Rapport du TP2
Programmation concurrente

Suivi par Mr Ilyes MEHADDI

Nom : TACHOUR
Prénom : Omar
N étudiant : 16706265
Année universitaire : 2020/2021

1/ Définition du Python:

- Python est le langage de programmation le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science.
- Créé en 1991, apparu à l'époque comme une façon d'automatiser les éléments les plus ennuyeux de l'écriture de scripts ou de réaliser rapidement des prototypes d'applications.
- Depuis quelques années, toutefois, il s'est hissé parmi les plus utilisés dans le domaine du développement de logiciels, de gestion d'infrastructure et d'analyse de données. Il s'agit d'un élément moteur de l'explosion du Big Data.

2/ Rapport sur le tutoriel :

Le tutoriel contient toutes les méthodes prédéfinies qui peuvent être appliquées sur des nombres, chaînes de caractères, listes, et tous les outils nécessaires pour programmer en python.

1. Introduction informelle à python :

→ Pour écrire un commentaire il suffit juste de commencer la saisie par un #
→ Pour déclarer une variable de type nombre, on doit écrire dans la console python le nom de variable , = , nombre.
→ Pour déclarer une variable de type char ou string, on écrit le nom du variable = « chaîne de caractères »

```
# c'est le premier commentaire
spam  =  1   # et c'est le deuxième commentaire
         # ... et maintenant un troisième!
text  =  "# Ce n'est pas un commentaire car il est entre guillemets."
```

- **Utiliser python comme une calculatrice :**

→ L'interpréteur agit comme une calculatrice, on peut réaliser n'importe quelle fonction arithmétique (+ , - , * , / , %(modulo) , //(Rejet de division, partie fractionnaire).

```
>>> 2  +  2
4
>>> 50  -  5 * 6
20
>>> ( 50  -  5 * 6 )  /  4
5,0
>>> huit  /  5   # division retourne toujours un nombre à virgule
flottante
1,6

>>> 17  /  3   # retourne la division classique un flotteur
5,666666666666667
>>>
>>> 17  //  3   # rejets de division étage , la partie fractionnaire
5
>>> 17  % de  3   #% le rendement de l' opérateur Le reste de la
division
2
>> > 5  *  3  +  2   # résultat * diviseur + reste
17
```

→ La puissance 2 avec l'opérateur **

```
>>> 5  **  2   # 5 au carré
25
>>> 2  **  7   # 2 à la puissance
7128
```

x=13.5
int(x) #pour rendre le nombre x à un entier
13
float(13) #convertir un nombre à un float
13.0
abs(-5) #renvoie une valeur absolue d'un nombre
5

- **Les opérations sur les bits des nombres entiers :**

x | y → *ou* <*or*> binaire de *x* et *y*
x ^ y → *ou* <*or*> exclusive binaire de *x* et *y*
x & y → *et binaire* <*and*> de *x* et *y*
x << n → *x* décalé vers la gauche de *n* bits
x >> n → *x* décalé vers la droite de *n* bits
~*x* → les bits de *x*, inversés

→ Pour faire appel à une variable il faut la déclarer avant.

→ En mode interactif, la dernière expression affichée est affectée à la variable `_` .

```
>>> taxe  =  12,5  /  100
>>> prix  =  100,50
>>> prix  *  taxe
12,5625
>>> prix  +  _
113,0625
>>> rond ( _ ,  2 )
113,06
```

- **Les chaînes de caractères :**

→ Les chaînes de caractères peuvent être écrites entre guillemets simples '…' ou doublets « ... ».

```
>>> 'spam eggs'   # guillemets simples
'spam eggs'
>>> 'doesn \' t '   # use \' pour échapper aux guillemets simples ...
"ne"
>>> "pas"   #. ..ou utilisez des guillemets doubles à la place
"ne"
>>> "" Oui, "dit-il."
«Oui», dit-il.
>>> " \" Oui, " dit-il."
«Oui», dit-il.
```

```
>>> '"N \' est pas ," dit-elle. '
«Non,» dit-elle.
```

