Cours de Python

Filière MIP S1 2023-2024

Pr. Abdelalim SADIQ
a.sadiq@uit.ac.ma

```
1. Introduction à Python
    - Qu'est-ce que Python
    - Installation et configuration
    - Les différences entre Python 2 et 3
2. Les fondamentaux de Python
    - Les types de données : entiers, flottants, chaînes de caractères
    - Les opérateurs arithmétiques, de comparaison, d'assignation et de logique
    - Les structures de contrôle de flux : if-else, for, while
    - Les fonctions : définition, arguments et valeur de retour
```

Introduction

Le langage de programmation Python a été créé en 1989 par Guido van Rossum, aux Pays-Bas. Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus dont G. van Rossum est fan. La première version publique de ce langage a été publiée en 1991. La dernière version de Python est la version 3. Plus précisément, la version 3.12 a été publiée en 10 Octobre 2023.

Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

- Il est multiplateforme.
- Il est gratuit.
- C'est un langage de haut niveau. Il demande relativement peu de connaissance sur le fonctionnement d'un ordinateur pour être utilisé.
- C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.
- Il est orienté objet. C'est-à-dire qu'il est possible de concevoir en Python des entités qui miment celles du monde réel (une voiture, un conducteur, une piste, etc.) avec un certain nombre de règles de fonctionnement et d'interactions.
- Il est relativement simple à prendre en main.
- Enfin, il est très utilisé en sience de données et plus généralement en analyse de données.
   Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, depuis l'enseignement secondaire jusqu'à l'enseignement supérieur.

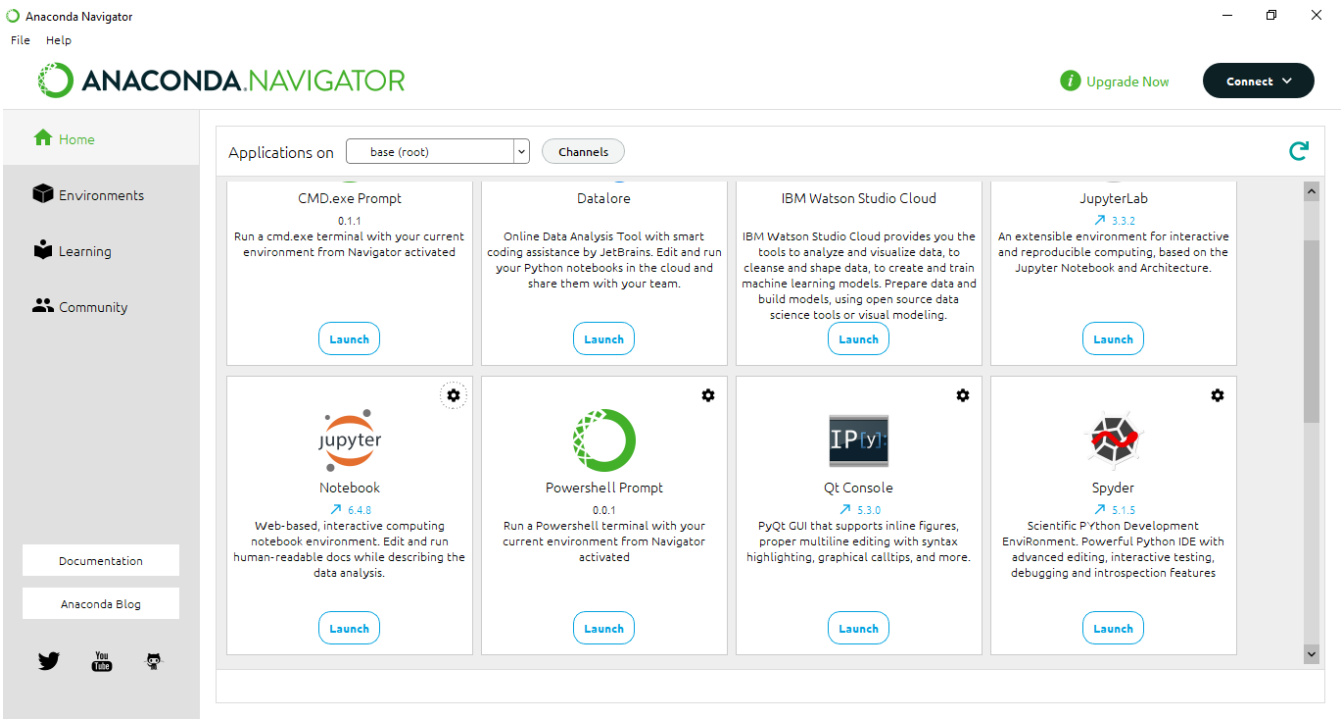
Installez Python via Anaconda

L'installation d'un environnement Python complet peut-être une vraie galère. Déjà, il faut télécharger Python et l'installer. Par la suite, télécharger un à un les packages dont on a besoin. Parfois, le nombre de ces librairies peut-être grand.

