Informatique de gestion - 11UlF12-C
Introduction à la programmation: Python

Christophe Desagre

May 08, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION À LA PROGRAMMATION

Ce syllabus a été conçu pour des étudiants en gestion de l'entreprise / ingénieur de gestion à l'ICHEC Brussels Management School. L'objectif de ce syllabus est de permettre à un novice en programmation de:

- Comprendre les bases de la programmation
- Appliquer ces principes au langage de programmation Python
- Convertir un algorithme en code Python

A ce titre, il se concentre d'avantage sur des aspects pratiques que sur des aspects purement techniques. La structure du syllabus a également été pensée en fonction de ces objectifs. Ainsi, à la fin de ce syllabus, l'étudiant sera capable de résoudre à l'aide de Python différentes études de cas dont notamment un tableau d'amortissement (comptabilité), un tableau de remboursement d'emprunt (finance), de résoudre le problème d'un voyageur de commerce (logistique), de réaliser une étude statistique, etc.

Définition

Un algorithme est la description des opérations nécessaires pour obtenir un résultat à partir de valeurs d'entrée, les "données". Un programme est un algorithme écrit dans un langage spécifique (Vigouroux, 2018, p. 25).

1.1 Contenu du syllabus

Le syllabus comporte théorie, exemples, et exercices. A la fin de chaque chapitre, des exercices sont proposés. Un code couleur est également utilisé. Lorsqu'il est fait référence à une variable dans le texte, le nom de cette variable apparaîtra comme ceci: variable. Les morceaux de code seront affichés avec un arrière fond gris et les résultats correspondants seront affichés juste en dessous, comme ceci:

```
a = 1
print(a)
```

```
1
```

1.2 Pourquoi Python?

Python est un langage de programmation inventé par en 1991 par Guido Van Rossum. En comparaison avec d'autres langages de programmation, la synthaxe de Python est simple à appréhender, ce qui en fait un très bon choix pour vous initier à la programmation. Sa synthaxe est en effet très proche de l'anglais (https://www.python.org/about/).

Il existe de nombreux autres langages informatiques. Les plus connus/utilisés sont mentionnés sur la photo ci-dessous:

- ASP : ASP ou .NET is a developer platform made up of tools, programming languages, and libraries for building many different types of applications. ASP.NET extends the .NET developer platform with tools and libraries specifically for building web apps. (https://dotnet.microsoft.com/en-us/apps/aspnet)
- C# / C++ : langages de programmation orienté-objet. Ce sont des langages compilés, ce qui les rend, en général plus rapides que les langages interprétés tels que Python.
- HTML (*HyperText Markup Language*) : n'est pas un langage de programmation mais un langage de balisage. Il permet de créer le contenu de site webs. Il fonctionne de pair avec CSS.
- CSS (*Cascading Style Sheets*) : s'occupe de la mise en page et du style des pages web.
- JavaScript est un langage de script conçu également pour les pages web.
- Php / SQL : sont des langages utilisés pour interagir avec des bases de données. Pour plus d'informations sur SQL, RDV l'année prochaine au cours de "Base de données pour la gestion".
- Java est également un langage de programmation orienté objet.

1.3 Hello, world!

Pour (essayer de) vous convaincre de la simplicité, voyons comment afficher le message “Hello, World!” en C++, en Java, et en Python.

En C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello, World!";

    return 0;
}
```

En Java

```java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

En Python

```
print("Hello, World!")
```

1.3.1 Discipline d'intérêt

Ensuite, Python est un langage utilisé dans de nombreuses disciplines liées de près ou de loin à la gestion : intelligence artificielle et *machine learning*, analyse de réseaux (sociaux), analyse textuelle, finance, etc.

- Intelligence artificielle
  - Machine learning
  - Réseaux de neurones
  - Reconnaissance d'images (la technologie qui dévérouille votre téléphone)

Définition

Machine learning is *programming computers to optimize a performance criterion using example data or past experience. We have a model defined up to some parameters, and learning is the execution of a computer program to optimize the parameters of the model using training data or past experience. The model may be predictive to make predictions in the future, or descriptive to gain knowledge from data, or both* (Alpaydin, 2020)

Voici quelques exemples de projets à réaliser en Python :

- Jeux
- Analyse de réseaux sociaux
  - Analyse du contenu de tweets
  - Analyse des interactions entre profils
- Finance
  - Stratégies de trading automatisées
  - Analyse de cours boursiers
  - Gestion d'un portefeuille financier
- Logistique
  - Optimisation d'un trajet à parcourir

1.3.2 Open-source

Finalement, Python est un langage *open-source*, gratuit, et qui fonctionne sur les différents systèmes d'exploitation (Windows, Mac, Linux).

1.4 Installation

Rendez-vous sur le site https://www.python.org/

Cliquez sur la rubrique *Downloads* (Téléchargements).

Suivez ensuite les instructions d'installation. Le processus d'installation se complète en quelques minutes.

L'installation de Python s'accompagne de IDLE (*Integrated Development and Learning Environment*).

L'interface d'IDLE est assez basique. Toutefois, des programmes plus avancés offrent de nombreuses options et possibilités (inutiles) lorsqu'on commence à programmer. Ouvrez le programme IDLE. Vous devriez voir apparaître la fenêtre suivante:

1.5 Premiers pas

Comme mentionné plus haut, il est coutume d'afficher le message "Hello, world!" quand on commence un langage de programmation. Vous pouvez ensuite écrire le code suivant:

Si vous quittez IDLE, ce que vous venez d'écrire sera perdu.

Nous allons créer un fichier que nous pouvons sauvegarder et dans lequel nous écrirons nos lignes de code. Cliquez sur File > New File. Une deuxième fenêtre apparaît. Sauvegardez le fichier sur votre ordinateur (File > Save As).

Ouvrez l'explorateur de fichiers (Windows) / Finder (Mac) et rendez-vous dans le dossier où vous venez de sauvegarder le fichier. Vous devrez trouver un fichier dont l'extension est .py. Une extension permet à l'ordinateur de reconnaître le type de fichier. Un fichier Word possède l'extension .docx, un fichier Excel possède l'extension .xlsx, etc.

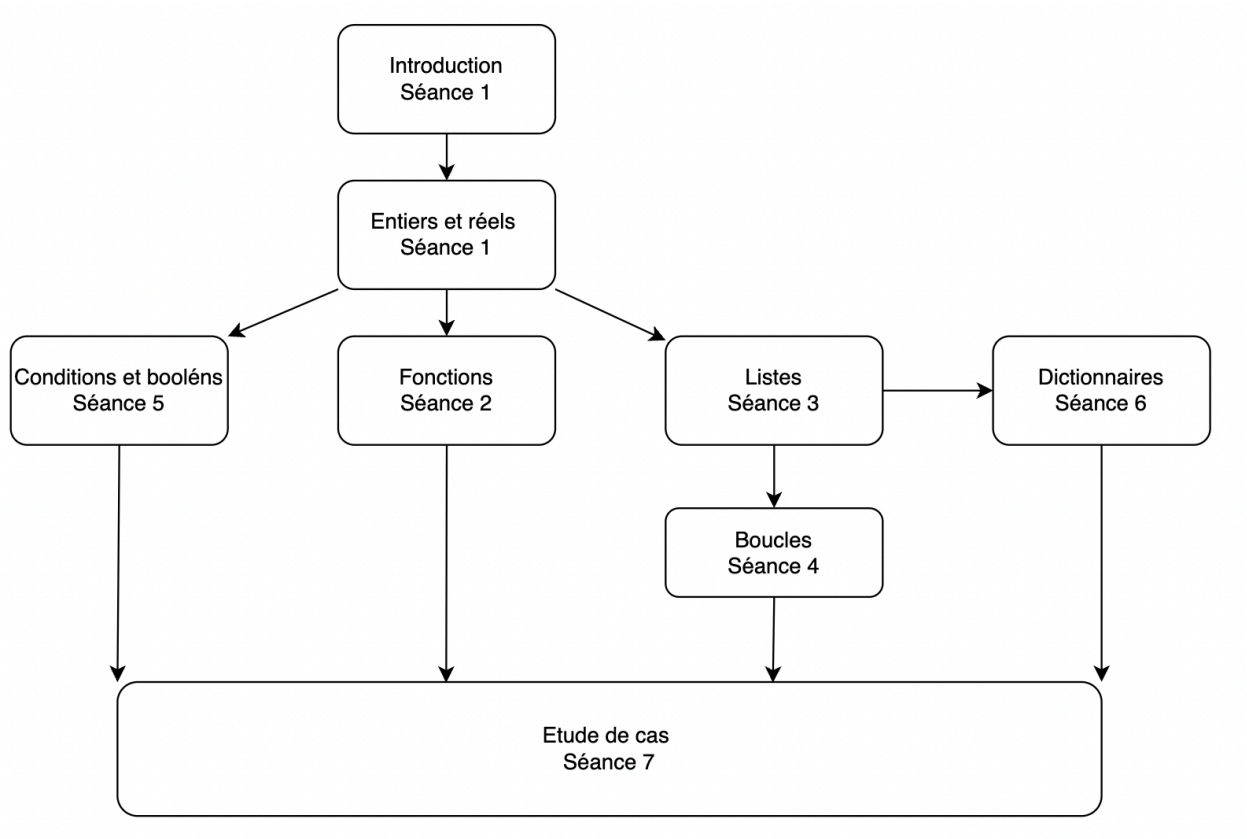
Réécrivez la ligne de code print("bonjour"). Ensuite, appuyez sur Run > Run Module. IDLE vous demande si vous souhaitez enregistrer le fichier. Appuyez sur Yes. Le fichier est sauvegardé et le résultat apparaît presque immédiatement dans la console. Pour exécuter le fichier, vous pouvez également utiliser F5.

Pour plus d'informations sur IDLE, regardez la vidéo ci-dessous !

Il existe bien sûr d'autres éditeurs de texte (SublimText, Notepad++) et même d'environnement de développement (*Integrated Development Environment - IDE*), tels que Visual Studio Code, PyCharm, Tony, etc.

1.6 Organisation du cours

Le cours se déroulera sur 7 séances. Voici le contenu abordé lors de chaque séance :



1.7 Références

Alpaydin, E. (2020). *Introduction to machine learning*, 4ème édition

Vigouroux, C. (2018). *Apprendre à développer avec JavaScript*, 3ème édition

CHAPTER

TWO

ENTIERS ET RÉELS

Il existe de nombreux types de variables en Python. Voici une liste non-exhaustive des types de variables que nous allons parcourir dans ce syllabus:

- Entiers
- Réels
- Booléens
- Chaînes de caractères

Nous verrons également des structures de données plus avancées, c'est-à-dire des variables qui peuvent contenir plus d'informations.

- Listes
- Dictionnaires

Ce chapitre s'intéresse aux types de base en Python: les entiers et les réels.

2.1 Créer une variable

Le but d'une variable est de stocker de l'information. De cette manière, nous pourrons faire référence à cette variable et utiliser son contenu.

- Attribuer la valeur de 1 à a et l'afficher.

```
a = 1
print(a)
```

```
1
```

Warning: Attention, il y a une différence importante entre l'environnement interactif fourni par le *Shell* et l'environnement "non-interactif" d'un fichier .py. Dans un environnement interactif, il suffit d'indiquer le nom de la variable pour voir le contenu de cette variable s'afficher. Ce ne sera pas le cas dans un fichier .py.

```
a = 1
a
```

```
1
```

To-do

Créez un script contenant le code a = 1 ainsi que print(a). Sauvez ce fichier et exécutez-le. A l'aide du *Shell*, faites apparaître la valeur de a. Recommencez ensuite en supprimant l'instruction print(a) et observez les différences.

Le contenu de cette variable est stocké à un endroit précis dans la mémoire de l'ordinateur. On peut connaître cet emplacement à l'aide de la fonction id. Les adresses mémoires sont le plus souvent indiquées sous forme hexadécimale. Pour obtenir ce format hexadécimal, nous utilisons la fonction hex.

```
hex(id(a))
```

```
'0x7fdc7802e930'
```

Cette adresse peut être convertie en adresse binaire à l'aide de la fonction bin.

```
bin(id(a))
```

```
'0b111111111011100011110000000001011101001001100000'
```

2.1.1 Interaction entre variables

- Attribuer la valeur de 1 à a
- Attribuer la valeur de 2 à b
- Afficher leur somme.

```
a = 1
b = 2
print(a + b)
```

```
3
```

On obtient bien le résultat attendu !

- Attribuer la valeur de 1 à a
- Attribuer la valeur de 2 à b
- Créer une variable c qui contient la somme de a et b
- Afficher c.

```
a = 1
b = 2
c = a + b
print(c)
```

```
3
```

Différentes opérations sont possibles: l'addition, la soustraction, la multiplication, la division, la division entière, le reste de la division entière (aussi appelé le modulo) et l'exposant. Voici un script avec a=7 et b=5.

```
a = 7
b = 5

print(a + b)    # addition
print(a - b)    # soustraction
print(a * b)    # multiplication
print(a / b)    # division
print(a // b)   # division entiere
print(a % b)    # modulo
print(a ** b)   # exposant
```

```
12
2
35
1.4
1
2
16807
```

Le résultat de la division de 7 par 5 donne 1.4. C'est un nombre décimal et on utilise le point comme séparateur de décimale (**ne pas utiliser de virgule!**). Un nombre décimal en Python est de type float, en comparaison avec les nombres entiers qui sont de type int, abbréviation de *integer*. Le float provient du fait que les nombres décimaux sont aussi appelés des nombres en virgule flottante. Ceci provient de la façon dont ces nombres sont stockés dans l'ordinateur.

```
a = 1
print(type(a))
```

```
<class 'int'>
```

De même, un nombre décimal est de type float.

```
b = 2.5
print(type(b))
```

```
<class 'float'>
```

Warning: Une erreur fréquente consiste à utiliser une virgule qu'un point pour séparer la partie entière de la partie décimale. Cela ne génère cependant pas une erreur dans la mesure où Python crée un tuple. Plus d'informations sur les tuples plus tard dans le cours.