→ En mode interactif, l'interpréteur affiche les chaînes de caractères entre guillemets et en protégeant les guillemets et autres caractères spéciaux avec des antislash. Bien que cela puisse paraître différent de ce qui a été donné (les guillemets peuvent changer) La chaîne est affichée entre guillemets si elle contient un guillemet simple mais aucun guillemet, sinon elle est affichée entre guillemets simples.

```
>>> '"N \' est pas ," dit-elle. '
«N \ 'est pas,» elle a dit.
>>> print ( '"N \' est pas," dit- elle. ' )
"N'est pas," dit-elle.
>>> s  =  'Première ligne. \ n Deuxième ligne. '   # \ n signifie
nouvelle ligne
>>> s   # sans print (), \ n est inclus dans la sortie
'Première ligne. \ nDeuxième ligne.'
>>> print ( s )   # avec print (), \ n produit une nouvelle ligne
Première ligne.
Deuxième ligne.
```

→ *Pour sauter une ligne on doit taper '\n'.*

```
>>> print ( 'C: \ some \ n ame' )  # here \ n signifie nouvelle ligne!
C: \ some
ame
>>> print ( r 'C: \ some \ name' )   # notez le r avant la citation
C: \ some \ name
```

→ Si on utilise les triples guillemets simples ou doubles les retours à la ligne et les tabulations se font automatiquement. Et si on veut les empêcher on ajoute \ à la fin de la ligne

```
print ( "" " \
Utilisation: thingy [OPTIONS]
     -h Afficher ce message d'utilisation
     -H hostname Nom d'hôte auquel se connecter
" "" )

Utilisation: thingy [OPTIONS]
     -h Afficher ce message d'utilisation
     -H nom d'hôte Nom d'hôte auquel se connecter
```

→ *Les chaînes de caractères peuvent être concaténées avec + ou répétées avec * et elles peuvent être aussi concaténées une fois qu'elles sont cote à cote.*

```
>>> # 3 fois 'un', suivi de 'ium'
>>> 3  *  'un'  +  'ium'
```

```
'unununium'

>>> 'Py' 'thon'

'Python'
```

→ *Elles peuvent être aussi indexées.*
→ *On peut accéder à un élément dans une chaîne de caractères par son indice.*

```
>>> mot  =  'Python'
>>> mot [ 0 ]   # caractère en position 0
'P'
>>> mot [ 5 ]   # caractère en position 5
'n'

>>> mot [ - 1 ]   # dernier caractère
'n'
>>> mot [ - 2 ]   # avant-dernier caractère
'o'
>>> mot [ - 6 ]
'P'

>>> mot [ 0 : 2 ]  # caractères de la position 0 (inclus) à 2 (exclus)
'Py'
>>> mot [ 2 : 5 ]  # caractères de la position 2 (inclus) à 5 (exclus)
'tho'

>>> mot [: 2 ]  +  mot [ 2 :]
'Python'
>>> mot [: 4 ]  +  mot [ 4 :]
'Python'

>>> 'J'  +  mot [ 1 :]
'Jython'
>>> mot [: 2 ]  +  'py'
'Pypy'
```

→ on peux compter la longueur d'une chaîne du caractères.

```
>>> s  =  'supercalifragilisticexpialidocious'
>>> len ( s )
34
```

- **Quelques fonctions prédéfinies sur les chaînes de caractères :**

→ str.capitalize() :

Renvoie une copie de la chaîne avec son premier caractère en majuscule et le reste en minuscule.

```
string = "python is AWesome."
capitalized_string = string.capitalize()

print('Old String: ', string)
Old String: python est génial

print('Capitalized String:', capitalized_string)
Chaîne en majuscules: Python est génial
```

→ str.casefold() :