Par ailleurs, il faut s'assurer de la compatibilité entre les versions des différentes packages qu'on a à télécharger.

Anaconda est une distribution scientifique de Python : c'est-à-dire qu'en installant Anaconda, vous installerez Python, Jupyter Notebook et des dizaines de packages scientifiques, dont certains indispensables à l'analyse de données !

Pour commencer, téléchargez la distribution Anaconda correspondant à votre système d'exploitation https://www.anaconda.com/products/distribution



Il existe plusieurs applications dans Anaconda Navigator tel que :

- JupyterLab
- JupyterNotebook
- Spyder
- Pycharm
- VSCode
- Orane 3 APP
- RStudio
- Anaconda powerShell

C'est quoi Jupyter Notebook

Jupiter Notebook est une plateforme Web open source qui facilite la création et le partage des documents qui contiennent du code. Jupiter Notebook permet de gérer des fichiers, des modules et aussi la présentation du travail. C'est un produit phare open-source de Projet Jupyter et est largement utilisé en science des données.

Il y a deux manières de lancer Jupiter Notebook.

- **Avec la console Anaconda prompt :** sur la barre de recherche, taper “Anaconda prompt” puis taper « jupiter notebook” :
- **Méthode directe :** taper directement dans la barre de recherche : “Anacondanavigator”. Une fois que l’interface est affichée, cliquez sur launch de l'application Jupyter.

Version de python

In [1]:
```
from platform import python_version
python_version()
```

Out[1]:
```
'3.9.12'
```

Les Calculs

Une des premières fonctionnalités d'un interpréteur est de faire des calculs:

In [2]:
```
1+2
```

Out[2]:
```
3
```

Tous les opérateurs sont utilisables:

In [3]:
```
10 -2
```

Out[3]:
```
8
```

In [4]:
```
5*6
```

Out[4]:
```
30
```

Les commentaires

Les commentaires peuvent être utilisés pour expliquer le code pour plus de lisibilité.

In [5]:
```
# Commentaire sur une seule ligne
val = 10
```

In [6]:
```
# Commentaire
# sur plusieurs
# lignes
val = 10
```

In [7]:
```
'''
Commentaire
sur plusieurs
lignes
'''
val = 10
```

In [8]:
```
"""
```

```
Commentaire
sur plusieurs
lignes
"""
val = 10
```

Identifiant

Un identifiant est un nom donné à des entités telles que des classes, des fonctions, des variables, etc. Il permet de différencier une entité d'une autre.

L'identifiant :

- Ne peut pas commencer par un chiffre
- Ne peut pas utiliser de symboles spéciaux
- Les mots clés ne peuvent pas être utilisés comme identifiants

In [9]:
```
1var = 10
```

```
  Input In [9]
    1var = 10
     ^
SyntaxError: invalid syntax
```

In [10]:
```
val@ = 35
```

```
  Input In [10]
    val@ = 35
         ^
SyntaxError: invalid syntax
```

In [11]:
```
import = 125
```

```
  Input In [11]
    import = 125
           ^
SyntaxError: invalid syntax
```

Python 3.9 contient 36 mots clés réservés. Ces mots ne peuvent pas être utilisés comme identifiant:

In [12]:
```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'br
eak', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'fr
om', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'r
aise', 'return', 'try', 'while', 'with', 'yield']
```

Les Variables

Une variable est une zone de la mémoire de l'ordinateur dans laquelle une valeur est stockée. Aux yeux du programmeur, cette variable est définie par un nom (identificateur), alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire.

Une variable python a les caractéristiques suivantes :

- Un pointeur vers une zone mémoire.

- Permettant de stocker une ou plusieurs données.
- Pouvant avoir plusieurs valeurs différentes dans un programme.
- Pas de type à préciser pour les variables.
- Une variable peut changer de type.
- Impossible de concaténer une chaîne et un nombre sans conversion.

