Chapitre 1 : Rappels sur la programmation en Python

1. Qu'est-ce que Python ?

Python a été créé en 1989 par **Guido van Rossum**, aux **Pays-Bas**. Le créateur a choisi ce nom en référence à la série télévisée ***Monty Python's Flying Circus*.** La première version publique du langage a été publiée en 1991.

La *Python Software Foundation* 1 est l'association qui organise le développement de Python et anime

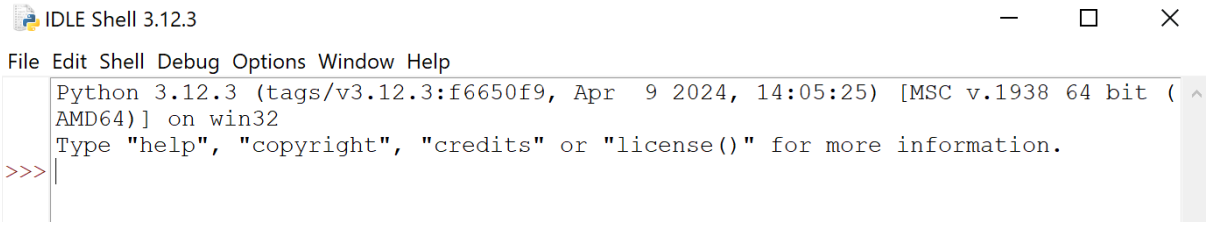
La communauté de développeurs et d'utilisateurs, sous l'égide de *la Python Software Foundation*, continue de développer ce langage.

Les caractéristiques intéressantes de ce langage sont :

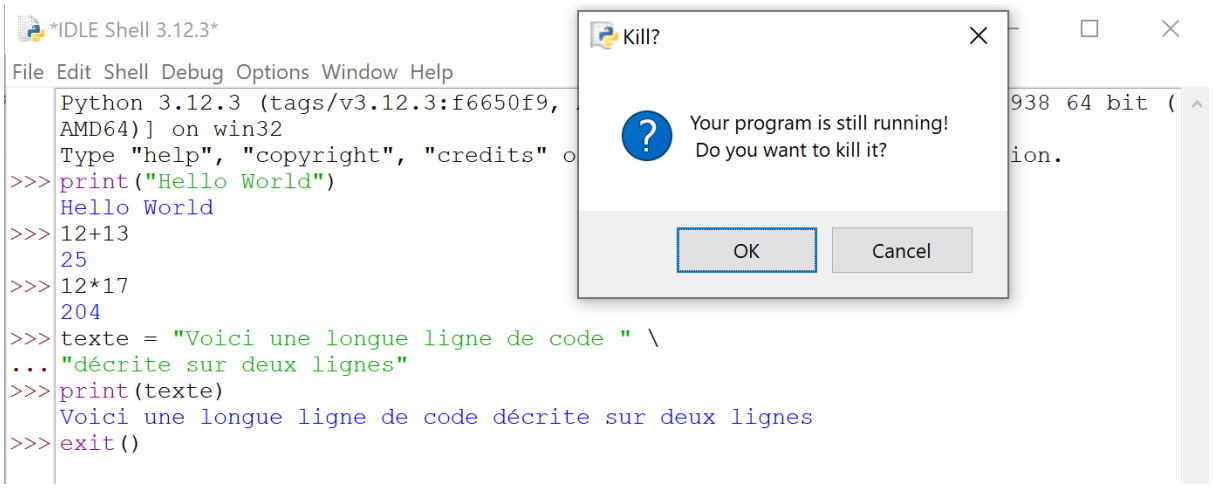
- Il marche sur de nombreux systèmes d'exploitation : Windows, Mac OS X, Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.
- L'installation de Python est gratuite sur n'importe quel ordinateur.
- Un script Python n'a pas besoin d'être compilé pour être exécuté.
- Il est possible de concevoir en Python des concepts qui contrefassent celles du monde réel (une molécule d'ADN, une protéine, un atome, etc.) avec un certain nombre de règles de fonctionnement et d'interactions.

1.1. Interpréteur shell

Un shell est un interpréteur de commandes interactif permettant d'interagir avec l'ordinateur.