Renvoie une copie *casefolded* de la chaîne. Les chaînes *casefolded* peuvent être utilisées dans des comparaison insensibles à la casse.

```
String = "PYTHON IS AWESOME",
# print lowercase string
print("Lowercase string:", string.casefold())
Lowercase string: python is awesome
```

→ str.center(width[, fillchar]) :

Donne la chaîne au centre d'une chaîne de longueur width. Le remplissage est fait enutilisant l'argument fillchar (qui par défaut est un espace ASCII). La chaîne d'origine est renvoyée si width est inférieur ou égale à len(s).

```
#!/usr/bin/python

str = "this is string example....wow!!!"
print ("str.center(40, 'a') : ", str.center(40, 'a'))

str.centerc40, 'a') :  aaaathis is string example....wow!!!aaaa
```

→ str.count(sub[, start[, end]]) :

Donne le nombre d'occurrences de sub ne se chevauchant pas dans le range [start, end]. Les arguments facultatifs start et end sont interprétés comme pour des slices.

```
# define string
string = "Python is awesome, isn't it?"
substring = "is"

count = string.count(substring)

# print count
print("The count is:", count)
The count is: 2
```

→ str.encode() :

La méthode string encode () renvoie la version codée de la chaîne donnée.

```
# unicode string

string = 'pythön!'
# print string
print('The string is:', string)
The string is: pythön!

# default encoding to utf-8
string_utf = string.encode()
# print result
print('The encoded version is:', string_utf)
The encoded version is: b'pyth\xc3\xb6n!'
```

→ str.endswith(suffix[, start[, end]]) :

Donne `True` si la chaîne se termine par *suffix*, sinon `False`. *suffix* peut aussi être un tuple de suffixes à rechercher. Si l'argument optionnel *start* est donné, le test se fait à partir de cette position. Si l'argument optionnel *end* est fourni, la comparaison s'arrête à cette position.

```
text = "Python is easy to learn."

result = text.endswith('to learn')
# returns False
print(result)
False
result = text.endswith('to learn.')
# returns True
print(result)
```

```
True
result = text.endswith('Python is easy to learn.')
# returns True
print(result)
True
```

str.expandtabs(tabsize=8)

Donne une copie de la chaîne où toutes les tabulations sont remplacées par un ou plusieurs espaces, en fonction de la colonne courante et de la taille de tabulation donnée.

```
str = 'xyz\t12345\tabc'

# no argument is passed
# default tabsize is 8
result = str.expandtabs()

print(result)
xyz    12345   abc
```

→ str.find(sub[, start[, end]]) :

Donne la première la position dans la chaîne où *sub* est trouvé dans le *slice* s[start:end]. Les arguments facultatifs *start* et *end* sont interprétés comme dans la notation des *slice*. Donne -1 si *sub* n'est pas trouvé.

```
quote = 'Let it be, let it be, let it be'

# first occurance of 'let it'(case sensitive)
result = quote.find('let it')
print("Substring 'let it':", result)
Substring 'let it': 11
# find returns -1 if substring not found
result = quote.find('small')
print("Substring 'small ':", result)
Substring 'small ': -1
# How to use find()
if (quote.find('be,') != -1):
        print("Contains substring 'be,'")
else:
        print("Doesn't contain substring")
Contains substring 'be,
```

→ str.format(*args, **kwargs) :

La méthode string format () formate la chaîne donnée en une sortie plus agréable en Python.

```
# default arguments

print("Hello {}, your balance is {}.".format("Adam", 230.2346))
Bonjour Adam, votre solde est de 230,2346.
# positional arguments
print("Hello {0}, your balance is {1}.".format("Adam", 230.2346))
Bonjour Adam, votre solde est de 230,2346.
# keyword arguments
print("Hello {name}, your balance is {blc}.".format(name="Adam", blc=230.2346))
Bonjour Adam, votre solde est de 230,2346.
# mixed arguments
print("Hello {0}, your balance is {blc}.".format("Adam", blc=230.2346))
Bonjour Adam, votre solde est de 230,2346.
```