En Python, la déclaration d'une variable et son initialisation (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps. Pour vous en convaincre, testez les instructions suivantes après avoir lancé l'interpréteur :

In [13]:
```
#Pour déclarer une variable
x = 2

#Pour afficher une variable
x
```

Out[13]:
```
2
```

In [14]:
```
#Pour utiliser une variable
y = x + 5
y
```

Out[14]:
```
7
```

In [15]:
```
#Pour déclarer plusieurs variables et leur affecter une même valeur
a = b = c = 1

#Pour afficher les valeurs respectives des variables précédentes
a,b,c
```

Out[15]:
```
(1, 1, 1)
```

In [16]:
```
# Pour déclarer plusieurs variables et leur affecter des valeurs différentes
d, e, f = 1, 2, "bonjour"
d,e,f
```

Out[16]:
```
(1, 2, 'bonjour')
```

Typage des variables

Comme le typage est dynamique en python, le type n'est pas précisé explicitement, il est implicitement liée à l'information manipulée. Par exemple, en écrivant, x=3.4, on ne précise pas le type de la variable x mais il est implicite car x reçoit une valeur réelle : x est de type réel ou float en python.

Le langage python possède un certain nombre de types de variables déjà définis ou types fondamentaux à partir desquels il sera possible de définir ses propres types. Voici une liste de type de variable:

- **Les integer ou nombres entiers :** comme son nom l'indique un entier est un chiffre sans décimales.
- **Les float ou nombre à virgules :** exemple : 1.5
- **Les nombres complexes**
- **Les strings ou chaine de caractères :** pour faire simple tout ce qui n'est pas chiffre.
- **Booléen :** bool

- **Binaires :** bytes, bytearray, memoryview

Pour déterminer le type d'une variable en utilise la fonction **type()**

Les types de base en Python :

Valeur	Type Python	Exemple
ϵ Z	Int	12,+20,-20
ϵ IR	Float	12., 20.0, 1.23e-6
ϵ C	Complex	2+9j, 2+9J)
caractère	Char	'1','f'
Valeur logique	Bool	True, False

In [17]:
```
x=2.5
type(x)
```

Out[17]:
```
float
```

In [18]:
```
x = 'bonjour'
type(x)
```

Out[18]:
```
str
```

In [19]:
```
x = True
type(x)
```

Out[19]:
```
bool
```

In [20]:
```
x= 5 + 3j
type(x)
```

Out[20]:
```
complex
```

In [21]:
```
x = b'Python'
type(x)
```

Out[21]:
```
bytes
```

Pour tester si une valeur est de certain type en utilise la fonction isinstance()

In [22]:
```
isinstance(2+5j, complex)
```

Out[22]:
```
True
```

In [23]:
```
isinstance(2.5, int)
```

Out[23]:
```
False
```

Interaction : Lecture/Ecriture

En Python, nous utilisons pour la lecture et l'écriture les fonctions :

- Pour afficher, on utilise la fonction print().

- Pour récupère ce qu'a tapé l'utilisateur sur le clavier, enutilise la fontion input(). cela renverra une chaîne en sortie

Généralement, les codes Python réels n'ont pas besoin de lire les entrées du clavier

In [24]:
```
val = "Salam Python"
print(val)
```

```
Salam Python
```

In [25]:
```
print("Salam", "comment vas-tu?", sep="---")
```

```
Salam---comment vas-tu?
```

In [26]:
```
x = 3.95
print("x =", x, type(x))
```

```
x = 3.95 <class 'float'>
```

In [27]:
```
x = int(3.5)
y = float(3)
z = int("3")
print("x:", type(x), " y:", type(y), " z:", type(z))
```

```
x: <class 'int'>  y: <class 'float'>  z: <class 'int'>
```

In [28]:
```
val = input("donnez la valeur de val :")
print(val,type(val))
```

```
donnez la valeur de val :2023
2023 <class 'str'>
```

In [29]:
```
val = int(input("donnez la valeur de val :"))
print(val,type(val))
```

```
donnez la valeur de val :2023
2023 <class 'int'>
```

Opérateurs de base

La liste des opérateurs qui s'appliquent aux nombres réels et entiers suit. Les trois premiers résultats << >>, | et & s'expliquent en utilisant la représentation en base deux. 8 << 1 s'écrit en base deux 100 << 1 = 1000, ce qui vaut 16 en base décimale : les bits sont décalés vers la droite ce qui équivaut à multiplier par deux. De même, 7 & 2 s'écrit 1011 & 10 = 10, qui vaut 2 en base décimale. Les opérateurs <<, >>, |, & sont des opérateurs bit à bit, ils se comprennent à partir de la représentation binaire des nombres entiers.