Voyons maintenant comment afficher l'entièreté du calcul (et pas uniquement le résultat final). Nous allons donc afficher plusieurs éléments dans la fonction print. Ces éléments doivent être séparés par une virgule. Il y a 2 possibilités :

- On fait référence à une variable et on souhaite en afficher le contenu.
- On souhaite simplement afficher du texte. Dans ce cas, le texte doit être entre guillemets ou entre apostrophes.

```
a = 7
b = 5

print(a, "+", b, "=", a + b)    # addition
```

(continues on next page)

(continued from previous page)

```
print(a, "-", b, "=", a - b)    # soustraction
print(a, "*", b, "=", a * b)    # multiplication
print(a, "/", b, "=", a / b)    # division
print(a,"//", b, "=", a// b)   # division entiere
print(a, "%", b, "=", a % b)    # modulo
print(a, "^", b, "=", a** b)   # exposant
```

```
7 + 5 = 12
7 - 5 = 2
7 * 5 = 35
7 / 5 = 1.4
7 // 5 = 1
7 % 5 = 2
7 ^ 5 = 16807
```

2.1.2 Exemple : calcul de TVA

- Créez une variable taux_tva égale à 21%, soit 0.21.
- Créez une variable htva égale à 1,000,000€
- Affichez la valeur de htva
- Calculez le montant de la TVA à payer et stocker le résultat dans tva_a_payer

$$TVA = Taux\_Tva \times HTVA$$

- Affichez tva_a_payer
- Calculez le montant TVA comprise et stocker le résultat dans tvac

$$TVAC = HTVA(1 + Taux\_Tva)$$

- Affichez tvac

```
taux_tva = 0.21
htva = 1000000
print("Chiffre d'affaires:", htva)
tva_a_payer = taux_tva * htva
print("TVA a payer:", tva_a_payer)
tvac = htva * (1+taux_tva)
print("TVAC :", tvac)
```

```
Chiffre d'affaires: 1000000
TVA a payer: 210000.0
TVAC : 1210000.0
```

Remarque

Lorsqu'on multiplie un integer par un float, on obtient un float.

Il est possible de convertir un float en integer à l'aide de la fonction int. La partie décimale sera perdue.

```
a = 1.4
print(a)
a = int(a)
print(a)
```

```
1.4
1
```

Inversément, il est possible de convertir un integer en float.

```
a = 1
print(a)
a = float(a)
print(a)
```

```
1
1.0
```

2.1.3 Séparateurs de milliers

Pour les grands nombres, il est possible (mais pas obligatoire) d'inclure des _ comme séparateurs de milliers. Ce n'est pas obligatoire, mais ça rend la lecture du chiffre plus facile. Ce caractère ne sera toutefois pas affiché.

```
taux_tva = 0.21
htva = 1_000_000
print("Chiffre d'affaires:", htva)
tva_a_payer = taux_tva * htva
print("TVA a payer:", tva_a_payer)
tvac = htva * (1+taux_tva)
print("TVAC :", tvac)
```

```
Chiffre d'affaires: 1000000
TVA a payer: 210000.0
TVAC : 1210000.0
```

2.1.4 Contrôler le nombre de décimales affichées

Pour contrôler le nombre décimales affichées, nous pouvons utiliser la fonction round. Cette fonction a besoin de deux informations: (1) le chiffre à arrondir, et (2) le nombre de décimales souhaitées.

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

```
0.3333333333333333
```

Nous pouvons utiliser la fonction round pour afficher la valeur avec 2 décimales.

```
a = 1 / 3
print(round(a, 2))
```

```
0.33
```

2.1.5 Règles à respecter

Lorsque nous créons une variable, il y a quelques règles à respecter au niveau du nom:

- Le nom d'une variable peut contenir des lettres, des chiffres, ou le caractère _.
- Le nom d'une variable ne peut pas commencer par un chiffre. Ainsi, taux_tva_21 est un nom acceptable, alors que 21_pourcent ne l'est pas.
- Python est sensible à la casse (*case sensitive*), ce qui implique qu'une lettre minuscule est différente d'une lettre majuscule. La variable a ne sera donc pas équivalente à la variable A.
- Le nom d'une variable ne peut donc pas contenir d'apostrophe, raison pour laquelle nous avons créé la variable chiffre_affaires_htva, et pas chiffre_d'affaires_htva.

Ensuite, certains mots-clés (*keywords*) sont réservés; ils ne peuvent pas être utilisés comme nom de variable. Ces mots-clés ont une signification particulière et nous aborderons certains de ces mots-clés dans ce syllabus. Pour connaître la liste des *keywords*, nous pouvons écrire help("keywords").

```
help("keywords")
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               class               from                or
None                continue            global              pass
True                def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
break               for                 not
```

2.2 Les commentaires

Dans le script précédent, l'opération qui est réalisée est indiquée sur la droite. Ce texte est précédé de #. C'est un commentaire.

Note: Un commentaire est un morceau de code qui ne sera pas exécuté.

Les commentaires sont des morceaux de code qui ne sont pas "lus" par le programme. Ils doivent commencer par le sigle #. Ils permettent d'expliquer le code. Ils permettent aussi d'éviter qu'un morceau de code soit exécuté. L'emploi des commentaires est une bonne pratique à adopter lorsqu'on réalise un code.

```
# Auteur: Christophe Desagre
# Cette ligne ne sera pas affichee
print('Cette ligne sera affichee.') # mais pas ce morceau
```

```
Cette ligne sera affichee.
```

2.3 Changer le contenu d'une variable

Comme son nom l'indique, le contenu d'une variable est … variable. Une même variable peut donc contenir différentes valeurs au fur et à mesure de l'exécution du script.

```
a = 5
print("a =", a)
a = 6
print("a =", a)
```

```
a = 5
a = 6
```

Nous pouvons également changer le contenu d'une variable à partir de son contenu actuel. Par exemple, si la variable a contient la valeur 5, on peut ajouter une unité à cette variable pour que le contenu de cette variable devienne 6. Ceci se fait facilement via:

```
a = 5
print("a =", a)
a = a + 1
print("a =", a)
```

```
a = 5
a = 6
```

2.4 Interaction avec l'utilisateur

Il est possible de faire interagir l'ordinateur et l'utilisateur. Dans l'image ci-dessous, l'ordinateur affiche le message "Please enter your name". L'utilisateur répond "John". Ensuite l'ordinateur lui répond "Hello John! And welcome!".

Pour récupérer une valeur fournie par l'utilisateur, il faut utiliser input. Le résultat sera stocké dans la variable prenom

> **Warning:** Si vous exécutez ceci dans IDLE, la question posée à l'utilisateur apparaît dans la console. Il faut y répondre et appuyer sur la touche return.

```
prenom = input("Please enter your name: ")
print("Hello", prenom, "! And welcome!")
```

```
Please enter your name: christophe
Hello christophe ! And welcome!
```

Par défaut, la réponse fournie par l'utilisateur sera stockée sous forme de chaîne de caractères. Si la réponse à la question doit être stockée en tant qu'entier / nombre décimal, il faut utiliser int() / float().

```
age = int(input("Quel age as-tu ? "))
print("Age", age)
```

```
Quel age as-tu ? 20
Age 20
```

```
taille = float(input("Quelle est votre taille (en mètre) ? "))
print("Taille", taille, "m")
```

```
Quelle est votre taille (en mètre) ? 1.80
Taille 1.8 m
```

2.5 Modules

Python est installé avec une série de modules supplémentaires. Ces modules contiennent des fonctionnalités supplémentaires.

2.5.1 Math

Le module math permet d'avoir accès à des fonctions mathématiques. Dans un premier temps, nous devons importer le module.

```
import math
```

Ce module permet notamment d'avoir accès au nombre π, au nombre d'Euler e ou à la fonction logarithmique.

```
p = math.pi
print(p)
```

```
3.141592653589793
```

```
e = math.exp(1)
print(e)
```

```
2.718281828459045
```

Le logarithme en base 10 du chiffre 100 est égal à 2 car:

$$10^2 = 100$$

```
x = math.log(100, 10)
print(x)
```

```
2.0
```

Pour plus d'informations, consultez la documentation du module à l'adresse suivante: https://docs.python.org/3/library/math.html

2.5.2 Random

Le module random permet de générer des chiffres aléatoires. La première instruction est d'importer le module.

```
import random
```

Dans ce module:

- random.randint permet de générer des nombres aléatoires entiers. Lors de l'appel de la fonction, il faut indiquer la borne inférieure et la borne supérieure. Par opposition à range, le chiffre généré sera compris entre la borne inférieure incluse et la borne supérieure incluse également.

```
chiffre = random.randint(1, 10)
print(chiffre)
```

```
4
```

random.choice permet de choisir aléatoirement un élément dans une liste. Pour plus d'informations sur les listes, voir *cette section*

```
cours = ["Excel", "Python", "Fondamentaux"]

random.choice(cours)
```

```
'Fondamentaux'
```

2.5.3 OS

Documentation officielle : https://docs.python.org/3/library/os.html

OS = *operating system*

```
import os
```

- Dans quel dossier se trouve mon fichier ? (*get current working directory*)

```
os.getcwd()
```

- Changer de dossier (*change directory*)

Il y a une différence dans la façon d'indiquer les chemins d'accès en fonction de l'ordinateur.

Mac

- Utiliser un slash /

```
os.chdir('/Users/christophe/.../Documents/Cours/Python')
```

Windows

- Utiliser un double backslash \

```
os.chdir('C:\\Users\christophe\\...\\Documents\\Cours\\Python')
```

2.6 Exercices

Note: Pour chaque exercice, l'output attendu se trouve juste en dessous.

2.6.1 Exercice 1

- Afficher le message "Hello, World!"

```
Hello, World!
```

2.6.2 Exercice 2

2.A

- Calculer et afficher le montant TVAC (tvac) sur base du montant HTVA (htva) et du taux de TVA (taux_tva).

$$\text{TVAC} = \text{HTVA} \times (1 + \text{taux de TVA})$$

Utiliser les valeurs ci-dessous:

- htva: 1000
- taux_tva: 0.21

```
1210.0
```

2.B

- Calculer et afficher le montant HTVA (htva) sur base du montant TVAC (tvac) et du taux de TVA (taux_tva).

$$\text{HTVA} = \frac{\text{TVAC}}{(1 + \text{taux de TVA})}$$

Utiliser les valeurs ci-dessous:

- tvac: 1210
- taux_tva: 0.21

```
1000.0
```

2.C

- Calculer et afficher le taux de tva (taux_tva) sur base du montant TVAC (tvac) et du montant HTVA (htva).

$$\text{taux de TVA} = \frac{\text{TVAC}}{\text{HTVA}} - 1$$

Utiliser les valeurs ci-dessous:

- tvac: 1210
- htva: 1000

```
0.20999999999999996
```

2.D

- Modifiez le script précédent de manière à afficher un chiffre arrondi à 2 décimales. Utilisez la fonction round dans la fonction print.

```
0.21
```

2.E

- Calculer et afficher le montant TVAC (tvac) sur base du montant HTVA (htva), du taux de TVA (taux_tva), et d'un taux de ristourne (taux_ristourne).

$$\text{TVAC} = \text{HTVA} \times (1 - \text{taux de ristourne}) \times (1 + \text{taux de TVA})$$

Utiliser les valeurs ci-dessous:

- htva: 1000
- taux_tva: 0.21
- taux_ristourne: 0.05

Output attendu:

Le prix TVAC est egal a 1149.5

```
Le prix TVAC est egal a 1149.5
```

2.6.3 Exercice 3

- Calculer et afficher l'indice de masse corporelle (imc) d'une personne de 75kg mesurant 1m80. Il se calcule grâce à:

$$\text{IMC} = \frac{\text{poids}}{\text{taille}^2}$$

Utiliser les valeurs ci-dessous:

- poids: 75
- taille: 1.80

```
23.15
```

2.6.4 Exercice 4

- Calculer la longueur de l'hypothénuse d'un triangle rectangle à l'aide de la formule de Pythagore:

$$a^2 + b^2 = c^2$$

De cette formule, on peut donc déduire que:

$$c = \sqrt{a^2 + b^2}$$

Utiliser les valeurs ci-dessous:

- a: 3
- b: 4

```
5.0
```

2.6.5 Exercice 5

- Calculer les racines d'un polynome du deuxième degré:

$$f(x) = ax^2 + bx + c$$

Pour calculer les racines d'une fonction, il faut tout d'abord calculer le Δ:

$$\Delta = b^2 - 4ac$$

Ensuite, si le Δ est positif, ce qui est le cas dans cet exemple, les deux racines, r_1 et r_2 sont:

$$r_1, r_2 = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Calculer les racines du polynome $f(x) = x^2 + 3x - 4$. Donc:

- a: 1
- b: 3
- c: -4

```
Racine 1: -4.0
Racine 2: 1.0
```

2.6.6 Exercice 6

- Calculer et afficher:
  - le perimetre d'un carre (carre_perimetre)
  - l'aire d'un carre (carre_aire)
  - le perimetre d'un rectangle (rectangle_perimetre)
  - l'aire d'un rectangle (rectangle_aire)
  - le perimetre d'un cercle (cercle_perimetre)
  - l'aire d'un cercle (cercle_aire)

Le carre possede des côtés de 5cm (carre_cote)

Le rectangle possede une longueur de 6cm (rectangle_longueur) et une largeur de 3cm (rectangle_largeur)

Le cercle possede un rayon de 4cm (cercle_rayon)

Importer la librairie math pour avoir acces à la valeur de π.

```
Carre         Perim.: 20         Aire: 25
Rect.         Perim.: 18         Aire: 18
Cercle         Perim.: 25.13         Aire: 50.27
```

2.6.7 Exercice 7

- Convertir une vitesse de 120km/h (v_kmh) en une vitesse exprimee en m/s (v_ms).

```
120 km/h correspond a 33.33 m/s.
```

2.6.8 Exercice 8

- Calculer et afficher le montant que vous obtiendrez dans 10 ans si vous placez 5,000€ à 6% sur un compte bancaire. Les intérêts sont composés, donc la formule est:

$$X_T = X_0 * (1 + taux)^T$$

Utiliser les valeurs ci-dessous:

- montant_initial: 5000
- taux: 0.06
- duree: 10

```
Montant final 8954.24
```

CHAPTER

THREE

FONCTIONS

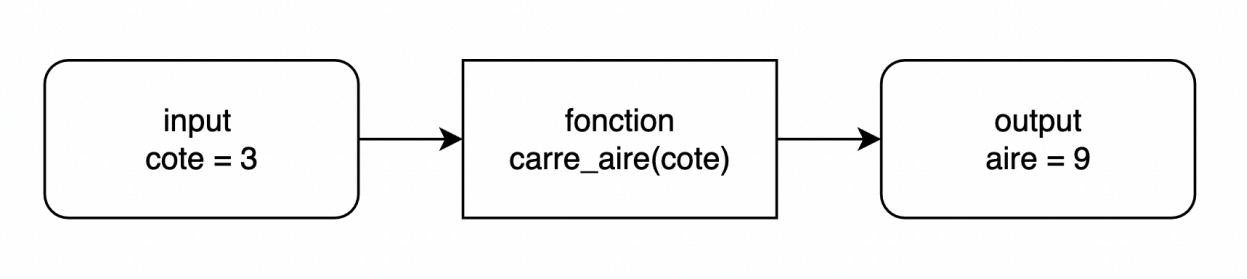
Définition

La programmation est l'art d'apprendre à un ordinateur comment accomplir des tâches qu'il n'était pas capable de réaliser auparavant. L'une des méthodes les plus intéressantes pour y arriver consiste à ajouter de nouvelles instructions au langage de programmation que vous utilisez sous la forme de fonctions originales (Swinnen, 2012).