On peut effectuer différentes opérations dans ce shell, par exemple :

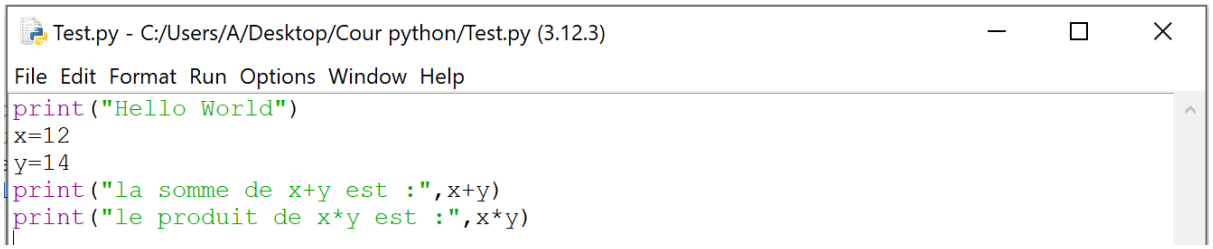


On peut créer un fichier **Test.py** en suivant ces étapes :

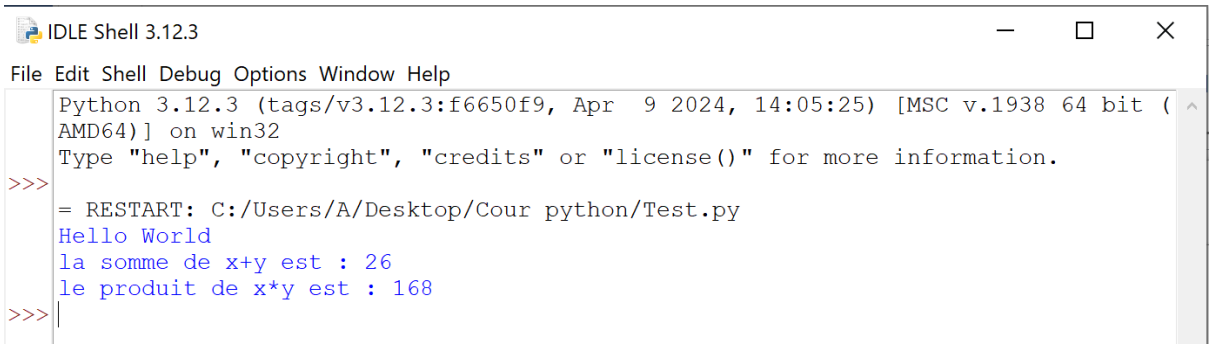
Dans la fenêtre **IDLE Shell 3.12.3,** choisir **New File**. Le fichier est alors créé sous le nom **untitled.** Ensuite, l'enregistrer sous le nom **Test.py.**

Exemple : dans le fichier Test.py on peut faire un exemple simple qui calculer et afficher la somme et le produit de deux nombres.

Fichier **Test.py**



Après avoir écrit le programme, on clique sur **Run**, puis sur **Run Module** pour obtenir le résultat qui s'affiche dans l'**IDLE Shell 3.12.3.**



2. les variables

Une **variable** est une zone de la mémoire de l'ordinateur dans laquelle une **valeur** est stockée.

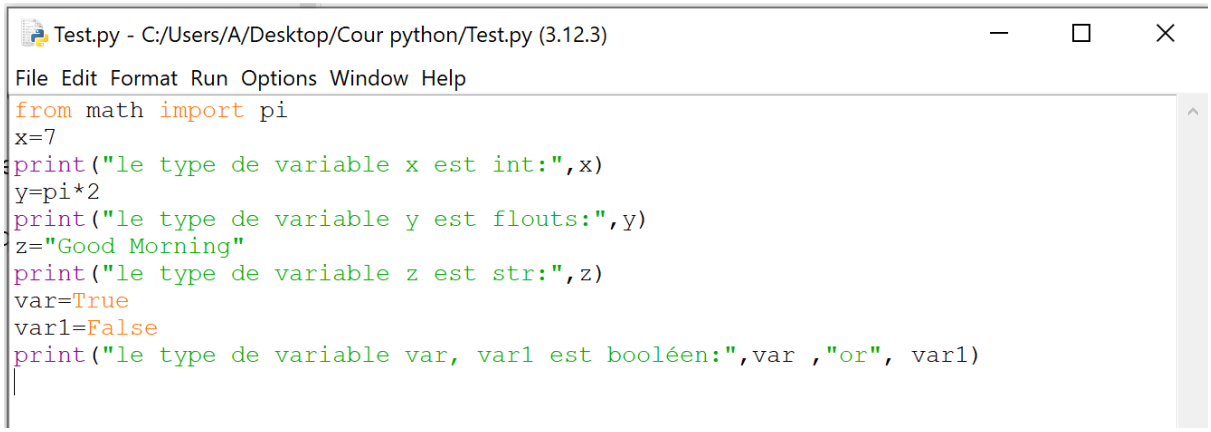
2.1. Les types de variables:

Les quatre principaux types de variable sont : les entiers (**integer** ou **int**), les nombres décimaux (**floats**), les chaînes de caractères (**string** ou **str**), et des variables de type booléen (qui prend la valeur **True** ou **False)**.