Opérateur	Signification	Exemple
<< >>	décalage à gauche, à droite	x = 8 << 1
|	opérateur logique ou bit à bit	x = 8 | 1
&	opérateur logique et bit à bit	x = 11 & 2
^	opérateur logique Ou exclusif bit à bit	x= 2^3
+ -	addition, soustraction	x = y + z
+= -=	addition ou soustraction puis affectation	x += 3
* /	multiplication, division	x = y * z
//	division entière, le résultat est de type réel si l'un des nombres est réel	x = y // 3

%	reste d'une division entière (modulo)	x = y % 3
*= /=	multiplication ou division puis affectation	x *= 3
**	puissance (entière ou non, racine carrée = ** 0.5)	x = y ** 3

In [30]:
```
x = 8 << 1
x
```

Out[30]:
```
16
```

In [31]:
```
bin(x)
```

Out[31]:
```
'0b10000'
```

In [32]:
```
11 & 2
```

Out[32]:
```
2
```

In [33]:
```
2^3
```

Out[33]:
```
1
```

In [34]:
```
x = 10**3
x
```

Out[34]:
```
1000
```

In [35]:
```
x = 10 / 3
x
```

Out[35]:
```
3.3333333333333335
```

In [36]:
```
x = 10 // 3
x
```

Out[36]:
```
3
```

In [37]:
```
x = 10 % 3
x
```

Out[37]:
```
1
```

bases de numérotation et codage(conversions)

Base	Exemple	Résultat
bin	bin(2)	0b10
oct	oct(12)	0o14
hex	hex(15)	0xf
int	int(0b10)	2
float	float("12")	12.0
ord	ord('a')	97 (code ASCII de a)

chr	chr(97)	'a'

In [38]:
```
ord('a')
```

Out[38]:
```
97
```

In [39]:
```
hex(12)
```

Out[39]:
```
'0xc'
```

In [40]:
```
ord('C')
```

Out[40]:
```
67
```

In [41]:
```
chr(67)
```

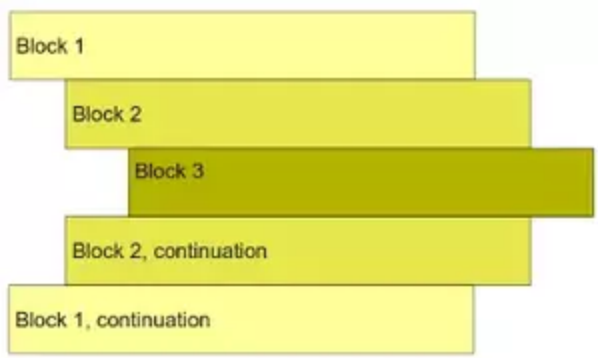
Out[41]:
```
'C'
```

Indentation

Une indentation représente un ou plusieurs espaces au début d'une ligne de code.

Il est recommandé avec Python d'indenter son code avec une tabulation (équivalent à 4 espaces).

L'indentation est primordiale avec Python car elle sert à déterminer les blocs qui constituent votre code là où d'autres langages privilégient par exemple les accolades ({ }) pour spécifier ces blocs.



Si l'indentation n'est pas correcte, nous nous retrouverons avec **IndentationError** Erreur.

In [42]:
```
if(5>2):
    print("5 est supérieur que 2")
```

```
5 est supérieur que 2
```

In [43]:
```
if(5>2):
print("5 est supérieur que 2")
```

```
  Input In [43]
    print("5 est supérieur que 2")
    ^
IndentationError: expected an indented block
```

Les conditions (if/elif/else)

Les tests sont un élément essentiel à tout langage informatique si on veut lui donner un peu de complexité

car ils permettent à l'ordinateur de prendre des décisions. Pour cela, Python utilise l'instruction if ainsi qu'une comparaison

In [44]:
```
x = 2
if x == 2:
    print (" Le test est vrai !")
```

```
 Le test est vrai !
```

In [45]:
```
x = 2
if x == 2:
    print (" Le test est vrai !")
else :
    print (" Le test est faux !")
```

```
 Le test est vrai !
```