La synthaxe pour définir une fonction est la suivante:

- Utiliser le mot clé def
- Donner un nom à la fonction
- Indiquer les arguments entre parenthèses
- : et passer à la ligne
- Instructions de la fonction
- Finir par le mot clé return et ce qui est renvoyé par la fonction

La fonction $f(x) = x^2$ permet de calculer le carré d'un nombre. Nous appellons cette fonction carre_aire. Cette fonction n'a besoin que d'un seul argument, cote, qui est la seule information nécessaire. En effet, si nous connaissons la longueur du côté du carré, nous pouvons calculer l'aire.




```
def carre_aire(cote):
    aire = cote**2
    return aire
```

Quand la fonction carre_aire reçoit le chiffre 3, elle exécutera certaines instructions de telle sorte à renvoyer la valeur 9. Cette valeur sera stockée dans la variable aire. Il n'est pas nécessaire que le nom de la variable à côté du return soit identique au nom de la variable dans lequel on stocke le résultat de la fonction.

```
aire = carre_aire(cote=3)
print(aire)
```

```
9
```

Remarque: on peut ne pas indiquer le nom du paramètre dans la parenthèse.

```
aire = carre_aire(3)
print(aire)
```

```
9
```

Lorsque le nombre d'instructions est relativement peu élevé, comme c'est le cas ici, nous pouvons directement renvoyer la valeur sans la stocker temporairement dans une variable.

```
def carre_aire(cote):
    return cote**2
```

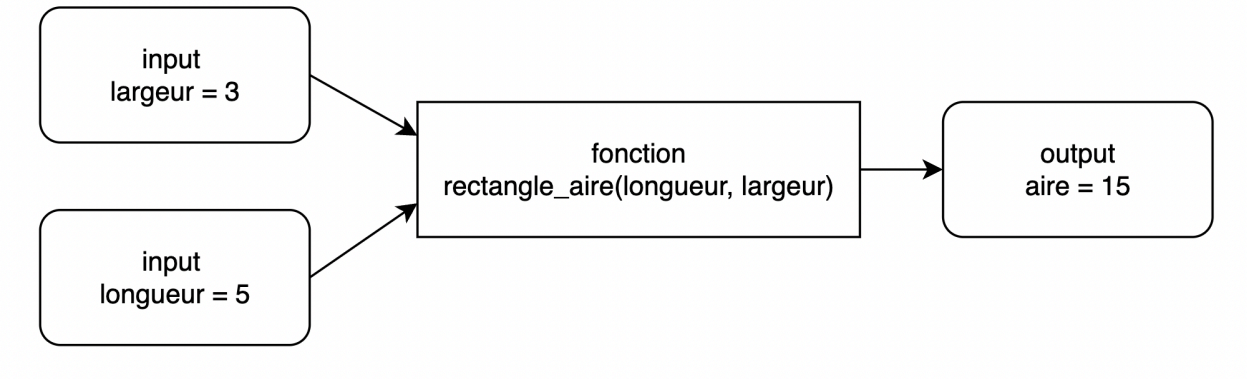
On peut tester à nouveau la fonction en passant la valeur 5. On peut également afficher directement le résultat renvoyé par la fonction, sans stocker ce résultat dans une variable.

```
print(carre_aire(5))
```

```
25
```

3.1 Fonctions à plusieurs arguments

La fonction carre_aire ne possède qu'un seul argument (un seul input). Nous allons créer une fonction qui calcule l'aire d'un rectangle, rectangle_aire qui a besoin de 2 arguments, la longueur et la largeur du rectangle. Nous calculerons ensuite l'aire d'un rectangle dont la longueur est 5cm et la largeur est 3cm.




```
def rectangle_aire(longueur, largeur):
    aire = longueur * largeur
    return aire
```

```
longueur=5
largeur=3
aire = rectangle_aire(longueur, largeur)
print(aire)
```

```
15
```

3.2 Pourquoi utiliser des fonctions ?

L'avantage d'utiliser une fonction est de pouvoir réutiliser des morceaux de code. Supposons que nous avons deux listes dont nous devons calculer la moyenne. Nous pouvons calculer les deux moyennes de la façon suivante (les différents éléments de syntaxe seront abordés plus tard dans le cours):

```
a = [2, 5, 3, 8, 6]
b = [5, 2, 1, 7, 4]

# calculer la moyenne de la liste a
somme_a = 0
for i in range(len(a)):
    somme_a = somme_a + a[i]
moyenne_a = somme_a / len(a)
print(moyenne_a)

# calculer la moyenne de la liste b
somme_b = 0
for i in range(len(b)):
    somme_b = somme_b + b[i]
moyenne_b = somme_b / len(b)
print(moyenne_b)
```

```
4.8
3.8
```

Nous remarquons cependant que le code nécessaire pour calculer la moyenne est évidemment similaire et il n'est pas utile ni efficient de procéder de la sorte. Le code suivant permet de calculer la moyenne de 2 listes à l'aide d'une fonction.

```
a = [2, 5, 3, 8, 6]
b = [5, 2, 1, 7, 4]

def moyenne(liste):
    somme = 0
    for i in range(len(liste)):
        somme = somme + liste[i]
    moyenne = somme / len(liste)
    return moyenne

moyenne_a = moyenne(a)
moyenne_b = moyenne(b)
print(moyenne_a)
print(moyenne_b)
```

```
4.8
3.8
```

3.3 Arguments par défaut

Nous allons créer une fonction tvac qui permet de calculer un montant TVAC sur base d'un montant HTVA (htva) et d'un taux de tva (taux_de_tva).

Testons ensuite cette fonction si htva = 1000 et taux_de_tva = 0.21. Nous faisons appel à la fonction et stockons le résultat dans montant_tvac. Finalement, nous affichons cette variable.

Pour rappel:

$$TVAC = HTVA \times (1 + \text{taux de tva})$$

```
def tvac(htva, taux_de_tva):
    tvac = htva * (1+taux_de_tva)
    return tvac

montant_tvac = tvac(htva=1000, taux_de_tva=0.21)
print("TVAC:", montant_tvac)
```

```
TVAC: 1210.0
```

Nous savons que le taux normal de TVA en Belgique est de 21%. Nous pouvons considérer que c'est une valeur par défault pour l'argument taux_de_tva. Dans ce cas, nous indiquons cette valeur lors de la définition de la fonction.

```
def tvac(htva, taux_de_tva=0.21):
    tvac = htva * (1+taux_de_tva)
    return tvac
```

Ensuite, il y a plusieurs possibilités lors de l'appel de la fonction:

- J'indique que le taux de tva est de 21%

```
montant_tvac = tvac(htva=1000, taux_de_tva=0.21)
print("TVAC:", montant_tvac)
```

```
TVAC: 1210.0
```

- Je n'indique rien et la fonction utilise l'argument par défault

```
montant_tvac = tvac(htva=1000)
print("TVAC:", montant_tvac)
```

```
TVAC: 1210.0
```

- J'indique un autre chiffre que l'argument par défault

```
montant_tvac = tvac(htva=1000, taux_de_tva=0.06)
print("TVAC:", montant_tvac)
```

```
TVAC: 1060.0
```

- Lorsque les paramètres sont indiqués, il n'est pas nécessaire de passer les arguments dans l'ordre.

```
montant_tvac = tvac(taux_de_tva=0.06, htva=1000)
print("TVAC:", montant_tvac)
```

```
TVAC: 1060.0
```

3.4 Exercice - partie 1

Créer une fonction polynome qui calcule et affiche les racines d'un polynôme de degré 2. Calculez les racines du polynôme

$$x^2 + 3x - 4$$

Dans un premier temps, il faut calculer Δ:

$$\Delta = b^2 - 4ac$$

Ensuite les deux racines, r_1 et r_2 sont:

$$r_1, r_2 = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Stocker les valeurs des racines dans rac1 et rac2, et affichez leurs valeurs respectives.

```
1.0
-4.0
```

Modifier la fonction de telle sorte à ce qu'elle renvoie les valeurs (sans les afficher au sein de la fonction).

```
Racine 1: 1.0
Racine 2: -4.0
```

Avant de poursuivre plus loin dans la création de fonctions, nous avons besoin de découvrir d'autres notions:

- Les listes
- Les instructions répétitives (boucle for)
- Les instructions conditionnelles (if et while)

Ces différents points de matière sont abordés dans les prochains chapitres.

3.5 Les fonctions - partie 2

3.5.1 Retour d'une fonction

Jusqu'à présent, chaque fonction retourne une valeur. C'est le plus souvent le cas. Il est toutefois possible de créer une fonction qui ne retourne rien. Par exemple, une fonction qui ne s'occupe que de l'affichage d'une matrice.

```
def print_matrix(mat):
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            print(mat[i][j], end="\t")
        print("")

m = [[1,2,3],[4,5,6]]

print_matrix(m)
```

```
1       2       3
4       5       6
```

Dans ce cas, la fonction ne possède pas de return. Elle renverra donc None si jamais on affecte le retour de la fonction à une variable (par erreur).

```
def print_matrix(mat):
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            print(mat[i][j], end="\t")
        print("")

m = [[1,2,3],[4,5,6]]

display_matrix = print_matrix(m)
print(display_matrix)
```

```
1       2       3
4       5       6
None
```

3.5.2 Imbriquer des fonctions

Une fonction peut faire appel à une autre fonction. Reprenons l'exemple du calcul des racines d'un polynôme. Nous pourrions créer dans un premier temps une fonction qui calcule le Δ. Ensuite, la fonction polynome ferait appel à cette fonction delta.

```
def delta(a, b, c):
    return b**2 - 4*a*c

def polynome(a, b, c):
    d = delta(a, b, c)
    r1 = (-b + d**0.5)/(2*a)
    r2 = (-b - d**0.5)/(2*a)
    print(r1)
    print(r2)

polynome(a=1, b=3, c=-4)
```

```
1.0
-4.0
```

3.5.3 La portée des variables

Quand une variable est définie au sein d'une fonction, elle n'existe qu'au sein de cette fonction. Il n'est donc pas possible d'y faire référence en dehors de cette fonction.

```
def add_1(x):
    x = x + 1
    return x

a = 3
print(a)
a = add_1(a)
print(a)
print(x)
```

```
3
4
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [12], in <cell line: 9>()
      7 a = add_1(a)
      8 print(a)
----> 9 print(x)

NameError: name 'x' is not defined
```

Il y a une exception à cette règle. Si le mot-clé global est utilisé, ceci permet d'accéder à cette variable, sans qu'elle soit initialisée au sein de la fonction, ni passée comme argument.

```
def add_1():
    global a
    a = a + 1
    return a

a = 3
print(a)
add_1()
print(a)
```

```
3
4
```

Il est également possible d'accéder à la valeur d'une variable qui se trouve en dehors de la fonction.

```
x = 2

def add_x(a):
    a = a + x
    return a

y = 5
y = add_x(y)
print(y)
```

```
7
```

3.5.4 Fonctions récursives

Une fonction récursive est une fonction qui fait appel à elle-même.

Un exemple connu de fonction récursive est la fonction factorielle.

La factorielle d'un nombre est obtenue de la façon suivante:

$$x! = x \times (x-1)!$$

et

$$0! = 1! = 1$$

Voici un tableau pour la factorielle des chiffres de 0 à 5:

x	x !
0	1
1	1
2	2
3	6
4	24
5	120

```
def factorial(a):
    if a <= 1:
        return 1
    else:
        return a * factorial(a-1)
```

Nous appelons la fonction au sein d'une boucle de manière à afficher le contenu du tableau précédent.

```
for i in range(6):
    print(i, factorial(i))
```

```
0 1
1 1
2 2
3 6
4 24
5 120
```

Remarque 1: Dans la définition de la fonction, nous voyons en effet que la fonction factorial fait appel à elle-même. Ceci ne fonctionnera que si nous indiquons également un cas de base, dans ce cas, le cas de base est de renvoyer 1 si x <= 1.

Remarque 2: Une fonction récursive peut également être codée de manière non-récursive.

```
def factorial(x):
    f = 1
    for i in range(1, x+1):
```

(continues on next page)

(continued from previous page)

```
        f = f * i
    return f

for i in range(6):
    print(i, factorial(i))
```

```
0 1
1 1
2 2
3 6
4 24
5 120
```

3.6 Exercices - partie 2

3.6.1 Exercice 1

Réaliser une fonction qui permet d'afficher les tables de multiplication et qui prend 2 arguments (a, le nombre à multiplier et b, le nombre de fois qu'il faut le multiplier). Cette fonction ne renvoie rien.

```
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
```

3.6.2 Exercice 2

Supposez que vous avez 5 boules numérotées de 1 à 5 dans un sac et que vous devez en tirer 2 sans remise. Quel est le nombre de possibilités différentes? Si vous connaissez votre cours de probabilité, vous savez que vous devez utiliser la combinaison où n représente le nombre total de boules et p le nombre de boules tirées.