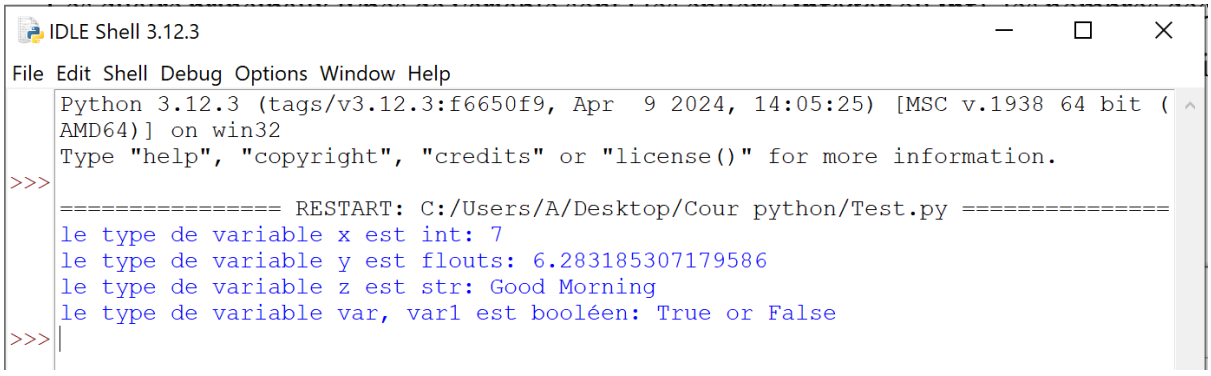
2.2. Nommage

Le nom des variables peut être formé de lettres minuscules, de lettres majuscules, de nombres (0 à 9) ou du caractère souligné (_). Vous ne pouvez pas utiliser d'espace dans un nom de variable.

Exemple :

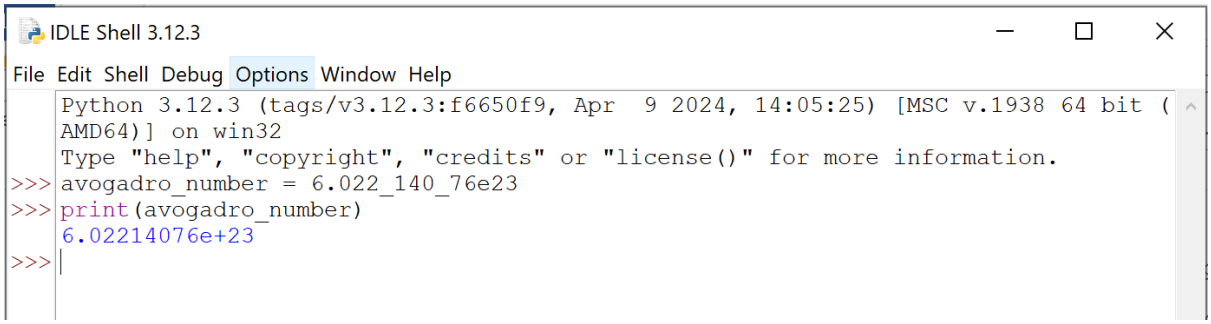


Le résultat :



2.3. Écriture scientifique:

- On utilise la lettre « e » pour la puissance.
- On utilise le caractère « souligné » (ou underscore) « _ » pour séparer des groupes de chiffres.



3. Les Opérations

3.1. Opérations sur les types numériques

Les quatre opérations arithmétiques de base s'effectuent simplement sur les types numériques (entiers et floats). Lorsqu'une opération est effectuée entre un entier et un float, le résultat est de type float.

« + » opérateur de somme

« - » opérateur de soustraction

« * » opérateur de produit

« / » opérateur de division

Il y a d'autre opération pour la puissance et le reste …

« ** » opérateur de puissance

« // » opérateur utilisant pour trouver le quotient

« % » opérateur de modulo, utilisant pour trouver le reste de division.

« += » opérateur d'incrémentation.

