `Compte` appelée `CompteLimite`.

Exemple 53

```
## definition de la classe Compte (omise) ###
class CompteLimite(Compte):
    limite_decouvert = 0

    def fixer_limite(self, limite):
        self.limite_decouvert = limite

    def retirer(self, montant):
        if self.solde - montant >= self.limite_decouvert :
            self.solde = self.solde - montant
        else:
            print("Il n'y a pas assez sur le compte.")

c1 = Compte(60)
c1.retirer(100)
c2 = CompteLimite(60)
c2.retirer(100)
c1.afficher_solde()
c2.afficher_solde()
```

La première ligne après le commentaire de l'exemple ci-dessus indique de la classe `CompteLimite` est une classe qui hérite de tous les attributs et méthodes de la classe `Compte` (indiquée entre parenthèse). La définition de cette sous-classe ne comporte que les éléments qu'on y ajoute (l'attribut `limite_decouvert` et la méthode `fixer_limite`) ainsi que les éléments qu'on modifie (ici la méthode `retirer` qui vérifie maintenant que la limite n'est pas dépassée). A l'exécution de ce programme, l'objet `c1` de classe `Compte` affichera un solde de -40 alors que l'objet `c2` de classe `CompteLimite` aura affichera un solde de 60, car cette classe n'autorise pas le découvert.

Exercice 44 Définir une classe `CompteNom` qui hérite de la classe `CompteLimite` et qui contient le nom et le prénom de détenteur du compte. Faire en sorte que son constructeur prenne le nom et le prénom en arguments et que la méthode `affiche_solde` de cette classe affiche également le nom et le prénom du détenteur du compte.

10 Les interfaces graphiques avec tkinter

Jusqu'à maintenant, tous les programmes que nous avons vu interagissent avec l'utilisateur par le biais de la console (ou du shell de Python). Bien que beaucoup de programmes utilisés par les professionnels sont de ce type, la plupart des programmes destinés au grand public proposent une *interface graphique*, c'est-à-dire une fenêtre contenant des éléments graphiques avec lesquels l'utilisateur peut interagir. Dans ce chapitre, nous allons voir comment créer une interface graphique avec Python en utilisant la bibliothèque la plus utilisée pour ceci `tkinter`. Cette bibliothèque (ou module, c'est-à-dire un ensemble

de fonctions et de classes déjà définies) permet de faire beaucoup de chose différentes, mais nous n'en utiliserons qu'une petite partie. Pour entrer plus dans les détails, il est conseillé de se référer à la documentation, disponible sur <https://docs.python.org/fr/3/library/tkinter.html>.

10.1 Eléments de base

L'interface graphique la plus simple consiste en une fenêtre dans laquelle il ne se passe rien :

Exemple 54

```
from tkinter import *
fenetre = Tk()
```

Il est possible de donner un titre et d'ajouter des éléments à cette fenêtre. Par exemple du texte, un bouton ou un canevas, c'est-à-dire un espace dans lequel on peut dessiner des formes.

Exemple 55

```
from tkinter import *
fenetre = Tk()
fenetre.title("Le titre de ma fenetre")
etiquette = Label(fenetre, text="Une etiquette")
bouton = Button(fenetre, text="Quitter", command=quit)
toile = Canvas(fenetre, width=500, height=500, bg="white")
etiquette.pack()
bouton.pack()
toile.pack()
mainloop()
```

Lorsqu'on crée un élément, on indique dans quelle fenêtre il se trouve (dans cette exemple `fenetre` puis il est possible d'entrer divers arguments qui vont influencer son apparence. L'argument `command` de `Button` permet d'indiquer quelle fonction va être appelée lorsque l'utilisateur appuie sur le bouton. Cela peut être une fonction du module (comme ici `quit()` qui ferme la fenêtre) ou une fonction écrite par le programmeur. Chaque élément créé est ensuite disposé dans la fenêtre avec la fonction `pack` qui calcule où mettre quoi. Si l'on veut soi-même décider de la disposition, on peut à la place utiliser la fonction `grid`.

La fonction `mainloop()` se met en fin de programme et indique le programme garde la fenêtre ouverte et se met à l'écoute des actions de l'utilisateur.