La combinaison se calcule via la formule suivante:

$$ {}_pC_n = \frac{n!}{p!(n-p)!} $$

Dans l'exemple:

$$ {}_2C_5 = \frac{5!}{2!(5-2)!} = \frac{120}{2 \times 6} = 10 $$

Pour vérifier, on notera que les 10 choix sont:

1-2 / 1-3 / 1-4 / 1-5

2-3 / 2-4 / 2-5

3-4 / 3-5

4-5

```
10
```

3.6.3 Exercice 3

Et si l'ordre dans lequel les boules sont tirées a de l'importance ?

Alors, on parle d'un arrangement et la formule est la suivante:

$$ {}_pA_n = \frac{n!}{(n-p)!} $$

Dans l'exemple:

$$ {}_2A_5 = \frac{5!}{(5-2)!} = \frac{120}{6} = 20 $$

C'est logique! On a deux fois plus de possibilités que dans l'exemple précédent. Tirer d'abord la boule n°1 et ensuite la n°2 et inversément compte pour 2 possibilités différentes!

```
20
```

3.6.4 Exercice 4

Le triangle de Pascal est un tableau qui calcule la combinaison entre n, le numéro de la ligne, et p, le numéro de la colonne. Bien sûr, $n \geq p$.

Si on génère ce tableau, on constate que chaque chiffre correspond à la somme de son voisin supérieur et de son voisin supérieur gauche.

```
      1     2     3     4     5     6     7     8     9     10
1     1
2     2     1
3     3     3     1
4     4     6     4     1
5     5     10    10    5     1
6     6     15    20    15    6     1
7     7     21    35    35    21    7     1
8     8     28    56    70    56    28    8     1
9     9     36    84    126   126   84    36    9     1
10    10    45    120   210   252   210   120   45    10
```

CHAPTER

FOUR

LISTES

Jusqu'à présent, chaque variable ne contenait qu'une seule valeur: un chiffre, une valeur booléenne, une chaîne de caractères,… Une liste est une structure de donnéees qui permet de contenir plusieurs variables, soit un ensemble de valeurs.

4.1 Créer une liste

Pour définir une liste dans Python, nous utilisons les crochets ([]). Chaque élément est séparé par une virgule.

Nous allons créer une liste a qui contient les chiffres de 1 à 5 et puis l'afficher.

```
a = [1, 2, 3, 4, 5]
print(a)
```

```
[1, 2, 3, 4, 5]
```

Les valeurs au sein de la liste ne doivent pas nécessairement être du même type (int, float, bool, …).

Nous allons définir et afficher une liste a qui contient les valeurs a, 1, True, et 2.

```
a = ['a', 1, True, 2]
print(a)
```

```
['a', 1, True, 2]
```

Tout comme les entiers sont de types int et les nombres décimaux sont de type float, les listes sont de type list.

```
print(type(a))
```

```
<class 'list'>
```

4.2 Les indices

Chaque élément d'une liste possède un indice (*index*) qui renseigne sa position dans la liste. Le premier indice est l'indice 0. Le dernier indice est $n-1$ si n est le nombre d'éléments dans la liste. Pour accéder à un élément particulier de la liste, il faut indiquer le nom de la liste, suivi de la position de cet élément entre crochets.

- Définir une liste a qui contient les chiffres 6, 4, 1, 3, et 2.
- Afficher le premier élément (position 0)
- Afficher le troisième élément (position 2)
- Afficher l'élément en position 5

```
a = [6, 4, 1, 3, 2]
print(a[0])
print(a[2])
print(a[5])
```

```
6
1
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [17], in <cell line: 4>()
      2 print(a[0])
      3 print(a[2])
----> 4 print(a[5])

IndexError: list index out of range
```

La dernière instruction génère une erreur. En effet, il n'y aucun élément en position 5. On essaie donc d'accéder à un élément en dehors de la liste. Cette erreur causera l'arrêt du programme.

On peut également accéder aux éléments en se référant à la position par rapport à la fin de la liste. Ainsi, le dernier élément aura la position -1, l'avant-dernier élément la position -2, etc.

```
a = [1, 2, 3, 4, 5]
print(a[-1])
print(a[-2])
```

```
5
4
```

4.3 Longueur d'une liste

Pour connaître la longueur d'une liste, nous pouvons utiliser la fonction len.

```
a = [1, 2, 3, 4, 5]
print(len(a))
```

```
5
```

4.4 Sous-ensemble d'une liste

Pour accéder à plusieurs éléments consécutifs au sein d'une liste, on utilisera le nom de la liste, suivi de crochets et on spécifie les indices de début et de fin. Pour rappel, les éléments seront affichés entre le premier indice inclus et le deuxième indice exclus.

```
print(a[1:4])
```

```
[2, 3, 4]
```

Si un troisième chiffre est mentionné, ce sera l'incrément. Le code ci-dessous affiche donc les éléments entre la position 0 (incluse) et la position 5 (exclue), par incrément de 2, soit les éléments aux positions 0, 2, et 4.

```
print(a[0:5:2])
```

```
[1, 3, 5]
```

4.5 Générer des listes rapidement

A l'aide de range et list, il est possible de générer rapidement des listes de taille potentiellement plus importante. range possède un comportement différent en fonction du nombre d'arguments qui sont fournis:

- 1 argument a : de 0 (inclus) à a (non-inclus)
- 2 arguments a et b : de a (inclus) à b (non-inclus)
- 3 arguments a, b, et c: de a (inclus) à b (non-inclus) par incrément de c
- 1 argument: list(range(a)) produit une liste de 0 à a (non-inclus)

```
a1 = list(range(20))
print(a1)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

- 2 arguments: list(range(a, b)) produit une liste de a (inclus) à b (non-inclus).

```
a2 = list(range(10, 15))
print(a2)
```

```
[10, 11, 12, 13, 14]
```

- 3 arguments: list(range(a, b, c)) produit une liste de a (inclus) à b (non-inclus) par pas de c.

```
a3 = list(range(10, 20, 2))
print(a3)
```

```
[10, 12, 14, 16, 18]
```

4.6 Tester l'appartenance

Pour savoir si un élément se trouve dans une liste, nous pouvons effectuer un test à l'aide de in. Par exemple, nous avons une liste qui contient des prénoms et nous souhaitons tester si un prénom se trouve dans la liste.

```
a = ['Christophe', 'Alain', 'Diane', 'Donatien', 'Sandra', 'Thierry']

print('Christophe' in a)
print('Dianne' in a)
```

```
True
False
```

4.7 Les listes à 2 dimensions

De nombreux problèmes mathématiques nécessitent de travailler avec des matrices, c'est-à-dire des tableaux à deux dimensions contenant des valeurs. Nous pouvons encoder une matrice en créant une liste de listes. Supposons que nous souhaitions créer une matrice de 3 lignes et 3 colonnes contenant les chiffres de 1 à 9, nous allons créer une liste contenant elle-même 3 listes:

1	2	3
4	5	6
7	8	9

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(m)
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Nous pouvons accéder à chaque sous-liste de manière séparée.

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print("1ere sous-liste:", m[0])
print("2eme sous-liste:", m[1])
print("3eme sous-liste:", m[2])
```

```
1ere sous-liste: [1, 2, 3]
2eme sous-liste: [4, 5, 6]
3eme sous-liste: [7, 8, 9]
```

L'élément 0 de la liste m est une liste. Si nous souhaitons accéder à des chiffres particuliers:

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(m[0][1])
print(m[1][1])
print(m[3][3])
```

```
2
5
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [12], in <cell line: 4>()
      2 print(m[0][1])
      3 print(m[1][1])
----> 4 print(m[3][3])

IndexError: list index out of range
```

Le troisième print génère une erreur car il n'y a pas de liste en position 3.

Nous pouvons finalement afficher les chiffres tels que présentés dans le tableau.

```
print(m[0][0], m[0][1], m[0][2])
print(m[1][0], m[1][1], m[1][2])
print(m[2][0], m[2][1], m[2][2])
```

```
1 2 3
4 5 6
7 8 9
```

4.8 Les méthodes sur listes

Il existe quelques méthodes qui permettent de modifier une liste, et notamment le fait d'ajouter un ou plusieurs éléments à une liste, potentiellement à une place bien déterminée.

4.8.1 Ajouter un élement à une liste

La méthode append permet d'ajouter un élément à la fin de la liste. Nous commençons par définir une liste qui contient 5 chiffres. Nous affichons cette liste et nous affichons également sa longueur. Ensuite, nous utilisons la méthode append pour ajouter le chiffre 0 à la fin de la liste. Nous réaffichons la liste pour montrer que le changement a bien eu lieu, et que la longueur de la liste a augmenté de 1. Ensuite, nous ajoutons une liste qui contient 2 chiffres, 9 et 5. Dans ce cas, c'est bien la liste qui est ajoutée. On remarque que la longueur de la liste n'a augmenté que de 1 à nouveau, même si on a ajouté une liste qui contient 2 chiffres.

```
a = [1, 2, 4, 6, 3]
print(a)
print("Longueur de la liste:", len(a))
a.append(0)
print(a)
print("Longueur de la liste:", len(a))
a.append([9, 5])
print(a)
print("Longueur de la liste:", len(a))
```

```
[1, 2, 4, 6, 3]
Longueur de la liste: 5
[1, 2, 4, 6, 3, 0]
Longueur de la liste: 6
[1, 2, 4, 6, 3, 0, [9, 5]]
Longueur de la liste: 7
```

4.8.2 Ajouter plusieurs éléments

Nous avons vu dans la section précédente que la méthode append ne permet pas d'ajouter plusieurs chiffres simultanément. La méthode extend permet de le faire. Nous repartons de la même liste initiale. A l'aide de la méthode extend, le nombre d'éléments de la liste passe de 5 à 7.

```
a = [1, 2, 4, 6, 3]
print(a)
print("Longueur de la liste:", len(a))
a.extend([9, 5])
print(a)
print("Longueur de la liste:", len(a))
```

```
[1, 2, 4, 6, 3]
Longueur de la liste: 5
[1, 2, 4, 6, 3, 9, 5]
Longueur de la liste: 7
```

4.8.3 Ajouter un élément à une place spécifique

Jusqu'à présent, nous avons ajouté un ou plusieurs chiffres à la fin de la liste. Si nous souhaitons les insérer à une place particulière au sein de la liste, nous pouvons utiliser la méthode insert. Dans ce cas, il faut spécifier 2 informations: (1) la place à laquelle nous souhaitons insérer le chiffre, et (2) le chiffre en question. Dans l'exemple ci-dessous, nous insérons le chiffre 6 à la position 2 (et pas le chiffre 2 à la position 6).

```
a = [1, 2, 3, 4, 5]
print(a)
a.insert(2, 6)
print(a)
```

```
[1, 2, 3, 4, 5]
[1, 2, 6, 3, 4, 5]
```

4.8.4 Supprimer un élément

Pour supprimer un élément d'une liste, nous devons utiliser la méthode remove. Si la liste contient plusieurs fois le chiffre à supprimer, cette méthode ne supprime que la première occurence rencontrée. Dans l'exemple ci-dessous, la liste a contient 2 fois le chiffre 3. Seul le premier a été supprimé.

```
a = [1, 3, 5, 2, 5, 3, 0, 4]
print(a)
a.remove(2)
print(a)
a.remove(3)
print(a)
```

```
[1, 3, 5, 2, 5, 3, 0, 4]
[1, 3, 5, 5, 3, 0, 4]
[1, 5, 5, 3, 0, 4]
```

4.8.5 Supprimer une liste

Avec la méthode clear, il est possible de supprimer le contenu d'une liste. Cependant la liste existe toujours, bien qu'elle sera vide.

del permet de supprimer la référence à la liste, ce qui implique qu'elle n'existe plus.

```
a = [1, 2, 4, 6, 3]
print("Adresse memoire", hex(id(a)))
print(a)

# on efface le contenu mais on ne la supprime pas
a.clear()
print("Adresse memoire", hex(id(a)))
print(a)

# suppression de la liste
del a
print("Adresse memoire", hex(id(a)))
```

```
Adresse memoire 0x7fc868698b40
[1, 2, 4, 6, 3]
Adresse memoire 0x7fc868698b40
[]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [15], in <cell line: 12>()
     10 # suppression de la liste
     11 del a
---> 12 print("Adresse memoire", hex(id(a)))

NameError: name 'a' is not defined
```

4.8.6 Trier une liste

La méthode sort permet de trier une liste. Par défault, cette méthode trie la liste dans l'ordre croissant. Pour trier une liste dans l'ordre décroissant, il faut spécifier reverse=True.

```
a = [1, 3, 5, 2, 5, 3, 0, 4]
print(a)

# tri et affichage de la liste
a.sort()
print(a)
```

```
[1, 3, 5, 2, 5, 3, 0, 4]
[0, 1, 2, 3, 3, 4, 5, 5]
```

```
# tri decroissant
a.sort(reverse=True)
print(a)
```

```
[5, 5, 4, 3, 3, 2, 1, 0]
```

4.8.7 Inverser une liste

reverse permet d'inverser une liste. Le premier élément devient le dernier et inversément, le deuxième devient avant-dernier, et ainsi de suite. Il ne s'agit donc pas d'un tri !

```
a = [1, 3, 5, 2, 5, 3, 0, 4]
print(a)
a.reverse()
print(a)
```

```
[1, 3, 5, 2, 5, 3, 0, 4]
[4, 0, 3, 5, 2, 5, 3, 1]
```

4.8.8 Compter le nombre d'éléments dans une liste

count permet de compter le nombre d'occurences d'un élément dans une liste.

La liste a contient 3 fois le chiffre 2.

```
a = [1, 2, 4, 6, 3, 2, 4, 2]
print(a)
print("Nb chiffre 2 : ", a.count(2))
```

```
[1, 2, 4, 6, 3, 2, 4, 2]
Nb chiffre 2 :  3
```

4.8.9 Trouver la position d'un élément dans une liste

index renvoie l'indice à laquelle se trouve un élément dans une liste si cet élément fait effectivement partie de la liste, et renvoie une erreur si cet élément ne fait pas partie de la liste.