Les opérateurs « -=, *= et /= » se comportent de manière similaire pour la soustraction, la multiplication et la division.

Test.py - C:/Users/A/Desktop/Cour python/Test.py (3.12.3)

File Edit Format Run Options Window Help

```
a=7
b=5
S1=a+b
S2=a-b
S3=a*b
S4=a/b
S5=a**b
S6=a//b
S7=a%b
i=1
i+=1
print("a=",a)
print("b=",b)
print("la somme de a+b=",S1,"la soustraction de a-b=",S2)
print("le produit a*b=",S3,"opérateur de division a/b=",S4)
print("opérateur de puissance a**b=",S5,"le quotient a//b=",S6)
print("le reste a%b=",S7,"l'incrémentation i+=",i)
```

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC v.1938 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/A/Desktop/Cour python/Test.py
a= 7
b= 5
la somme de a+b= 12 la soustraction de a-b= 2
le produit a*b= 35 opérateur de division a/b= 1.4
opérateur de puissance a**b= 16807 le quotient a//b= 1
le reste a%b= 2 l'incrémentation i+= 2
>>>
```

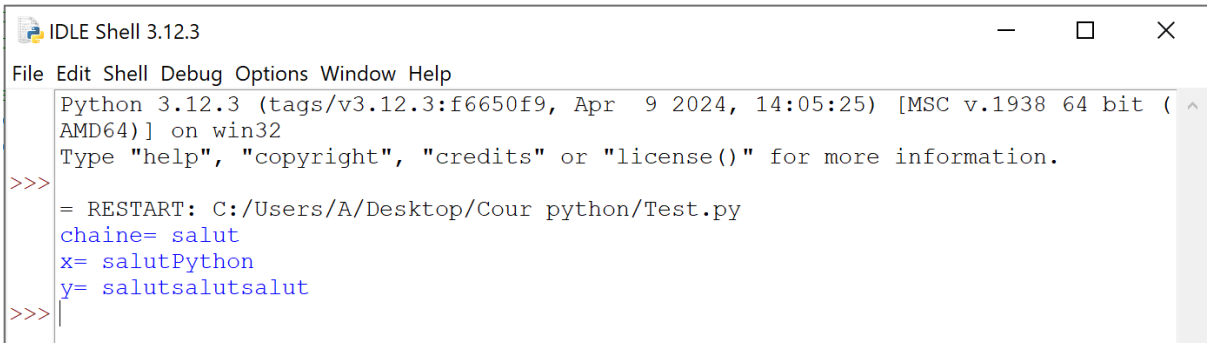
3.2. Opérations sur les chaînes de caractères

Pour les chaînes de caractères, deux opérations sont possibles, l'addition « + » pour concatène les caractères, et la multiplication « * » pour répéter les caractères.

Test.py - C:/Users/A/Desktop/Cour python/Test.py (3.12.3)

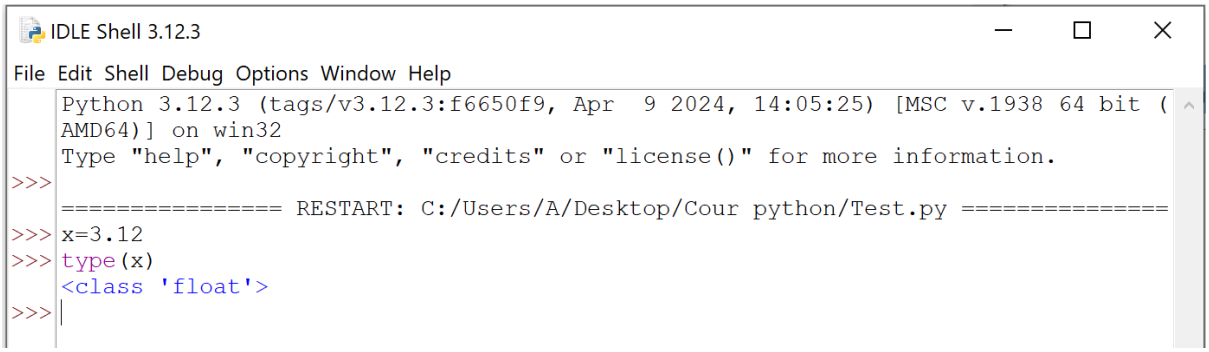
File Edit Format Run Options Window Help