In [46]:
```
a = 5
if a > 5:
    a = a + 1
    print(a)
elif a == 5:
    a = a + 1000
    print(a)
else:
    a = a - 1
    print(a)
```

```
1005
```

In [47]:
```
# test imbriqué
a = 10.
if a > 0:
    print( 'a est strictement positif')
    if a >= 10:
        print( 'a est un nombre')
    else:
        print( 'a est un chiffre')
    a += 1
elif a != 0:
    print( 'a est strictement négatif')
else:
    print('a est nul')
```

```
a est strictement positif
a est un nombre
```

Boucle FOR

Quand on sait combien de fois doit avoir lieu la répétition, on utilise généralement **Boucle bornée** for.
La boucle for permet de faire des itérations sur un élément, comme une chaine de caractères par exemple ou une liste .

In [48]:
```
v = "Salam GLCC"
for lettre in v:
    print(lettre)
```

```
S
a
l
a
m
```

```
G
L
C
C
```

Range

Il est possible de créer une boucle facilement avec range :
La fonction **range()** renvoie une séquence de nombres, commençant par 0 par défaut, et incrémentée de 1 (par défaut), et s'arrête avant un nombre spécifié.

la syntaxe de Range() :

range(start, stop, step)

In [49]:
```
x = range(6)
for n in x:
    print(n)
```

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

In [50]:
```
x = range(3, 6)
for n in x:
    print(n)
```

```
3
4
5
```

In [51]:
```
x = range(3, 20, 2)
for n in x:
    print(n)
```

```
3
5
7
9
11
13
15
17
19
```

In [52]:
```
# le cas d'une liste
animaux = [" girafe ", " tigre ", " singe ", " souris "]
for animal in animaux :
    print ( animal )
```

```
 girafe
 tigre
 singe
 souris
```

In [53]:
```
for i in range(len(animaux)):
    print("l'indice", i, "et celui de l'animal", animaux[i])
```

```
l'indice 0 et celui de l'animal  girafe
l'indice 1 et celui de l'animal  tigre
l'indice 2 et celui de l'animal  singe
l'indice 3 et celui de l'animal  souris
```

Boucle While

Le mot-clé while signifie tant que en anglais. Le corps de la boucle sera répété tant que une condition est vraie.

Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté. Si la condition reste toujours vraie, alors le corps de la boucle est répété indéfiniment.

In [54]:
```
i = 0
while i < 4:
    print("i a pour valeur", i)
    i = i + 1
```

```
i a pour valeur 0
i a pour valeur 1
i a pour valeur 2
i a pour valeur 3
```

Les instructions **break** et **Continue**

L'instruction **break** permet de « casser » l'exécution d'une boucle (while ou for). Elle fait sortir de la boucle et passer à l'instruction suivante.

In [55]:
```
for i in range(10):
    print("debut iteration", i)
    print("bonjour")
    if i == 2:
        break
    print("fin iteration", i)
print("apres la boucle")
```

```
debut iteration 0
bonjour
fin iteration 0
debut iteration 1
bonjour
fin iteration 1
debut iteration 2
bonjour
apres la boucle
```

In [ ]:
```
while True:
    n = int(input("donnez un entier > 0 : "))
    print("vous avez fourni", n)
    if n > 0:
        break
print("reponse correcte")
```

L'instruction **continue** permet de passer prématurément au tour de boucle suivant. Elle fait continuer sur la prochaine itération de la boucle.

In [56]:
```
for i in range(4):
    print("debut iteration", i)
    print("bonjour")
    if i < 2:
        continue
    print("fin iteration", i)
print("apres la boucle")
```

```
debut iteration 0
bonjour
```

```
debut iteration 1
bonjour
debut iteration 2
bonjour
fin iteration 2
debut iteration 3
bonjour
fin iteration 3
apres la boucle
```

Les structures de données

Les structures de données sont des moyens spécifiques d'organiser et de stocker des données afin qu'elles puissent être consultées et travaillées de manière efficace. Les structures de données définissent la relation entre les données et les opérations pouvant être effectuées dessus.

Python dispose de plusieurs types prédéfinis de structures de données:

1. Les chaînes de caractères
2. Les listes
3. Les Tuples
4. Les dictionnaires
5. Les ensembles

1. Les chaînes de caractères

Les chaînes ne sont rien d'autre que du texte simple. En Python, nous déclarons des chaînes entre " et " ou 'et' ou '''et''' ou " " " et " " ".