```
profs = ['Christophe', 'Alain', 'Diane', 'Donatien', 'Sandra', 'Thierry']

print(profs.index('Christophe'))
print(profs.index('Diane'))
print(profs.index('Dianne'))
```

```
0
2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [16], in <cell line: 5>()
      3 print(profs.index('Christophe'))
      4 print(profs.index('Diane'))
----> 5 print(profs.index('Dianne'))

ValueError: 'Dianne' is not in list
```

4.8.10 Supprimer un élément d'une liste sur base de sa position

pop permet de supprimer un élément à une position spécifique dans une liste. Si aucun argument n'est mentionné, c'est le dernier élément de la liste qui est supprimé.

```
a = [1, 3, 5, 2, 5, 3, 0, 4]
print(a)
a.pop(3)      # supprime le chiffre 2 (indice=3)
print(a)
a.pop()       # supprime le chiffre 4 (dernier element)
print(a)
```

```
[1, 3, 5, 2, 5, 3, 0, 4]
[1, 3, 5, 5, 3, 0, 4]
[1, 3, 5, 5, 3, 0]
```

4.8.11 Copier une liste

Supposons qu'une liste soit stockée dans la variable a. Si on souhaite copier le contenu de cette variable a dans une autre variable, b, on pourrait simplement écrire b = a. En affichant les 2 variables, on constate que le contenu des 2 variables est semblable.

```
a = [1, 2, 3, 4]
b = a
print(a)
print(b)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

Cependant, si on modifie la variable a, par exemple, en ajoutant un chiffre à l'aide de la méthode append, on constate que la variable b est également affectée.

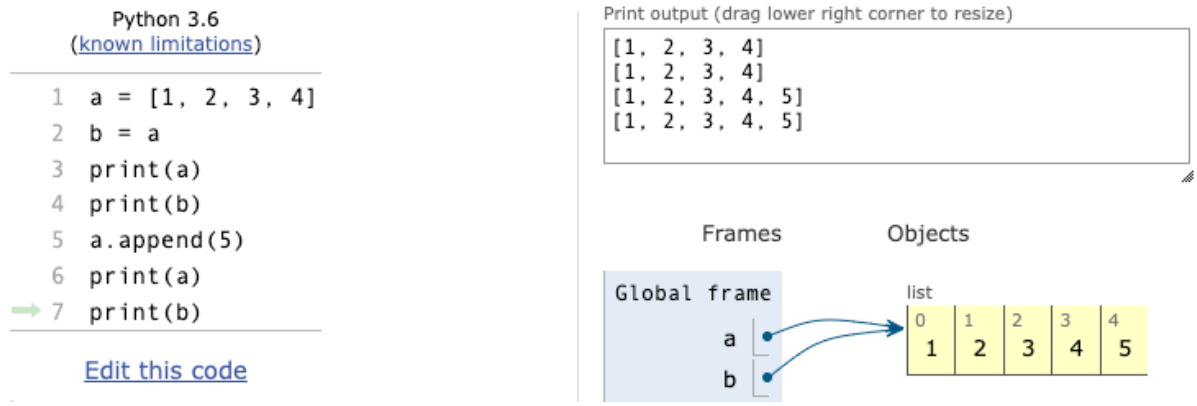
```
a.append(5)
print(a)
print(b)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

La raison est que le signe d'égalité utilisé plus haut a fait en sorte que les 2 variables a et b pointent vers le même objet. On peut d'ailleurs s'en convaincre en affichant l'adresse mémoire de cette liste. Le schéma ci-dessous illustre le fait que les 2 variables pointent vers la même liste.

```
print(hex(id(a)))
print(hex(id(b)))
```

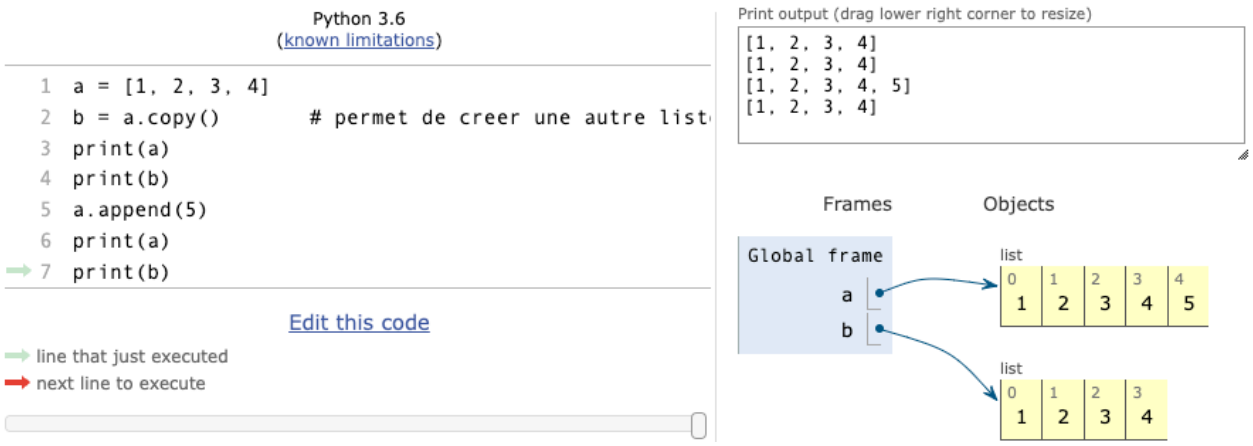
```
0x7fb5f0b8ec00
0x7fb5f0b8ec00
```



La méthode copy permet d'effectuer une copie de la liste a et de la stocker dans la variable b. Ceci crée une 2ème liste qui est indépendante de la première liste. Une modification de a n'affecte donc pas b.

```
a = [1, 2, 3, 4]
b = a.copy()        # permet de creer une autre liste
print(a)
print(b)
a.append(5)
print(a)
print(b)
print(hex(id(a)))
print(hex(id(b)))
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
0x7fb5a0034a80
0x7fb5a00349c0
```



De manière similaire, b = a[:] permet d'atteindre le même résultat.

```
a = [1, 2, 3, 4]
b = a[:] # permet de faire une copie
print(a)
print(b)
a.append(5)
print(a)
print(b)
print(hex(id(a)))
print(hex(id(b)))
```

CHAPTER

FIVE

INSTRUCTIONS RÉPÉTITIVES

Il arrive fréquemment qu'une instruction soit répétée un certain nombre de fois. Il est dès lors plus efficace d'écrire une cette instruction au sein d'une instruction répétitive / une boucle.

Par exemple, supposons que nous souhaitions afficher les chiffres de 0 à 5, nous pourrions écrire :

```
print(0)
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
0
1
2
3
4
5
```

L'instruction print est répétée un certain nombre de fois. Voyons comment une boucle for pourrait nous aider à écrire ceci de manière plus concise.

```
for i in range(6):
    print(i)
```

```
0
1
2
3
4
5
```

Si on souhaite traduire cette expression en français, nous pourrions écrire :

Pour i (for i) qui prend toutes les valeurs jusqu'à 6 (in range(6)), afficher la valeur de i (print(i)). Comme lors de la création d'une fonction, il faut :

- Terminer la première ligne par :.
- Ajouter une indentation aux instructions qui font partie de la boucle. Sans cette indentation, le code ne fonctionnera pas. Vous pouvez également retenir qu'après chaque :, il faut au minimum une ligne indentée.

Dans le chapitre sur les listes, nous avons également vu que range avait un comportement différent en fonction du nombre d'arguments qui sont fournis:

- 1 argument a : de 0 (inclus) à a (non-inclus)

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

- 2 arguments a et b : de a (inclus) à b (non-inclus)

```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

- 3 arguments a, b, et c: de a (inclus) à b (non-inclus) par incrément de c

```
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

Si l'incrément est négatif, les chiffres affichés seront donc décroissants (il faut que $a > b$ dans ce cas).

```
for i in range(10, 1, -2):
    print(i)
```

```
10
8
6
4
2
```

5.1 Parcourir une liste

Pour afficher séparémment chaque élément d'une liste, on peut utiliser une boucle. Dans une liste de n éléments, les éléments sont indexés de la position 0 à la position $n-1$. Il faut donc créer une variable qui prendra les valeurs de 0 (inclus) à n (non-inclus). Pour accéder à la longueur de la liste, et donc connaître n, nous pouvons utiliser la fonction len.

```
voyelles = ['a', 'e', 'i', 'o', 'u', 'y']

for i in range(len(voyelles)):
    print(voyelles[i])
```

```
a
e
i
o
u
y
```

Il est également possible de parcourir une liste de la façon suivante :

```
voyelles = ['a', 'e', 'i', 'o', 'u', 'y']

for voyelle in voyelles:
    print(voyelle)
```

```
a
e
i
o
u
y
```

Cette façon de parcourir la liste est plus intuitive et plus lisible. Toutefois, il ne sera pas toujours possible d'utiliser cette 2ème méthode, notamment lorsqu'on aura besoin d'effectuer des manipulations sur la liste qui nécessite de connaître la position de l'élément au sein de la liste, comme le tri d'une liste par exemple.

5.2 Compteur

Dans cette section, nous ne considérerons que des listes composées uniquement de chiffres entiers. Maintenant que nous sommes capables de parcourir la liste, nous allons pouvoir réaliser des scripts permettant de calculer la somme des éléments, de trouver le minimum et le maximum, etc.

Pour calculer la somme d'une liste, nous devons commencer par initialiser une variable somme=0. Ensuite, nous allons parcourir chaque élément de la liste à l'aide d'une boucle for et ajouter successivement chacun de ces chiffres à la somme.

Pour implémenter un compteur, la logique est très semblable. Comme précédemment, nous initialisons une variable compteur à 0 et ensuite, nous augmentons la valeur du compteur à chaque élément.

```
a = [1, 2, 3, 4, 5]
compteur = 0
for i in range(len(a)):
    compteur = compteur + 1
```

(continues on next page)

(continued from previous page)

```
print(compteur)
```

```
5
```

Ce script n'est pas très utile car il renvoie la même information que la longueur de la liste. Par contre, nous pouvons légèrement modifier ce script de manière à compter uniquement les chiffres qui respectent une certaine condition. Par exemple, pour compter le nombre de chiffres pairs (ou impairs) d'une liste, nous pouvons écrire:

```
a = [1, 2, 3, 4, 5]
nb_pairs = 0
nb_impairs = 0
for i in range(len(a)):
    if a[i] % 2 == 0:
        nb_pairs = nb_pairs + 1
    else:
        nb_impairs = nb_impairs + 1

print("Nombres pairs :", nb_pairs)
print("Nombres impairs :", nb_impairs)
```

```
Nombres pairs : 2
Nombres impairs : 3
```

Pour calculer le minimum d'une liste, nous allons initialiser une variable minimum. Cependant, nous ne pouvons pas lui donner une valeur telle que 0 car ce chiffre n'est peut-être pas dans la liste. C'est d'ailleurs le cas dans notre exemple. Nous allons donc considérer que, dans un premier temps, le minimum sera égal au premier chiffre de la liste. Ensuite, nous parcourons l'entièreté de la liste et nous testons si le chiffre en question est plus petit que le minimum actuel. Si oui, nous effectuons le remplacement (et nous avons une nouvelle valeur pour le minimum).

```
a = [8, 2, 5, 3, 4]
minimum = a[0]
for i in range(len(a)):
    if a[i] < minimum:
        minimum = a[i]

print("Minimum:", minimum)
```

```
Minimum: 2
```

5.3 Double boucle

Nous avons également vu dans le chapitre précédent qu'une liste pouvait contenir d'autres listes. Cette structure s'apparente au concept mathématique de matrices. Nous allons créer une matrice (3x3) qui contient les chiffres de 1 à 9.

```
matrice = [[1,2,3],[4,5,6],[7,8,9]]
```

Si nous souhaitons afficher chaque élément de manière séparée, il faudrait écrire 9 instructions print.

```
print(matrice[0][0])
print(matrice[0][1])
print(matrice[0][2])
#------------------#
print(matrice[1][0])
print(matrice[1][1])
print(matrice[1][2])
#------------------#
print(matrice[2][0])
print(matrice[2][1])
print(matrice[2][2])
```

```
1
2
3
4
5
6
7
8
9
```

A l'aide de ce qui vient d'être vu précédemment, on pourrait obtenir le même résulat à l'aide d'une instruction for. Vous pouvez remarquer que 3 blocs ont été séparés à l'aide de #—#. Bien que ça n'ait aucun effet sur le code, étant donné que c'est un commentaire, cela permet de constater que la seule différence entre ces trois blocs est la valeur du chiffre dans le premier crochet. Nous pouvons donc écrire:

```
for i in range(3):
    print(matrice[i][0])
    print(matrice[i][1])
    print(matrice[i][2])
```

```
1
2
3
4
5
6
7
8
9
```

Mais ce n'est pas tout ! En effet, on constate maintenant (même si on aurait déjà pu le constater plus tôt) que les instructions au sein de la boucle for sont similaires, à l'exception près du chiffre qui se trouve cette fois-ci dans le deuxième crochet. Nous allons donc pouvoir écrire une 2ème boucle au sein de la première. Nous utiliserons la lettre j.

```
for i in range(3):
    for j in range(3):
        print(matrice[i][j])
```

```
1
2
3
4
```

(continues on next page)

(continued from previous page)

```
5
6
7
8
9
```

Top ! On est passé de 9 lignes de codes à 3 lignes, tout ça pour obtenir le même résultat. Pour obtenir un output plus qualitatif et avoir un affichage qui s'apparente à celui d'une matrice, il conviendrait maintenant d'afficher 3 chiffres, puis de passer à la ligne, d'afficher les 3 chiffres suivants, et ainsi de suite. Rappelez-vous que par défaut, l'instruction print passe à la ligne, mais qu'il est possible de changer de comportement par défault en indiquant end=… et ce que l'on souhaite voir afficher après avoir affiché les différents éléments du print. Si nous décidons "d'afficher" une tabulation, obtenu à l'aide de \t après chaque chiffre, ceci devient:

```
for i in range(3):
    for j in range(3):
        print(matrice[i][j], end="\t")
```

```
1       2       3       4       5       6       7       8       9
```

On constate maintenant que tous les chiffres sont affichés sur la même ligne. Il ne reste plus qu'à "afficher" un retour à la ligne à chaque fois que 3 chiffres ont été affichés. Ceci revient à afficher un retour à la ligne une fois que la boucle intérieure (celle avec la variable j) est terminée.