```
chaine="salut"
x=chaine+"Python"
y=chaine*3
print("chaine=",chaine)
print("x=",x)
print("y=",y)
```



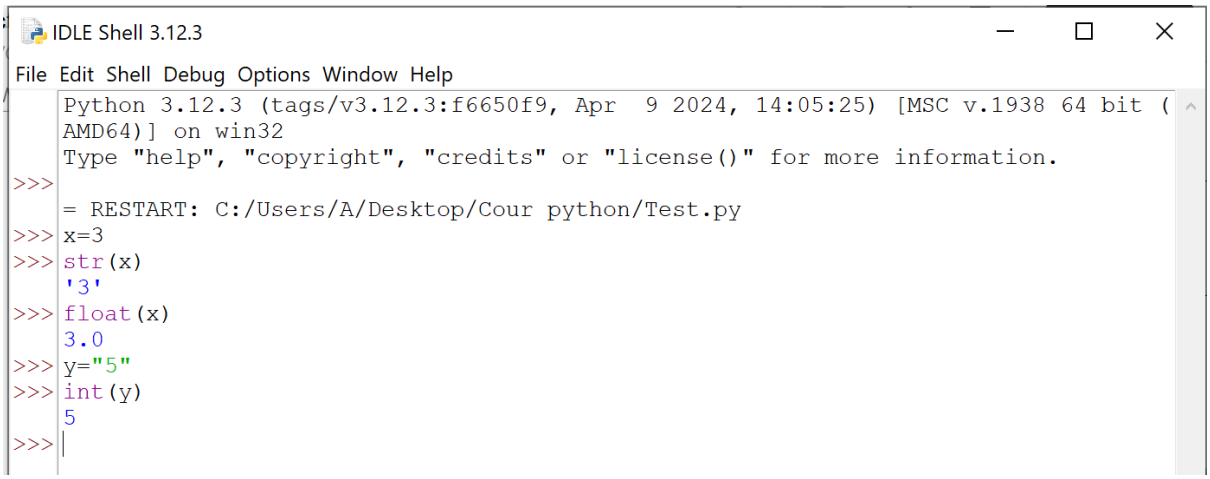
- **La fonction type ()**

On utilise la fonction type () pour connaitre le type de variable.



3.3. Conversion de types

La conversion de type signifier de passer d'un type à un autre utilisant les fonctions suivant : int(), float() et str().



- **La fonction print()**

La fonction print () utilisant pour affichage des contenues de caractères.

- **f-strings:**

Les **f-strings** permettent une meilleure organisation de l'affichage des variables.

```
>>> x=32
>>> nom="MOUAD"
>>> print(f"{nom} a {x} ans ")
MOUAD a 32 ans
>>> print("{nom} a {x} ans ")
{nom} a {x} ans
>>>
```

4. Les Tests

Les tests permettent à l'ordinateur de prendre des décisions. Pour cela, Python utilise l'instruction if associée à une comparaison.

Exemple 1 : if simple

```
age = 20

if age >= 18:
    print("Vous êtes majeur.")
```

```
= RESTART: C:/Users/A/Desktop/Cour python/Test.py
Vous êtes majeur.
>>>
```

Exemple 2 : if avec else

```
age = 15

if age >= 18:
    print("Vous êtes majeur.")
else:
    print("Vous êtes mineur.")
```

```
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py ===============
Vous êtes mineur.
>>>
```

4.1. Tests multiples

Les tests multiples permettent de tester plusieurs conditions en même temps en utilisant des opérateurs booléens. Les deux opérateurs les plus couramment utilisés sont **OU** et **ET**.

Opérateur OU :

Condition 1	Opérateur	Condition 2	Résultat
Vrai	OU	Vrai	Vrai
Vrai	OU	Faux	Vrai
Faux	OU	Vrai	Vrai
Faux	OU	Faux	Faux

Opérateur ET :

Condition 1	Opérateur	Condition 2	Résultat
Vrai	ET	Vrai	Vrai
Vrai	ET	Faux	Faux
Faux	ET	Vrai	Faux
Faux	ET	Faux	Faux

Exemple avec et (and)

On veut vérifier que l'âge est **entre 18 et 30 ans**.

```
age = 25

if age >= 18 and age <= 30:
    print("L'âge est compris entre 18 et 30 ans.")
else:
    print("L'âge n'est pas dans l'intervalle.")
```

```
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py ===============
L'âge est compris entre 18 et 30 ans.
>>>
```

Exemple avec ou (or)

On veut vérifier si la personne est **étudiant ou retraité**.