Exercice 45 Créer un programme qui ouvre deux fenêtres, une dont avec fond rouge appelée "fenetre rouge" et l'autre avec fond bleu appelée "fenêtre bleue". En utilisant des objet de classe `Label`, ajouter un texte dans chaque fenêtre ainsi qu'un bouton permettant de quitter. Est-ce que le bouton ferme toutes les deux fenêtre ou juste celle dans laquelle on a cliqué ?

10.2 Dessiner sur un Canvas

Un `Canvas` est une surface sur laquelle il est possible de dessiner des formes géométriques, telles que des lignes, rectangles, ovals et polygones.

Exemple 56

```
from tkinter import * 1
fenetre = Tk() 2
toile = Canvas(fenetre,width=500, height=500,bg="white") 3
toile.pack() 4
## rectangle allant du point (0,10) au point (30,40) 5
carre = toile.create_rectangle(0,10,30,40,fill="red") 6
## ligne allant du point (50,60) au point (100,110) d'epaisseur 3 7
ligne = toile.create_line(50,60,100,110,fill="blue",width=3) 8
## ovale inclu dans un rectangle allant de (50,60) a (100,110) 9
ovale = toile.create_oval(200,220,260,280,fill = "green",width=2) 10
## polygone defini par les points (40,200), (60,220), (80,300), 11
(60,340).
forme = toile.create_polygon(40,200,60,220,80,300,60,340,fill=" 12
yellow")
mainloop() 13
```

Dans l'exemple ci-dessus les nombres indiqués correspondent aux coordonnées du `Canvas`. L'origine, c'est-à-dire le point (0,0) correspond au coin en haut à gauche. La première coordonnée indique la position sur l'axe horizontal et la seconde coordonnée la position sur l'axe vertical en descendant (C'est comme en mathématique sauf que l'axe des y pointe en bas).

Exercice 46 Ecrire un programme qui ouvre une fenêtre sur laquelle est dessinée une maison.

Il est également possible d'inclure des images au format gif dans un `Canvas`. Dans l'exemple ci-dessus un fichier nommé `arbre.gif` doit être dans le même répertoire que le fichier contenant le programme python.

Exemple 57

```
from tkinter import * 1
fenetre = Tk() 2
toile = Canvas(fenetre,width=500, height=500,bg="white") 3
toile.pack() 4
## l'image se trouve dans le fichier arbre.gif 5
monimage = PhotoImage(file="arbre.gif") 6
## on affiche l'image centree sur le point (250,250) 7
arbre = toile.create_image(250,250,image=monimage) 8
mainloop() 9
```

Dans l'exemple ci-dessus, l'image s'affichera dans sa taille d'origine. Si l'on désire modifier cette taille, il faut modifier le fichier `arbre.gif` avec un programme tel que Gimp ou Photoshop. L'autre solution, plus pratique, consiste à utiliser le module PIL qui contient différentes fonction pour modifier des images comme dans l'exemple ci-dessous.

Exemple 58

```
from tkinter import *
## on va utiliser le module PIL
from PIL import Image, ImageTk
fenetre = Tk()
toile = Canvas(fenetre,width=500, height=500,bg="white")
toile.pack()
## on ouvre un fichier image appele arbre.gif
monimagePIL = Image.open("arbre.gif")
## on change la taille de l'image 1 100 x 100
monimagePIL = monimagePIL.resize((100,100),Image.ANTIALIAS)
## on convertit en format affichable dans un Canvas
monimage = ImageTk.PhotoImage(monimagePIL)
## on affiche l'image centree sur le point (250,250)
arbre = toile.create_image(250,250,image=monimage)
mainloop()
```

Exercice 47 Chercher une image au format gif sur le web et écrire un programme qui affiche cette image dans une fenêtre en trois position différentes.