```
for i in range(3):
    for j in range(3):
        print(matrice[i][j], end="\t")
    print("")
```

```
1       2       3
4       5       6
7       8       9
```

5.4 Break

L'instruction break permet de stopper une boucle, si une condition particulière est rencontrée.

```
for i in range(1, 10):
    print(i)
    if i == 5:
        break
```

```
1
2
3
4
5
```

Notez que si l'instruction break se situe avant l'instruction print, alors le résultat n'est pas tout à fait le même.

```
for i in range(1, 10):
    if i == 5:
        break
    print(i)
```

```
1
2
3
4
```

5.5 Continue

L'instruction continue permet de continuer la boucle tout en passant une itération particulière. Supposons qu'on souhaite diviser 10 par tous les chiffres compris entre -5 et 5. Nous savons qu'une division par 0 entraînera une erreur. Il faut donc passer cette itération.

```
for i in range(-5, 6):
    if i == 0:
        print('Pas de division par zero')
        continue
    print("10 /", i, " =", round(10/i, 2))
```

```
10 / -5  = -2.0
10 / -4  = -2.5
10 / -3  = -3.33
10 / -2  = -5.0
10 / -1  = -10.0
Pas de division par zero
10 / 1  = 10.0
10 / 2  = 5.0
10 / 3  = 3.33
10 / 4  = 2.5
10 / 5  = 2.0
```

5.6 Pass

L'instruction pass permet d'indiquer qu'on complètera une partie de code plus tard. Dans l'exemple ci-dessous, on parcourt la liste qui contient les chiffres de 1 à 10. Pour les chiffres pairs, on complètera les instructions à réaliser plus tard. Pour les chiffres impairs, on se contente de les afficher.

```
a = list(range(1, 10))
for i in a:
    if i % 2 == 0:
        # to-do : complete this part
        pass
    else:
        print(i)
```

```
1
3
5
7
9
```

5.7 Les compréhensions de liste

Nous avons vu comment créer des fonctions et comment parcourir une liste. Nous allons maintenant voir comment appliquer une fonction sur l'ensemble des éléments d'une liste.

Nous allons créer une liste a qui contient les chiffres entre 1 et 5. Ensuite, nous allons définir une fonction square qui permet de calculer le carré d'un chiffre. Finalement, on appliquera la fonction sur chaque élément de la liste et chaque valeur sera stockée dans la liste a_square.

```
a = list(range(1, 6))

def square(x):
    return x**2

a_square = []
for i in a:
    a_square.append(square(i))

print(a_square)
```

```
[1, 4, 9, 16, 25]
```

Une autre possibilité est d'utiliser la syntaxe suivante :

```
a_square = [square(i) for i in a]
print(a_square)
```

```
[1, 4, 9, 16, 25]
```

Voyons comment utiliser cette syntaxe si on souhaite représenter une fonction:

$$y = 2x^2 - 4x + 3$$

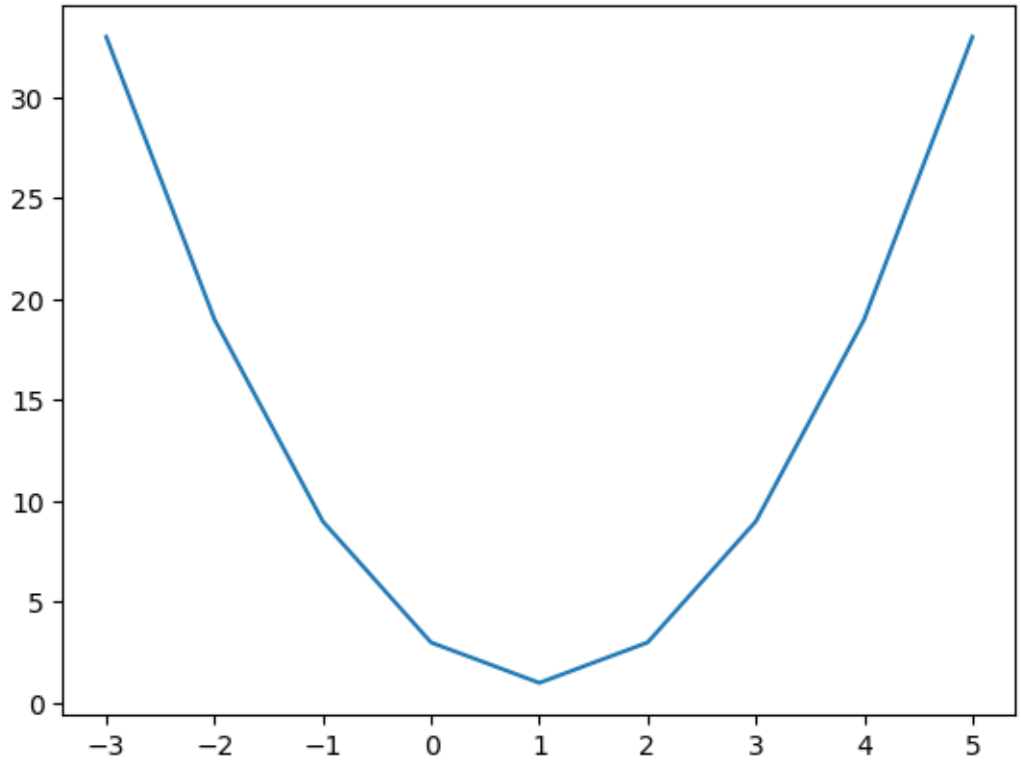
Nous représentons les valeurs entre -3 et 5.

```
def f(x):
    return 2*x**2 - 4*x + 3

x = list(range(-3, 6))
y = [f(i) for i in x]

print(x)
print(y)
```

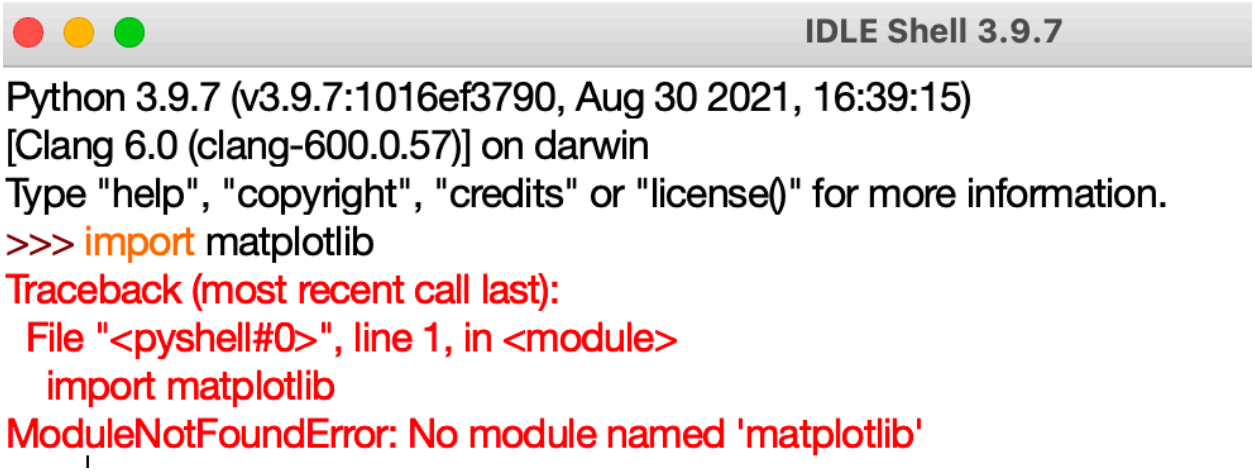
```
[-3, -2, -1, 0, 1, 2, 3, 4, 5]
[33, 19, 9, 3, 1, 3, 9, 19, 33]
```

Ensuite, on importe la librairie matplotlib et plus spécifiquement le module pyplot.

```
import matplotlib.pyplot as plt

plt.plot(x, y);
```



Si vous utilisez IDLE, il est probable que vous rencontriez cette erreur :



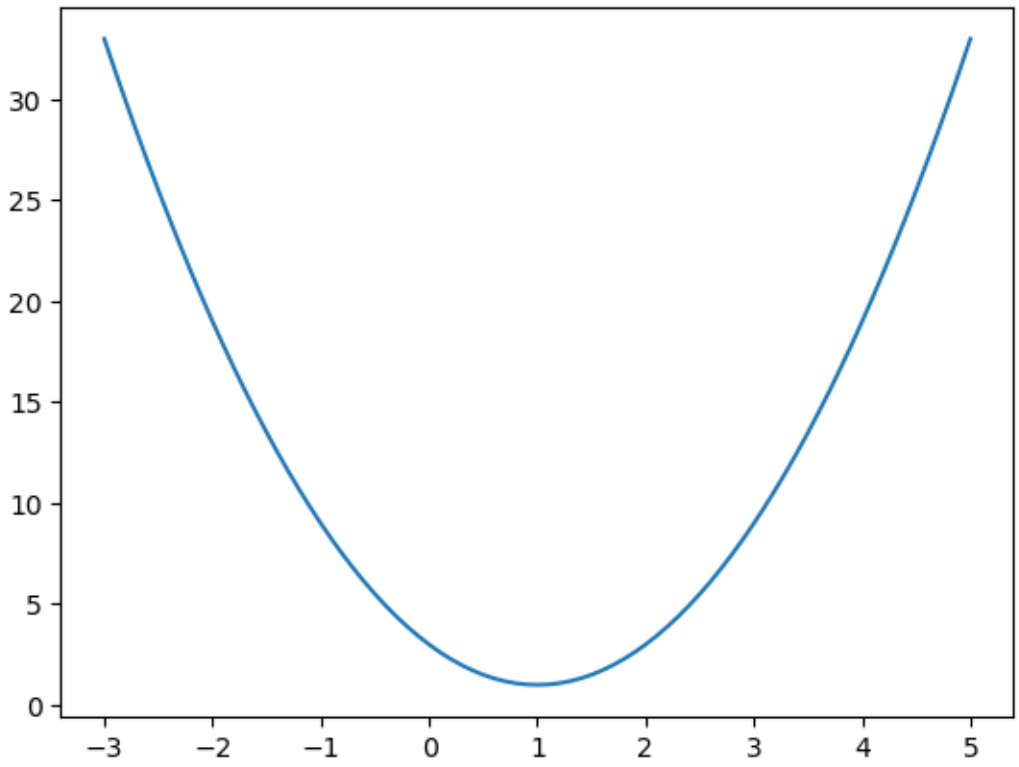
Cette erreur signifie que la librairie n'est pas installée. Dans ce cas, vous pouvez :

- installer via IDLE (voir instructions sur Moodle).
- utiliser une solution en ligne (https://colab.research.google.com/ - nécessite un compte Google et une connexion à internet)

- installer Anaconda (https://www.anaconda.com/)

Pour améliorer l'allure de la courbe, il est nécessaire de représenter plus de points. Notez qu'il n'est pas possible d'avoir un incrément décimal dans la fonction range. Pour contourner ceci, nous générons une liste de chiffres entre -30 et 50 et ensuite, nous divisons tous ces points par 10.

```
x = list(range(-30, 51))
x = [i/10 for i in x]
y = [f(i) for i in x]
plt.plot(x, y);
```



5.8 Exercices

5.8.1 Exercice 1

Créez une variable n égal à 5. Affichez ensuite tous les chiffres entre 1 et n.

```
1
2
3
4
5
```

5.8.2 Exercice 2

Demandez à l'utilisateur d'entrer une borne inférieure et une borne supérieure. Stockez ces informations dans les variables a et b. Affichez tous les chiffres compris entre ces bornes, a et b inclus.

```
Entrez une borne inferieure: 2
Entrez une borne superieure: 10
2
3
4
5
6
7
8
9
10
```

5.8.3 Exercice 3

3A

A l'aide d'une boucle, créez un compte à rebours qui affiche les chiffres de 10 à 0.

```
10
9
8
7
6
5
4
3
2
1
0
```

3B

Améliorez le script précédent en utilisant time.sleep(1) qui permet de faire une pause dans l'exécution de 1 seconde pour avoir un rendu qui ressemble vraiment à un compte à rebours. N'oubliez pas d'importer le module time.

```
10
9
8
7
6
5
4
3
2
1
0
```

5.8.4 Exercice 4

Créez une liste a qui contient les chiffres 2, 5, 8, 9, et 4. Calculez ensuite la somme des éléments de cette liste sans utiliser la fonction sum.

```
28
```

5.8.5 Exercice 5

Reproduisez le contenu (uniquement la zone bleue) de l'image ci-dessous. Chaque chiffre correspond à la somme entre le numéro de ligne et le numéro de colonne.

```
0       1       2       3       4       5
1       2       3       4       5       6
2       3       4       5       6       7
3       4       5       6       7       8
4       5       6       7       8       9
5       6       7       8       9       10
```

5.8.6 Exercice 6

L'opérateur $\sum$, très utilisé en mathématiques et statistiques, indique une somme. Par exemple :

$$\sum_{i=1}^{6} = 1 + 2 + 3 + 4 + 5 + 6 = 21$$

Implémentez un script qui calcule et affiche la somme de 1 à n (supposez que $n = 6$).

```
21
```

5.8.7 Exercice 7

L'opérateur Π, également très utilisé en mathématiques et statistiques, indique un produit. Par exemple:

$$\Pi_{i=1}^{6} = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

Implémentez un script qui calcule et affiche le produit des chiffres entre 1 et n (supposez que $n = 6$).

```
720
```

CHAPTER

SIX

INSTRUCTIONS CONDITIONNELLES

Jusqu'à présent, nous avons considéré une exécution "linéaire". La première ligne est exécutée, ensuite la deuxième, et ainsi de suite. En pratique, il est possible de n'exécuter que certaines parties du code, sous certaines conditions.

L'instruction if réalise un test logique. Un test logique débouche sur 2 possibilités mutuellement exclusives:

- Vrai (True)
- Faux (False)

Pour tester si un nombre est pair ou impair, nous pouvons calculer le reste de la division entière de ce nombre par 2. S'il ne reste rien (le reste vaut 0) alors le nombre est pair, sinon il est impair.

Pour rappel, on peut calculer le reste de la division entière (*le modulo*) à l'aide du symbôle %. Le reste de la division entière de 5 par 2 vaut 1.