```
statut = "étudiant"

if statut == "étudiant" or statut == "retraité":
    print("Réduction accordée.")
else:
    print("Pas de réduction.")
```

```
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py ===============
Réduction accordée.
>>>
```

4.2. Boucles for

Les boucles utilisant pour réaliser les tâches répétitives de manière compacte et efficace.

```
# Exemple d'utilisation de la boucle for
for i in range(1, 6):   # range(1,6) génère les nombres 1,2,3,4,5
    print("Valeur de i =", i)
```

```
>>>
= RESTART: C:/Users/A/Desktop/Cour python/Test.py
Valeur de i = 1
Valeur de i = 2
Valeur de i = 3
Valeur de i = 4
Valeur de i = 5
>>>
```

4.3. Comparaisons

Python est capable d'effectuer toute une série de comparaisons entre le contenu de deux variables, telles que :

Opérateur de comparaison	Signification
==	égal à
!=	différent de
>	strictement supérieur à
>=	supérieur ou égal à
<	strictement inférieur à
<=	inférieur ou égal à

```
x = 10
y = 15

print("x == y ?", x == y)    # égalité
print("x != y ?", x != y)    # différent
print("x < y ?", x < y)      # inférieur
print("x > y ?", x > y)      # supérieur
print("x <= y ?", x <= y)    # inférieur ou égal
print("x >= y ?", x >= y)    # supérieur ou égal
```

```
x == y ? False
x != y ? True
x < y ? True
x > y ? False
x <= y ? True
x >= y ? False
>>>
```

4.4. Boucles while

La boucle est une série d'instructions qui s'exécutent tant qu'une condition est vraie.

Exemple :

```
i = 1  # initialisation
while i <= 5:   # la boucle continue tant que i <= 5
    print("Valeur de i =", i)
    i = i + 1   # incrémentation pour éviter une boucle infinie
```

```
---------- RESTART: C:/Users/A/Desktop/Cour Python/test.py ----------
Valeur de i = 1
Valeur de i = 2
Valeur de i = 3
Valeur de i = 4
Valeur de i = 5
>>
```

```
i = 0
while i < 10:
    reponse = input("Entrez un entier supérieur à 10 : ")
    i = int(reponse)
```

```
========== RESTART: C:/Users/A/Desktop/Cour Python/test.py ==========
Entrez un entier supérieur à 10 : 5
Entrez un entier supérieur à 10 : 20
>>>
```

- **La fonction input() :**

La fonction **input()** prend en argument un message (sous la forme d'une chaîne de caractères), demande à l'utilisateur d'entrer une valeur et renvoie celle-ci sous forme d'une chaîne de caractères, qu'il faut ensuite convertir en entier avec la fonction **int()**.

5. Les Modules

Les **modules** sont des programmes Python qui contiennent des fonctions que l'on est amené à souvent réutiliser (on les appelle aussi bibliothèques, ou *libraries* en anglais). Ce sont des « boîtes à outils » qui vous seront très utiles.

5.1. Importation de modules

nous avons rencontré la notion de module plusieurs fois, notamment lorsque nous avons voulu tirer un nombre aléatoire :

```
>>> import random
>>> random.randint(5, 20)
19
>>>
```

Nous utilisons la fonction randint(5, 20) du module *random*. Cette fonction renvoie un nombre entier tiré aléatoirement entre 5 inclus et 20 inclus.

6. Les Fichiers

6.1. Lecture dans un fichier

Une grande partie de l'information en biologie est stockée sous forme de texte dans des fichiers. Pour traiter cette information, vous devez le plus souvent lire ou écrire dans un ou plusieurs fichiers. Python possède pour cela de nombreux outils qui vous simplifient la vie.

- **Méthode .readlines()**

Avant de passer à un exemple concret, créez un fichier dans l'éditeur de texte de votre choix avec le contenu suivant :

```
fichier=open("C:/Users/A/Desktop/animaux.txt", "r")
lignes=fichier.readlines()
for ligne in lignes:
    print(ligne.strip())
```

```
>>>
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py ===============
souris
girafe
lion
leopard
vache
chat
chien
>>>
```

Nom de fichier.close() utilisant pour fermer le fichier 🗃 .