Il est souvent utile d'obtenir les coordonnées d'un objet dessiné en utilisant la méthode `coords` :

Exemple 59

```
from tkinter import *
fenetre = Tk()
toile = Canvas(fenetre,width=500, height=500,bg="white")
toile.pack()
rond = toile.create_oval(200,220,260,280,fill = "pink",width=2)
pos = toile.coords(rond)
print("la position du rond est ", pos)
```

10.3 Capturer les actions de l'utilisateur

Il est également possible de réagir aux actions de l'utilisateur grâce à la fonction `bind`.

```
from tkinter import *
def pissetouche(event):
    print("Touche pressee", event.char)
    return

def bougesouris(event):
    print("souris bougee au point", event.x, event.x)
    return

def cliquegauche(event):
    print("souris cliquee au point", event.x, event.x)
    return
```

```

fenetre = Tk()
fenetre.bind("<Key>",pressetouche)
fenetre.bind("<Motion>",bougesouris)
fenetre.bind("<Button-1>", cliquEGAuche)

```

Chaque fois que l'utilisateur exécute une de ces actions (appuie sur une touche, bouge la souris ou appuie sur le bouton gauche de la souris) la fonction correspondante est appelée. Ceci se fait de manière asynchrone, c'est ce qu'on appelle des *événements*. Les principales actions détectables sont résumées dans le tableau suivant :

| Code | Événement |
|---|---|
| <Button> | l'utilisateur clique sur un bouton de la souris |
| <Button-1> | l'utilisateur clique sur le bouton gauche de la souris |
| <Button-2> | l'utilisateur clique sur le bouton du milieu de la souris |
| <Button-3> | l'utilisateur clique sur le bouton du droit de la souris |
| <ButtonRelease> | l'utilisateur relâche un bouton de la souris |
| <Double-Button> | l'utilisateur double-clique sur le bouton de la souris |
| <Key> | l'utilisateur appuie sur une touche |
| a | l'utilisateur appuie sur la touche a (cela fonctionne aussi pour les autres caractères) |
| <space> <Return> <Left> <Up> <Right> <Down> | l'utilisateur appuie sur la touche indiquée |
| <Enter> | la souris arrive sur la fenêtre |
| <Leave> | la souris quitte la fenêtre |

Exercice 48 Créer un programme qui ouvre une fenêtre contenant une étiquette et qui met l'arrière-plan en vert si on passe dessus avec la souris et en rouge sinon.

Indices : Utiliser les événements <Enter> et <Leave>. La couleur de l'arrière-plan peut être mise en rouge de la manière suivante : `etiquette.configure(background="red")` (où `etiquette` est le nom de votre étiquette).

Il est possible de modifier un dessin sur un **Canvas** en fonction de l'action de l'utilisateur. Pour ceci on peut utiliser les méthodes (de la classe **Canvas**) `move` (qui déplace un objet dessiné) `delete`, qui efface un objet dessiné.

Exemple 60

```

from tkinter import *
1
2
def gauche(event):
3
4     ## bouge rond de -10 pixels horizontalement et 0 verticalement
    toile.move(rond,-10,0)
5
6     return
7
def droite(event):
8
9     ## bouge rond de 10 pixels horizontalement et 0 verticalement
    toile.move(rond,10,0)
10

```

```

    return 11
def efface(event): 12
    toile.delete(rond) 13
    return 14
15
fenetre = Tk() 16
toile = Canvas(fenetre,width=500, height=500,bg="white") 17
toile.pack() 18
rond = toile.create_oval(200,220,260,280,fill = "pink",width=2) 19
## appeler la fonction gauche lorsqu'on appuie sur <Left> 20
fenetre.bind("<Left>",gauche) 21
## appeler la fonction droite lorsqu'on appuie sur <Right> 22
fenetre.bind("<Right>",droite) 23
## appeler la fonction efface lorsqu'on appuie sur <Return> 24
fenetre.bind("<Return>",efface) 25
mainloop() 26
27

```

Dans l'exemple ci-dessus, le rond bouge à droite lorsque l'utilisateur appuie sur la flèche droite, à gauche lorsque l'utilisateur appuie sur la flèche gauche et est effacé lorsque l'utilisateur appuie sur la touche Return.

Exercice 49 Ecrire un programme qui affiche une fenêtre sur laquelle est dessinée un rectangle qui bouge en haut et en bas lorsque l'utilisateur appuie (sur le clavier) sur la flèche qui monte ou qui descend.

10.4 Créer une animation

Afin de créer une animation, il faut décomposer chaque mouvement en une succession de petits pas qui sont exécutés l'un après l'autre. La librairie `tkinter` inclut un ordonnanceur (*scheduler* en anglais) à qui l'on peut spécifier quels sont les pas à effectuer et à quelle fréquence. Ainsi l'exemple ci-dessous fait bouger un carré rouge sur un `Canvas`.