In [57]:
```
str1 = 'SALAM PYTHON'
print(str1)
```

```
SALAM PYTHON
```

In [ ]:
```
syntaxe1 = "Première forme avec un retour à la ligne"
print(syntaxe1)
```

In [58]:
```
syntaxe2 = str("Deuxième forme. Chaîne brute\n sans retour à la ligne")
print(syntaxe2)
```

```
Deuxième forme. Chaîne brute
 sans retour à la ligne
```

In [59]:
```
syntaxe3 = '''Troisièm forme de Chaîne'''
print(syntaxe3)
```

```
Troisièm forme de Chaîne
```

In [60]:
```
syntaxe4 = """
quatrième forme
multi-lignes
très utile pour
la documentation
"""
print(syntaxe4)
```

```
quatrième forme
multi-lignes
très utile pour
```

la documentation

In [61]:
```
guillemets = "L'eau vive"
guillemets
```

Out[61]:
```
"L'eau vive"
```

In [62]:
```
apostrophes = 'Il a dit "gère !"'
apostrophes
```

Out[62]:
```
'Il a dit "gère !"'
```

In [63]:
```
# la longueur d'une chaîne
s = "abcd"
len(s)
```

Out[63]:
```
4
```

In [64]:
```
#Concaténatione de deux chaîn
s1 = "abc"
s2 = "efg"
s3 = s1 + " " + s2
s3
```

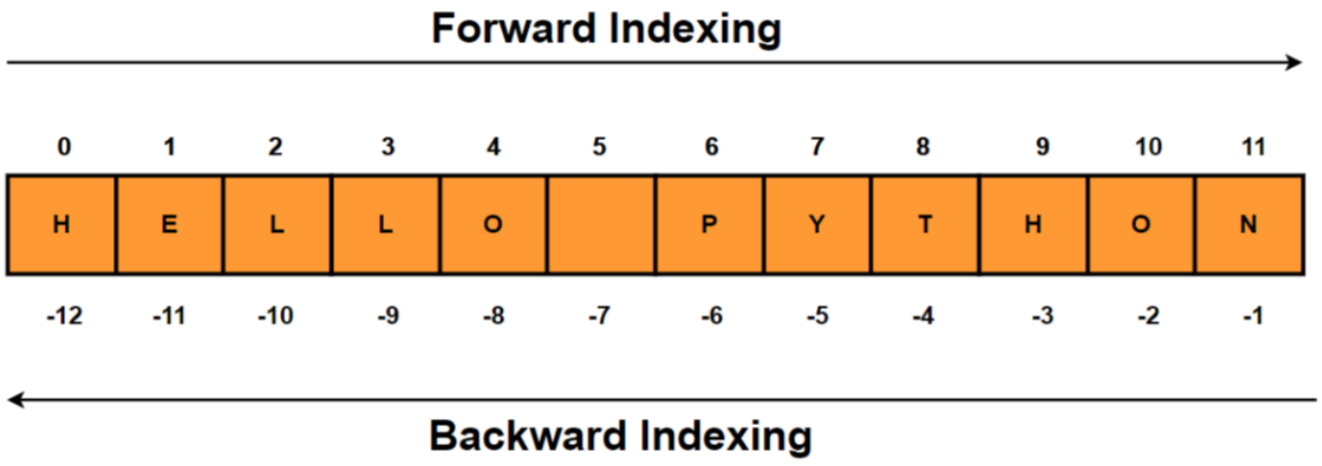
Out[64]:
```
'abc efg'
```

In [65]:
```
# répétition d'une chaîne
s4 = 'python '
s5 = s4 * 3
print(s5)
```

```
python python python
```

Indexation des Chaînes

Pour indexer une chaîne, on utilise l'opérateur [ ] dans lequel l'index, un entier signé qui commence à 0, indique la position d'un caractère



In [66]:
```
str ="Salam Python"
str
```

Out[66]:
```
'Salam Python'
```

In [67]:
```
str[0]
```

Out[67]:
```
'S'
```

In [68]:
```
str[len(str)-1]
```

Out[68]:
```
'n'
```

In [69]:
```
str[-1]
```

Out[69]:
```
'n'
```

In [70]:
```
str[5]
```

Out[70]:
```
' '
```

In [71]:
```
str[-5]
```

Out[71]:
```
'y'
```

Tranchage d'une chaîne (String Slicing)

On suppose que i et j sont positifs et L une chaîne de langueur N.
– La tranche L[i:j] est la sous-liste [L[i],L|i+1],...,L[j-1]]
– La tranche L[i:] est la sous-liste [L[i],L|i+1],...,L[N-1]]
– La tranche L[:j] est la sous-liste [L[0],L|1],...,L[j-1]]
– La tranche L[i:j:p] est la sous-liste de la tranche L[i:j] constituée des éléments L[k] tel que : [L[i],L|i+p],L[i+2p]...]
– En particulier, L[::-1] est une copie en miroir de L.