```
print(5 % 2)
```

```
1
```

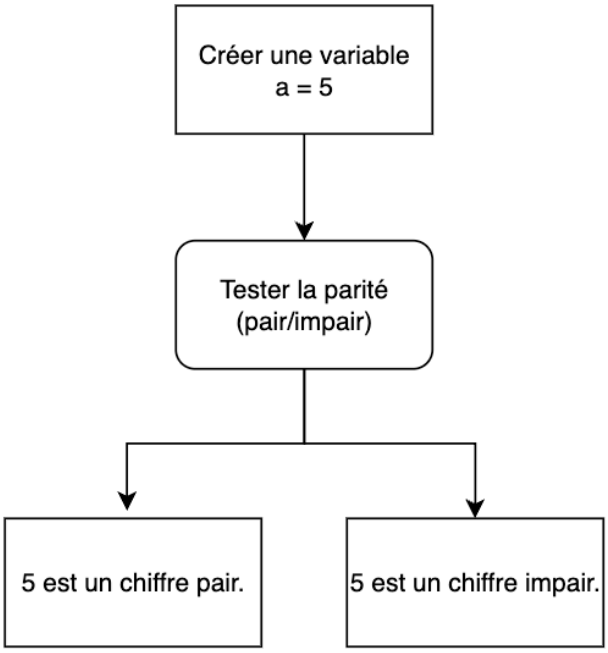
On peut donc tester si le reste de la division entière de 5 par 2 est égal à zéro. Dans ce cas-ci, le test renvoie la valeur False.

```
print(5 % 2 == 0)
```

```
False
```

Voici un exemple complet avec:

1. La définition d'une variable a
2. Un test pour savoir si le contenu de la variable est pair ou impair
3. L'affichage d'un message cohérent avec le résultat du test



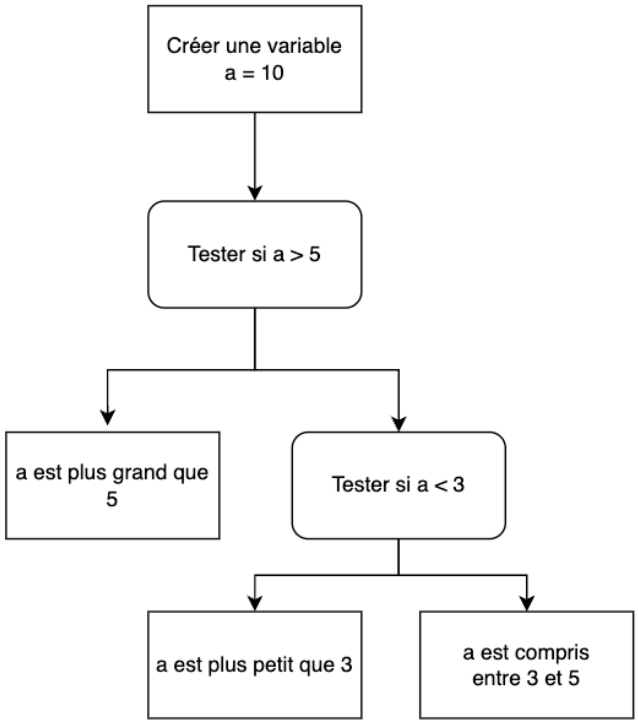

```
a = 5
if a % 2 == 0:
    print(a, "est pair.")
else:
    print(a, "est impair.")
```

```
5 est impair.
```

Il y a plusieurs éléments importants dans ce script. On utilise le mot-clé if pour l'instruction conditionnelle. Ensuite, on écrit le test réalisé. Notez que pour tester une égalité, nous devons utiliser le double symbôle d'égalité (==). Si ce test renvoie la valeur True, alors la première partie du code est exécute. Sinon (else), la deuxième partie est exécutée. Enfin, il y a une tabulation dans le code qui permettent de délimiter les différents morceaux du code. En Python, cette indentation est fondamentale. Deux codes similaires avec des indentations différentes donneront des résultats potentiellement différents.

6.1 Instructions imbriquées

Il est possible d'imbriquer plusieurs conditions. Nous définissons une variable a=10. Ensuite, nous testons si a est plus grand que 5, et si ce n'est pas le cas, nous testons si a est plus petit que 3. Nous afficherons un message cohérent avec le résultat des différents tests.




```
a = 10
if a > 5:
    print(a, "est plus grand que 5")
else:
    if a < 3:
        print(a, "est plus petit que 3")
    else:
        print(a, "est compris entre 3 et 5")
```

```
10 est plus grand que 5
```

En Python, il est possible de contracter un else suivi d'un if par un elif. Ceci rend le code plus lisible car toutes les conditions se trouvent au même niveau.

```
a = 10
if a > 5:
    print(a, "est plus grand que 5")
elif a < 3:
    print(a, "est plus petit que 3")
else:
    print(a, "est compris entre 3 et 5")
```

```
10 est plus grand que 5
```

Etant donné que l'indentation est fondamentale, deux codes similaires avec des indentations différentes donneront des résultats potentiellement différents.

```
a = 11
if a > 5:
    if a % 2 == 0:
        print("plus grand que 5 et pair")
    elif a % 2 == 1:
        print("plus grand que 5 et impair")
```

```
plus grand que 5 et impair
```

```
a = 11
if a > 5:
    if a % 2 == 0:
        print("plus grand que 5 et pair")
elif a % 2 == 1:
    print("plus grand que 5 et impair")
```

Ce second code n'affichera aucun résultat car aucune condition atteinte par le programme n'est vraie.

6.2 Les booléens

Une valeur booléenne peut prendre deux valeurs: VRAI True ou FAUX False. Ces valeurs booléennes permettent de réaliser des tests logiques (AND, OR, XOR).

```
a = True
print(a)
```

```
True
```

```
b = False
print(b)
```

```
False
```

Au niveau du type de variables, ce sont donc des variables booléennes, en anglais bool.

```
a = True
print(type(a))
```

```
<class 'bool'>
```

6.3 Les opérations logiques

Il existe plusieurs opérations logiques:

- “Et”
- “Ou”
- “Ou exclusif”

ET

- a et b est vrai si et seulement si a est vrai et b est vrai. Dans les autres cas, l’opération est fausse.

a	b	a “et” b
True	True	True
False	True	False
True	False	False
False	False	False

Pour cette opération, il faut utiliser le symbôle &.

```
a = True
b = True
print(a, "and", b, "is", a & b)
```

```
True and True is True
```

```
a = False
b = True
print(a, "and", b, "is", a & b)
```

```
False and True is False
```

```
a = True
b = False
print(a, "and", b, "is", a & b)
```

```
True and False is False
```

```
a = False
b = False
print(a, "and", b, "is", a & b)
```

```
False and False is False
```

OU

- a ou b est vrai si et seulement si au moins une des deux variables est vraie. Dans les autres cas, l'opération est fausse.

a	b	a "ou" b
True	True	True
False	True	True
True	False	True
False	False	False

Pour cette opération, il faut utiliser une barre verticle (|).

```
a = True
b = True
print(a, "or", b, "is", a | b)
```

```
True or True is True
```

```
a = False
b = True
print(a, "or", b, "is", a | b)
```

```
False or True is True
```

```
a = True
b = False
print(a, "or", b, "is", a | b)
```

```
True or False is True
```

```
a = False
b = False
print(a, "or", b, "is", a | b)
```

```
False or False is False
```

Ou exclusif

- a ou b est vrai si et seulement si exactement une des deux variables est vraie. Dans les autres cas (si les deux sont vraies ou si les deux sont fausses), l'opération est fausse.

a	b	a "ou exclusif" b
True	True	False
False	True	True
True	False	True
False	False	False

Pour cette opération, il faut le symbôle ^.

```
a = True
b = True
print(a, "or", b, "is", a ^ b)
```

```
True or True is False
```

```
a = False
b = True
print(a, "xor", b, "is", a ^ b)
```

```
False xor True is True
```

```
a = True
b = False
print(a, "xor", b, "is", a ^ b)
```

```
True xor False is True
```

```
a = False
b = False
print(a, "xor", b, "is", a ^ b)
```

```
False xor False is False
```

Les booléens comme résultat d'un test

Les valeurs booléennes peuvent aussi être le résultat d'un test. Commençons par définir a=5 et b=3.

```
a = 5
b = 3
```

Pour tester une égalité, il faut utiliser ==.

Warning: Une erreur fréquente est de n'utiliser qu'un seul signe d'égalité.

Comme a n'est pas égal à b, le résultat du test est faux.

```
print(a==b)
```

```
False
```

C'est vrai que a est plus grand que b.

```
print(a>=b)
```

```
True
```

Et donc c'est faux que a soit plus petit que b.

```
print(a<b)
```

```
False
```

Finalement, c'est vrai que le contenu des deux variables est différent.

```
print(a!=b)
```

```
True
```

6.4 Tests multiples

Supposez que les variables a, b, c, et d possèdent respectivement les valeurs 4, 5, 6 et 7.

Est-ce que l'affirmation suivante est vraie ou fausse ?

```
a = 4
b = 5
c = 6
d = 7
```

```
print((a > c) | (b != d))
```

```
True
```

En effet, il convient d'évaluer les expressions en suivant l'ordre imposé par les parenthèses. a n'est pas plus grand que c mais en même temps b est différent de d. Nous avons donc: False or True, ce qui est True.

6.5 Exercices

6.5.1 Exercice 1

En repartant de l'exercice sur le calcul des racines d'un polynôme:

$$f(x) = ax^2 + bx + c$$

Pour calculer les racines d'une fonction, il faut tout d'abord calculer le Δ:

$$\Delta = b^2 - 4ac$$

Ajoutez une condition après le calcul du delta :

- Si Δ est positif, il y a 2 racines
- Si Δ est nul, il n'y qu'une racine

- Si Δ est négatif, il n'y a pas de racine

Si le Δ est positif, r_1 et r_2 sont:

$$r_1, r_2 = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Si le Δ est nul, r_1 est:

$$r_1 = \frac{-b}{2a}$$

Si le Δ est négatif, il n'y a pas de racine.

Calculer les racines du polynome $f(x) = x^2 + -4x + 4$. Donc:

- a: 1
- b: -4
- c: 4

```
Racine: 2.0
```

6.5.2 Exercice 2

Demander à l'utilisateur d'entrer un entier. Indiquer si le chiffre est pair ou impair.

```
Entrez un nombre entier: 2
2 est un chiffre pair.
```

6.5.3 Exercice 3

Demander à l'utilisateur d'entrer deux entiers. Calculer la somme et indiquer si celle-ci est paire ou impaire.

```
Entrez un nombre entier: 5
Entrez un nombre entier: 3
8 est un chiffre pair.
```

6.5.4 Exercice 4

Demander à l'utilisateur d'entrer deux entiers. Indiquer si a est plus grand que b, plus petit que b, ou si les chiffres sont égaux.

```
Entrez un nombre entier: 4
Entrez un nombre entier: 2
4 est plus grand que 2
```

6.5.5 Exercice 5

Demander à l'utilisateur d'entrer deux entiers a et b. On fait l'hypothèse que l'utilisateur entre deux entiers différents. Vérifier si ce sont des multiples (vérifier dans les deux sens: a peut être multiple de b ou b peut être un multiple de a).

```
Entrez un nombre entier: 16
Entrez un nombre entier: 4
16 est un multiple de 4
```

6.5.6 Exercice 6

Demander à l'utilisateur d'entrer un montant HTVA, un taux de TVA et un montant TVAC. Vérifier ensuite si les informations fournies sont cohérentes.

```
Entrez un montant HTVA: 1210
Entrez un taux de TVA: 0.21
Entrez un montant TVAC: 1000
Les informations sont incoherentes
```

6.5.7 Exercice 7

- Importer la librairie random
- Générer une valeur aléatoire comprise entre 1 et 10 et stocker cette valeur dans la variable valeur
- Demander à l'utilisateur d'entrer un entier et stocker cet entier dans la variable essai
- Tester si valeur et essai sont égaux (en valeur)
  - Si c'est le cas, afficher "Gagné !"
  - Sinon, afficher "Perdu, c'était [valeur]"

```
Entrer un chiffre: 5
Perdu, c'était 8
```

6.5.8 Exercice 8

- Importer la librairie random
- Générer une valeur aléatoire comprise entre 1 et 10 et stocker cette valeur dans la variable valeur
- Demander à l'utilisateur d'entrer un entier et stocker cet entier dans la variable essai
- Tester si valeur et essai sont égaux (en valeur)
  - Si c'est le cas, afficher "Gagné !"
  - Sinon, tester si la essai est plus grand que valeur
    - Si c'est le cas, afficher "Trop grand, c'était [valeur]"
    - Sinon, afficher "Trop petit, c'était [valeur]"

```
Entrer un chiffre: 2
Trop petit, c'était 4
```

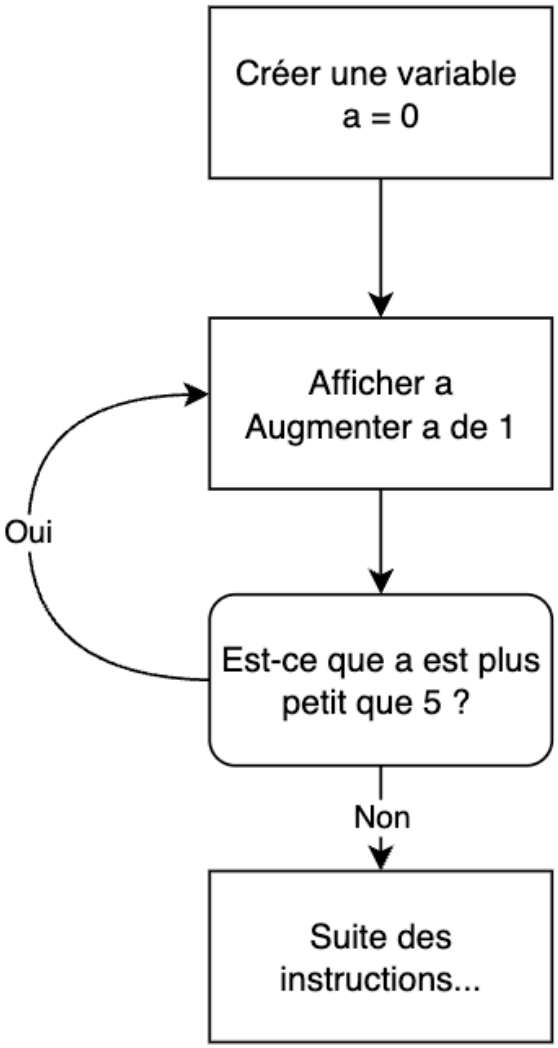
CHAPTER

SEVEN

INSTRUCTIONS WHILE

Après avoir découvert, for et if, il est temps de découvrir while. L'instruction while s'utilise quand il faut exécuter certaines instructions **tant qu'une condition est vraie**.

Par exemple, tant que a est plus petit que 5, j'affiche sa valeur et j'augmente ensuite cette valeur de 1. A chaque itération, la valeur de a est affichée. Ensuite, a augmente de 1. Finalement, on teste si a est toujours plus petit que 5.