Exemple 61

```

from tkinter import * 1
2
def unpas(): 3
    toile.move(carre,10,0) 4
    toile.after(10,unpas) 5
    return 6
7
fenetre = Tk() 8
toile = Canvas(fenetre,width=500, height=500,bg="white") 9
toile.pack() 10
carre = toile.create_rectangle(0,0,20,20,fill="red") 11
unpas() 12
mainloop() 13

```

Dans l'exemple ci-dessus, la fonction `unpas` fait bouger le carré rouge. Mais elle ne le fait avancer que de 10 pixels vers la droite. Mais, elle est rappelée toute les 10 millisecondes grâce à l'utilisation de la fonction `after`. Cette fonction indique à l'ordonnanceur d'exécuter la fonction `unpas` dans 10 millisecondes. Ainsi chaque fois que cette fonction est appelée, elle prévoit quelle sera réexécutée dans 10 millisecondes. La fonction `mainloop` est une boucle infinie qui vérifie s'il n'y a des actions de la part de l'utilisateur et exécute les actions prévues par l'ordonnanceur. Elle ne s'arrête que lorsqu'on ferme la fenêtre.

Exercice 50 Modifier l'exemple ci-dessus pour que le carré s'arrête lorsqu'il a dépassé la position de 200 pixels.

Exercice 51 - Le jeu du serpent Le but de ce jeu est de programmer le fameux jeu du serpent qui s'allonge en mangeant des fruits mais qui ne doit ni se mordre la queue ni sortir de son terrain.

La programmation de ce jeu se fera en plusieurs étapes qu'il s'agira de vérifier une à une en exécutant le programme.

1. Ecrire un programme qui ouvre une fenêtre.
2. Dans la fenêtre ajouter un `Canvas` appelé `toile` et y dessiner un carré de 10 pixels de côté en utilisant la méthode `Canvas.create_rectangle`. Ce sera la tête du serpent.
3. Ecrire une fonction `bouge` qui fait avancer le carré de 10 pixels. Y ajouter une instruction `toile.after(100,bouge)` qui fait que cette fonction est ré-exécutée toutes les 100 millisecondes.
4. Définir une variable globale `direction` qui indique dans quelle direction avance la tête du serpent ("Left", "Right", "Up" ou "Down") et l'intégrer dans la fonction `bouge`.
5. Permettre à l'utilisateur de modifier la variable `direction` en appuyant sur les flèches du clavier, grâce à la fonction `Tk.bind`. Ceci permet à l'utilisateur de diriger la tête du serpent.
6. Définir la variable globale `serpent` qui est une liste. Au début du jeu, cette liste ne contient que le carré représentant la tête du serpent. Définit la fonction `allonge` qui ajoute un carré à la liste `serpent`.
7. Définir la fonction `avance` qui fait avancer tout le serpent de 10 pixels. Pour ceci chaque carré de la liste `serpent` prend la place de son prédécesseur. La tête, elle, avance toujours dans la direction donnée par la variable `direction`. L'utilisateur peut maintenant diriger l'entier du serpent avec le clavier.
8. Définir la variable globale `fruits` qui est aussi une liste de rectangles. Il s'agit des fruits que le serpent doit attraper. Initialiser cette liste avec deux rectangles dont la position est générée aléatoirement dans le cadre de la fenêtre. Utiliser la fonction `random.randint` pour la génération des nombres aléatoires.
9. Ecrire une fonction `mange` qui vérifie si la tête du serpent est sur un fruit.
10. Modifier votre code pour que le fruit mangé réapparaisse ailleurs dans la fenêtre et pour que le serpent s'allonge lorsqu'il mange un fruit.

11. Ecrire une fonction qui vérifie que le serpent ne se mord pas la queue. Arrêter le jeu si c'est le cas.
12. Ecrire une fonction qui vérifie que la tête du serpent ne sorte pas de la fenêtre. Arrêter le jeu si c'est la cas.
13. Définir une variable globale `score` qui calcule le nombre de fruits mangés et afficher le score dans la fenêtre au moyen d'un Label.
14. Ajouter un message de fin lorsque la partie est finie.

11 Programmer un jeu avec pygame

12 Les graphiques avec matplotlib

13 Le calcul matriciel avec numpy