In [72]:
```
str[6:10]
```

Out[72]:
```
'Pyth'
```

In [73]:
```
str[6:]
```

Out[73]:
```
'Python'
```

In [74]:
```
str[:5]
```

Out[74]:
```
'Salam'
```

In [75]:
```
str[2:12:2]
```

Out[75]:
```
'lmPto'
```

In [76]:
```
str[::-1]
```

Out[76]:
```
'nohtyP malaS'
```

Méthodes associées aux chaînes de caractères

Voici quelques méthodes spécifiques aux objets de type string :

In [77]:
```
#Les méthodes .lower() et .upper() renvoient un texte en minuscule et en majuscule respe
x = "girafe "
x.upper ()
```

Out[77]: 'GIRAFE '

In [78]:
```
'TIGRE'.lower ()
```

Out[78]: 'tigre'

In [79]:
```
#Pour mettre en majuscule la première lettre seulement,
x [0].upper () + x [1:]
```

Out[79]: 'Girafe '

In [80]:
```
#ou plus simplement utiliser la méthode adéquate :
x. capitalize ()
```

Out[80]: 'Girafe '

- **istitle()** : retourne True si seule la première lettre de chaque mot de la chaîne est en majuscule.
- **isalnum(), isalpha(), isdigit() et isspace()** : retournent True si la chaîne ne contient respectivement que des caractères alphanumériques, alphabétiques, numériques ou des espaces.
- **startswith(prefix)et endswith(suffix)** : testent si la sous chaîne définie par start et stop commence respectivement par préfix ou finit par suffix.
- **lower(), upper(), capitalize() et swapcase()** : retournent respectivement une chaîne en minuscule, en majuscule, en minuscule commençant par une majuscule, ou en casse inversée.
- **expandtabs([tabsize])** : remplace les tabulations par tabsize espaces (8 par défaut).
- **center(width[, fillchar]), ljust(width[, fillchar]) et rjust(width[, fillchar])** : retournent respectivement une chaîne centrée, justifiée à gauche ou à droite, complétée par le caractère fillchar (ou par l'espace par défaut).
- **zfill(width)** : complète ch à gauche avec des 0 jusqu'à une longueur maximale de width.
- **strip([chars]), lstrip([chars]) et rstrip([chars])** : suppriment toutes les combinaisons de chars (ou l'espace par défaut) respectivement au début et en fin, au début, ou en fin d'une chaîne.
- **find(sub[, start[, stop]])** : renvoie l'index de la chaîne sub dans la sous-chaîne start à stop, sinon renvoie -1. rfind() effectue le même travail en commençant par la fin. index() et rindex() font de même mais produisent une erreur (exception) si la chaîne n'est pas trouvée.
- **replace(old, new[, count])** : remplace count instances (toutes par défaut) de old par new.
- **split(seps[, maxsplit])** : découpe la chaîne en maxsplit morceaux (tous par défaut). rsplit() effectue la même chose en commençant par la fin et striplines() effectue ce travail avec les caractères de fin de ligne.
- **join(seq)** : concatène les chaînes du conteneur seq en intercalant entre chaque élément la chaîne sur laquelle la méthode est appliquée.

In [81]:
```
s = "cHAise basSe"
s.startswith('cH') , s.endswith('aSSe')
```

Out[81]: (True, False)

In [82]:
```
s.lower(), s.upper(), s.capitalize(), s.swapcase()
```

Out[82]: ('chaise basse', 'CHAISE BASSE', 'Chaise basse', 'ChaISE BASsE')

In [83]:
```
s.center(20, '-'), s.rjust(20, '@')
```

Out[83]: ('----cHAise basSe----', '@@@@@@@@cHAise basSe')

In [84]:
```
s.zfill(20)
```

Out[84]:
```
'00000000cHAise basSe'
```

In [85]:
```
s.strip('ce')
```

Out[85]:
```
'HAise basS'
```

Exercice

Ecrire un programme Python qui lit une chaîne de caractères CH au clavier et qui compte les occurrences des lettres de l'alphabet en ne distinguant pas les majuscules et les minuscules. Utiliser un tableau ABC de dimension 26 pour mémoriser le résultat.