```
with open("C:/Users/A/Desktop/animaux.txt", "r") as fichier:
    lignes = fichier.readlines()
    for ligne in lignes:
        print(ligne.strip())
```

```
>>>
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py ===============
souris
girafe
lion
leopard
vache
chat
chien
>>>
```

- **Méthode .read()**

La méthode .read() lit tout le contenu d'un fichier et renvoie une chaîne de caractères unique :

```
with open("C:/Users/A/Desktop/animaux.txt", "r") as fichier:
    a=fichier.read()
    print(a)
```

```
= RESTART: C:/Users/A/Desktop/Cour python/Test.py
souris
girafe
lion
leopard
vache
chat
chien

>>>
```

- **Méthode .readline()**

La méthode .readline() (sans s à la fin) lit une ligne d'un fichier et la renvoie sous forme de chaîne de caractères. À chaque nouvel appel de .readline(), la ligne suivante est renvoyée. Associée à la boucle while, cette méthode permet de lire un fichier ligne par ligne :

```
with open("C:/Users/A/Desktop/animaux.txt", "r") as fich:
     ligne=fich.read()
     while ligne != "":
         print(ligne)
         ligne=fich.readline()
```

6.2. Écriture dans un fichier

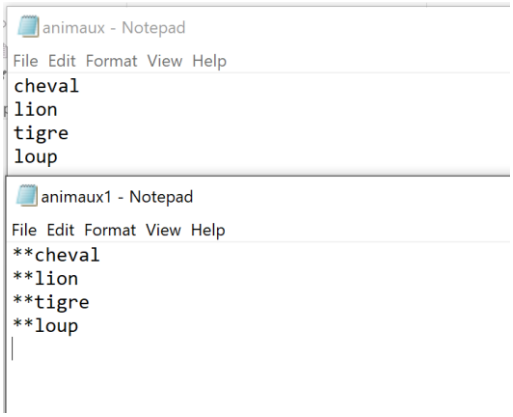
Écrire dans un fichier est aussi simple que de le lire. Voyez l'exemple suivant :

```
with open("C:/Users/A/Desktop/animaux.txt", "r") as fich:
    with open("C:/Users/A/Desktop/animaux_copy.txt", "w") as copie:
        for animal in fich:
            copie.write(animal)
```

6.3. Ouvrir deux fichiers avec l'instruction with

On peut avec l'instruction with ouvrir deux fichiers (ou plus) en même temps. Voyez l'exemple suivant :

```
with open("C:/Users/A/Desktop/animaux.txt", "r") as fich1, open ("C:/Users/A/Desktop/animaux1.txt", "w") as fich2:
        for ligne in fich1:
          fich2.write("**"+ligne)
```



7. Les Listes

Une **liste** est une structure de données qui contient une collection d'objets Python. Il s'agit d'un nouveau type par rapport aux entiers, *float*, booléens et chaînes de caractères que nous avons vus jusqu'à maintenant. On parle aussi d'**objet séquentiel** en ce sens qu'il contient une séquence d'autres objets.

```
>>> animaux=["girafe","tiger","singe","souris"]
>>> animaux
['girafe', 'tiger', 'singe', 'souris']
>>>
```

Un des gros avantages d'une liste est que vous accédez à ses éléments par leur position. Ce numéro est appelé **indice** (ou *index*) de la liste.

```
>>> animaux=["girafe","tiger","singe","souris"]
>>> animaux
['girafe', 'tiger', 'singe', 'souris']
>>> animaux[3]
'souris'
>>>
```

Les listes supportent l'opérateur + de concaténation, ainsi que l'opérateur * pour la duplication :

```
>>> animaux=["girafe","tiger","singe","souris"]
>>> animaux
['girafe', 'tiger', 'singe', 'souris']
>>> animaux[3]
'souris'
>>> ani1=["chat" , "chien"]
>>> animaux+ani1
['girafe', 'tiger', 'singe', 'souris', 'chat', 'chien']
>>> ani1*4
['chat', 'chien', 'chat', 'chien', 'chat', 'chien', 'chat', 'chien']
>>>
```

- **Fonction len ()**

L'instruction len() vous permet de connaître la longueur d'une liste, c'est-à-dire le nombre d'éléments que contient la liste.

- **Les fonctions range() et list()**

L'instruction range() est une fonction spéciale en Python qui génère des nombres entiers compris dans un intervalle. Lorsqu'elle est utilisée en combinaison avec la fonction list(), on obtient une liste d'entiers.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(4)
range(0, 4)
>>> list(range(4))
[0, 1, 2, 3]
>>>
```