```
a = 0
while a < 5:
    print(a)
    a = a + 1
```

```
0
1
2
3
4
```

7.1 Boucle infinie

Lorsqu'on utilise while, il est possible de créer une boucle infinie… Il faut donc être particulièrement attentif à ce qui est écrit. Si plutôt que d'écrire a = a - 1, j'écris (de manière eronnée) a = a + 1, a sera toujours plus grand que zéro et le script continuera à l'infini…

```
a = 5
print("i vaut:", a)
while a > 0:
    print("Je diminue a de 1")
    print("a vaut maintenant: ", a)
    a = a + 1 # erreur!
```

7.2 Exercices

7.2.1 Exercice 1

Modifier le script du jeu "Deviner un chiffre" de manière à permettre à l'utilisateur d'entrer un essai tant qu'il ne trouve pas la bonne réponse.

```
Entrer un chiffre: 5
Entrer un chiffre: 4
Entrer un chiffre: 6
Entrer un chiffre: 3
Entrer un chiffre: 2
Entrer un chiffre: 1
Entrer un chiffre: 8
Entrer un chiffre: 7
Entrer un chiffre: 9
Bravo
```

7.2.2 Exercice 2

Modifier le script du jeu “Deviner un chiffre” de manière à permettre à l’utilisateur d’entrer un essai tant qu’il ne trouve pas la bonne réponse, avec un maximum de 3 essais.

```
Entrer un chiffre: 1
Entrer un chiffre: 2
Entrer un chiffre: 3
Game Over!
Bonne reponse: 5
```

CHAPTER

EIGHT

CHAÎNES DE CARACTÈRES

Une chaine de caractères est comparable à une liste dans le sens où chaque lettre (ou symbole) d'une chaine de caractère possède une position au sein de la chaine de caractères. Tout comme les listes, la première lettre de la chaîne de caractères possède l'indice 0.

```
prenom = 'Christophe'
print(prenom[0])
```

```
C
```

Nous pouvons donc facilement afficher les initiales du nom et du prénom d'une personne.

```
prenom = 'Christophe'
nom = 'Desagre'
print('Mes initiales sont', prenom[0], nom[0])
```

```
Mes initiales sont C D
```

Pour aller plus loin

Si on souhaite que les initiales soient collées, il faudrait utiliser le paramètre sep dans la fonction print. Ce paramètre affiche par défault un espace entre chaque élément à afficher. En choisissant de ne rien afficher entre chaque élémént, on peut donc obtenir le résultat souhaité.

```
prenom = 'Christophe'
nom = 'Desagre'
print('Mes initiales sont ', prenom[0], nom[0], sep="")
```

```
Mes initiales sont CD
```

Bien que cela puisse paraître étrange à première vue, il est possible d'utiliser l'opérateur d'addition entre des chaines de caractères. Ceci permet de les assembler, en utilisant le symbôle +. Ainsi, nous pouvons définir deux variables, prenom et nom. Ensuite, nous pouvons créer une 3ème variable nom_complet en additionnant les deux premières variables.

```
prenom = 'Christophe'
nom = 'Desagre'
nom_complet = prenom + nom
print(nom_complet)
```

```
ChristopheDesagre
```

Note: Pour obtenir un résultat plus qualitatif, nous pouvons ajouter un espace entre le prénom et le nom.

```
prenom = 'Christophe'
nom = 'Desagre'
nom_complet = prenom + " " + nom
print(nom_complet)
```

```
Christophe Desagre
```

Pour afficher une chaîne de caractères sur plusieurs lignes, on peut utiliser 3 simples quotes ou 3 double quotes

```
adresse_bureau = '''Rue au Bois, 365
Bruxelles'''
print(adresse_bureau)
```

```
Rue au Bois, 365
Bruxelles
```

Pour connaître la longueur d'une chaîne de caractères, on peut utiliser len.

```
prenom = 'Christophe'
print(len(prenom))
```

```
10
```

Etant donné qu'une chaîne de caractères peut être considérée comme une liste de caractères, il est donc possible de ne retenir que certains caractères en fonction de leur positionnement. Par exemple, les 5 premiers caractères de mon prénom sont:

```
prenom = 'Christophe'
print(prenom[:5])
```

```
Chris
```

Et les 3 derniers sont:

```
prenom = 'Christophe'
print(prenom[-3:])
```

```
phe
```

Si je n'affiche qu'une lettre sur 2, cela donne:

```
prenom = 'Christophe'
print(prenom[::2])
```

```
Crsoh
```

8.1 Les méthodes sur chaînes de caractères

8.1.1 Minuscules et majuscules

lower permet de transformer toutes les lettres d'une chaîne de caractères en minuscules.

```
prenom = 'Christophe'
print(prenom.lower())
```

```
christophe
```

Inversément, upper permet de transformer toutes les lettres d'une chaîne de caractères en majuscules.

```
prenom = 'christophe'
print(prenom.upper())
```

```
CHRISTOPHE
```

Finalement, capitalize permet d'avoir une première lettre en majuscule, et toutes les autres en minuscule.

```
prenom = 'christophe'
print(prenom.capitalize())
```

```
Christophe
```

8.2 Exercices

8.2.1 Exercice 1

Sur base du nom et du prénom, créez une adresse email "prenom.nom@ichec.be".

Utilisez :

- prenom = Christophe
- nom = Desagre

```
christophe.desagre@ichec.be
```

8.2.2 Exercice 2

Réaliser un script qui permet de tester si une adresse mail finit par "@ichec.be". Si c'est le cas, indiquez que l'adresse mail est valide.

```
Adresse mail valide
```

Demander à l'utilisateur d'entrer un mot et tester si celui-ci est un palindrome.

8.2.3 Exercice 3

Compter le nombre de voyelles dans un mot

```
3
```

8.2.4 Exercice 4

Compter le nombre de consonnes dans un mot

```
7
```

8.2.5 Exercice 5

Réaliser un script qui permet de vérifier qu'un compte bancaire est correct. En Belgique, un compte bancaire sous format IBAN commence par "BE" et est suivi de 14 chiffres. Les différentes étapes pour déterminer s'il s'agit d'un compte IBAN valide sont les suivantes (https://fr.wikipedia.org/wiki/International_Bank_Account_Number):

- Prendre les 4 premiers caractères et les placer à la fin de la chaîne de caractère
- Remplacer les lettres "B" et "E" par leur équivalent hexadécimal (B=11 et E=14)
- Vérifier que le modulo du nombre obtenu à l'étape 2 par 97 est égal à 1.

```
True
```

CHAPTER

NINE

DICTIONNAIRES

Dans ce chapitre, nous aborderons le concept de dictionnaire. Le fil conducteur de ce chapitre sera de répliquer le tableau suivant:

Nom	Age	Profession
Christophe	30	Enseignant

Compte tenu des concepts abordés jusqu'o présent, nous pourrions utiliser deux listes : une première liste qui contient les attributs (théoriques), c-à-d le prénom, l'âge, et la profession, et une deuxième liste qui contient les valeurs, c-à-d Christophe, 30, et enseignant. Le code suivant permet d'implémenter cette approche

```
liste1 = ['Prenom', 'Age', 'Profession']
liste2 = ['Christophe', 30, 'Enseignant']

print(liste1[0], ":", liste2[0])
print(liste1[1], ":", liste2[1])
print(liste1[2], ":", liste2[2])
```

```
Prenom : Christophe
Age : 30
Profession : Enseignant
```

Après analyse de ce code, on comprend que pour afficher un élément particulier, il faut connaître sa position au sein de la liste. Dans l'exemple précédent, étant donné qu'il n'y a que 3 attributs, c'est facile. Mais quand serait-il si le nombre d'attributs augmentait?

9.1 Création d'un dictionnaire

Les dictionnaires permettent de solutionner ce problème. Un dictionnaire est défini en Python à l'aide des accolades (*curly brackets*) { et }. Un dictionnaire est composé de clés (les attributs théoriques) et de valeurs. Lors de la création du dictionnaire, il faut associer ces clés et ces valeurs.

Chaque clé est unique !

- Créer un dictionnaire informations qui contient les informations mentionnées dans le tableau ci-dessus.

```
informations = {'Nom':'Christophe',
                'Age': 30,
                'Profession':'Enseignant'}
```

(continues on next page)

(continued from previous page)

```
print("Nom:", informations['Nom'])
print("Age:", informations['Age'])
print("Profession:", informations['Profession'])
```

```
Nom: Christophe
Age: 30
Profession: Enseignant
```

- Pour connaître la liste des clés du dictionnaire:

```
print("Keys:", informations.keys())
```

```
Keys: dict_keys(['Nom', 'Age', 'Profession'])
```

- Pour connaître la liste des valeurs du dictionnaire

```
print("Values:", informations.values())
```

```
Values: dict_values(['Christophe', 30, 'Enseignant'])
```

- Pour connaître les couples clés-valeurs du dictionnaire:

```
print("Items:")
print(informations.items())
```

```
Items:
dict_items([('Nom', 'Christophe'), ('Age', 30), ('Profession', 'Enseignant')])
```

- Pour parcourir l'ensemble du dictionnaire:

```
for key, value in informations.items():
    print(key, "-", value)
```

```
Nom - Christophe
Age - 30
Profession - Enseignant
```

Tout d'abord, il faut se rappeler que informations.items() retourne des paires clé/valeur. C'est la raison pour laquelle, il y a deux éléments juste à côté du mot-clé for, c-à-d key et value. Ensuite, ces deux variables sont affichées dans le print.

Nous allons maintenant ajouter d'autres personnes dans le tableau :

Nom	Age	Profession
Christophe	30	Enseignant
Francois	40	Directeur
Juliette	20	Etudiante

```
prenoms = ['Christophe', 'Francois', 'Juliette']
ages = [30, 20, 40]
```

(continues on next page)

(continued from previous page)

```
professions = ['Enseignant', 'Directeur', 'Etudiante']

informations = {'Nom': prenoms,
                'Age': ages,
                'Profession': professions}

for key, value in informations.items():
    print(key, "-", value)
```

```
Nom - ['Christophe', 'Francois', 'Juliette']
Age - [30, 20, 40]
Profession - ['Enseignant', 'Directeur', 'Etudiante']
```

L'ensemble des informations est affichée mais pour revenir à un format qui ressemble d'avantage à un tableau, le code se complique considérablement.

```
for key in informations.keys():
    print("%10s"%(key), end=" ")
print("")
for i in range(len(informations[key])):
    for value in informations.values():
        print("%10s"%(value[i]), end=" ")
    print("")
```

```
       Nom        Age Profession
 Christophe         30 Enseignant
   Francois         20  Directeur
   Juliette         40  Etudiante
```

9.2 Exercices

9.2.1 Exercice 1

Réaliser un script qui permet de compter le nombre d'occurences de chaque élément dans une liste.

Utiliser la liste suivante:

[2, 4, 3, 9, 5, 9, 0, 9, 2, 3, 9]

```
2 - 2
4 - 1
3 - 2
9 - 4
5 - 1
0 - 1
```

9.2.2 Exercice 2

Réaliser un script qui permet de trouver le mode d'une liste de valeurs (le mode est la valeur qui revient le plus fréquemment). Utiliser la liste définie précédemment. Le mode est donc le chiffre 9 qui apparaît 4 fois dans la liste.

```
mode 9 - 4 occurences
```

CHAPTER

TEN

EXERCICES SUPPLÉMENTAIRES

Pour chacun des codes ci-dessous, déterminez ce qui sera affiché par ce script.

10.1 Double boucles

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(3):
        s += m[i][j]
print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(3):
        s += m[i][j]
    print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

for i in range(3):
    s = 0
    for j in range(3):
        s += m[i][j]
    print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for j in range(3):
    for i in range(3):
        s += m[i][j]
    print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]
```

(continues on next page)

(continued from previous page)

```
for j in range(3):
    s = 0
    for i in range(3):
        s += m[i][j]
    print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

for i in range(3):
    for j in range(i+1):
        print(i, j, m[i][j])
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(i+1):
        s += m[i][j]
print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(3):
        if i == j:
            s += m[i][j]
print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(3):
        if i > j:
            s += m[i][j]
print(s)
```

```
m = [[1,2,3],[4,5,6],[7,8,9]]

s = 0
for i in range(3):
    for j in range(3):
        if i <= j:
            s += m[i][j]
print(s)
```

```
m1 = [[1,2,3]]
m2 = [[1],[3],[5]]
s = 0
for i in range(1):
```

(continues on next page)

(continued from previous page)

```
    for j in range(3):
        s += m1[i][j]*m2[j][i]
print(s)
```

10.2 while

```
x = 0
while x < 5:
    print(x)
    x = x + 1
```

```
x = 0
while x < 5:
    x = x + 1
    print(x)
```

```
x = 0
while x < 5:
    x = x + 1
print(x)
```

```
x = 10
while x < 5:
    x = x + 1
print(x)
```

```
x = 10
while x > 5:
    x = x + 1
    print(x)
```

```
x = 10
while x > 5:
    x = x + 1
print(x)
```