Exemple :
Entrez une ligne de texte (max. 100 caractères) :
master
La chaîne "master" contient :
1 fois la lettre 'a'
1 fois la lettre 'e'
1 fois la lettre 'm'
1 fois la lettre 'r'
1 fois la lettre 's'
1 fois la lettre 't'

In [87]:
```
alphabet="abcdefghijklmnopqrstuvwxyz"
ch=input("donner une valeur ")
res=[]
for i in range(len(alphabet)):
    res.append(0)
    for j in range(len(ch)):
        if(alphabet[i]==ch[j]):
            res[i]=res[i]+1
res
```

Out[87]:
```
donner une valeur bonjour
[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0]
```

2. Liste

Une liste est une collection ordonnée et modifiable d'éléments éventuellement hétérogènes.
Les éléments d'une liste sont séparés par des virgules, et entourés de crochets.

Exemples :

In [88]:
```
l1 = [1,2,3,4,5,6] # e n t i e r s
l1
```

Out[88]:
```
[1, 2, 3, 4, 5, 6]
```

In [89]:
```
l2 = [3.14, 0.693, 2.78] # f l o t t a n t s
l2
```

Out[89]:
```
[3.14, 0.693, 2.78]
```

In [90]:
```
l3 = [True, True, False] # b o o l é en s
l3
```

Out[90]:
```
[True, True, False]
```

In [91]:
```
l4 = ['a', 'bra', 'ca', 'da', 'bra'] # chaînes de caractères
l4
```

Out[91]:
```
['a', 'bra', 'ca', 'da', 'bra']
```

In [92]:
```
l5 = [1, 1.609, 'bonjour', None]
l5
```

Out[92]:
```
[1, 1.609, 'bonjour', None]
```

In [93]:
```
range(10), range(2, 10, 2)
```

Out[93]:
```
(range(0, 10), range(2, 10, 2))
```

In [94]:
```
#list vide
list1 = []
print(type(list1))
```

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

In [95]:
```
L = ['Dans', 'python', 'tout', 'est', 'objet']
T = L
T[4] = 'bon'
T
```

Out[95]:
```
['Dans', 'python', 'tout', 'est', 'bon']
```

In [96]:
```
liste_1 = [1, 2, 3]
listes = [liste_1, [4, 5], "abcd"]
listes
```

Out[96]:
```
[[1, 2, 3], [4, 5], 'abcd']
```

In [97]:
```
L = [21, 14, 10, 6, 98, 5]
len(L)
```

Out[97]:
```
6
```

In [98]:
```
L[-2]
```

Out[98]:
```
98
```

In [99]:
```
L[1:4]
```

Out[99]:
```
[14, 10, 6]
```

In [100…
```
L[:3]
```

Out[100]:
```
[21, 14, 10]
```

In [101…
```
L[-3:]
```

Out[101]:
```
[6, 98, 5]
```

In [102…
```
L[:]
```

Out[102]:
```
[21, 14, 10, 6, 98, 5]
```

Ajouter, supprimer et modifier des éléments

In [103…
```
List =['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
# Ajouter un élément à la fin de la liste
List.append('nine')
List
```

Out[103]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [104…
```
# Ajouter un élément à l'emplacement d'index 9
List.insert(9,'ten')
List
```

Out[104]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [105…
```
List.insert(1,'ONE')
List
```

Out[105]:
```
['one',
 'ONE',
 'two',
 'three',
 'four',
 'five',
 'six',
 'seven',
 'eight',
 'nine',
 'ten']
```

In [106…
```
List.remove('ONE')
List
```

Out[106]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [107…
```
List.pop()
List
```

Out[107]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [108…
```
del List[7]
List
```

Out[108]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'nine']
```

In [109…
```
List[0] = 0
List[1] = 1
List[2] = 2
List
```

Out[109]:
```
[0, 1, 2, 'four', 'five', 'six', 'seven', 'nine']
```

In [110…
```
List.reverse()
List
```

Out[110]:
```
['nine', 'seven', 'six', 'five', 'four', 2, 1, 0]
```

In [111…
```
List.clear()
List
```

```
[]
```

Out[111]:

In [112…

```
del List
List
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [112], in <cell line: 2>()
      1 del List
----> 2 List

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