8. Dictionnaires et tuples

8.1. Dictionnaires

Un **dictionnaire** contient une collection d'objets Python auxquels on accède à l'aide d'une **clé** de correspondance plutôt qu'un indice. Ainsi, il ne s'agit pas d'objets séquentiels comme les listes, mais plutôt d'objets dits de correspondance (*mapping objects* en anglais) ou tableaux associatifs.

```
= RESTART: C:/Users/A/Desktop/Cour python/Test.py
>>> animal1={}
>>> animal1["nom"]="girafe"
>>> animal1["taille"]=5.0
>>> animal1["poids"]=1100
>>> animal1
{'nom': 'girafe', 'taille': 5.0, 'poids': 1100}
>>>
```

- **Fonction len()**

Comme pour les listes, l'instruction len() renvoie la longueur du dictionnaire, sauf qu'ici il s'agit du nombre de couples clé / valeur. Voici un exemple d'utilisation :

```
>>> animal1
{'nom': 'girafe', 'taille': 5.0, 'poids': 1100}
>>> len(animal1)
3
>>>
```

- **Itération sur les clés pour obtenir les valeurs**

Si on souhaite voir toutes les associations clés / valeurs, on peut itérer sur un dictionnaire de la manière suivante :

```
animal2 = {'nom': 'singe', 'poids': 70, 'taille': 1.75}
for key in animal2:
        print(key, animal2[key])
```

```
================ RESTART: C:/Users/A/Desktop/Cour python/Test.py =====
nom singe
poids 70
taille 1.75
>>>
```

- **Méthodes .keys() et .values()**

Les méthodes .keys() et .values() renvoient, comme vous vous en doutez, les clés et les valeurs d'un dictionnaire :

```
nom singe
poids 70
taille 1.75
>>> animal2.keys()
dict_keys(['nom', 'poids', 'taille'])
>>> animals2.values()
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    animals2.values()
NameError: name 'animals2' is not defined. Did you mean: 'animal2'?
>>> animal.values()
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    animal.values()
NameError: name 'animal' is not defined. Did you mean: 'animal2'?
>>> animal2.values()
dict_values(['singe', 70, 1.75])
>>>
```

```
>>> animal2.values()
dict_values(['singe', 70, 1.75])
>>> list(animal2.values())
['singe', 70, 1.75]
>>>
```

- **Méthode .items()**

La méthode .items() renvoie un nouvel objet dict_items :

```
>>> animal2.items()
dict_items([('nom', 'singe'), ('poids', 70), ('taille', 1.75)])
>>>
```

```
>>> for key, val in animal2.items():
...     print(key, val)
...
...
nom singe
poids 70
taille 1.75
>>>
```

- **Méthode .get()**

La méthode .get() extrait la valeur associée à une clé mais ne renvoie pas d'erreur si la clé n'existe pas :

```
>>> animal2.get("nom")
'singe'
>>> animal2.get("age")
>>>
```

Ici, la valeur associée à la clé nom est singe, mais la clé age n'existe pas. On peut également indiquer à .get() une valeur par défaut si la clé n'existe pas :

```
>>> animal2.get("age", 23)
23
```

- **Liste de dictionnaires**

En créant une liste de dictionnaires qui possèdent les mêmes clés, on obtient une structure qui ressemble à une base de données :

```
>>> animal2
{'nom': 'singe', 'poids': 70, 'taille': 1.75}
>>> animal1={"nom":'girafe', "poid":1100, "taille": 5}
>>> animaux=[animal1, animal2]
>>> animaux
[{'nom': 'girafe', 'poid': 1100, 'taille': 5}, {'nom': 'singe', 'poids': 70, 'ta
ille': 1.75}]
>>>
>>> for ani in animaux :
...     print(ani["nom"])
...
...
girafe
singe
>>>
```

8.2. Tuples

Les **tuples** (« n-uplets » en français) sont des **objets séquentiels** correspondant aux listes, mais ils sont toutefois **non modifiables**. On dit aussi qu'ils sont **immuables**. Vous verrez ci-dessous que nous les avons déjà croisés à plusieurs reprises !

```
>>> tup1=(1, 3 , 5)
>>> tub1
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    tub1
NameError: name 'tub1' is not defined. Did you mean: 'tup1'?
>>> tup1
(1, 3, 5)
>>> type(tup1)
<class 'tuple'>
>>> tup1=tup1 + (9,)
>>> tup1
(1, 3, 5, 9)
>>>
```

Les tuples sont souvent utilisés pour l'**affectation multiple**, c'est-à-dire, affecter des valeurs à plusieurs variables en même temps :

```
>>> x,y,z = 10, 1, 7
>>> x
10
>>> y
1
>>> z
7
>>>
```