→ str.format_map(mapping) :

La méthode format_map () est similaire à str.format (** mapping) sauf que str.format (** mapping) crée un nouveau dictionnaire contrairement à str.format_map (mapping).

```
point = {'x':4,'y':-5}

print('{x} {y}'.format_map(point))
4  -5
point = {'x':4,'y':-5, 'z': 0}
print('{x} {y} {z}'.format_map(point))
4 -5 0
```

→ str.index(sub[, start[, end]]) :

La méthode index () renvoie l'index d'une sous-chaîne à l'intérieur de la chaîne (si elle est trouvée). Si la sous-chaîne n'est pas trouvée, cela déclenche une exception.

```
sentence = 'Python programming is fun.'

result = sentence.index('is fun')
print("Substring 'is fun':", result)
Substring 'is fun': 19
```

```
result = sentence.index('Java')
print("Substring 'Java':", result)
Traceback (most recent call last):
  File "<string>", line 6, in
    result = sentence.index('Java')
ValueError: substring not found
```

→ str.isalnum() :

Donne `True` si tous les caractères de la chaîne sont alphanumériques et qu'il y a au moins un caractère, sinon `False`.

```
name = "M234onica"

print(name.isalnum())
True

name = "M3onica Gell22er "
print(name.isalnum())
False

name = "Mo3nicaGell22er"
print(name.isalnum())
True

name = "133"
print(name.isalnum())
True
```

→ str.isalpha() :

Donne `True` si tous les caractères de la chaîne sont alphabétiques et qu'elle contient au moins un caractère, sinon `False`.

```
name = "Monica"

print(name.isalpha())
True
# contains whitespace
name = "Monica Geller"
print(name.isalpha())
False
```

```
# contains number
name = "Mo3nicaGell22er"
print(name.isalpha())
False
```

→ str.isdecimal() :

Renvoie true si tous les caractères de la chaîne sont des caractères décimaux et qu'il y a au moins un caractère, false dans le cas contraire.

```
s = "28212"

print(s.isdecimal())
True

# contains alphabets
s = "32ladk3"
print(s.isdecimal())
False
# contains alphabets and spaces
s = "Mo3 nicaG el l22er"
print(s.isdecimal())
False
```

→ str.isdigit() :

Renvoie true si tous les caractères de la chaîne sont des chiffres et qu'il y a au moins un caractère, false dans le cas contraire.

```
s = "28212"

print(s.isdigit())
True
# contains alphabets and spaces
s = "Mo3 nicaG el l22er"
print(s.isdigit())
False
```

→ str.isidentifier() :

Donne `true`si la chaîne est un identifiant valide selon la définition du langage, sinon il renvoie `False .`

```
str = 'Python'

print(str.isidentifier())
True

str = 'Py thon'
print(str.isidentifier())
False

str = '22Python'
print(str.isidentifier())
False

str = ''
print(str.isidentifier())
False
```

→ str.islower() :

Donne `True` si tous les caractères capitalisables [de la chaîne sont en minuscules et qu'elle contient au moins un caractère capitalisable. Donne `False` dans le cas contraire.

```
s = 'this is good'

print(s.islower())
True
s = 'th!s is a1so g00d'
print(s.islower())
True

s = 'this is Not good'
print(s.islower())
False
```

→ str.isnumeric() :

Donne `True` si tous les caractères de la chaîne sont des caractères numériques, et qu'elle contient au moins un caractère, sinon `False`.

```
s = '1242323'

print(s.isnumeric())
True

#s = '²3455'
```

```
s = '\u00B23455'
print(s.isnumeric())
True

# s = '½'
s = '\u00BD'
print(s.isnumeric())
True

s = '1242323'
s='python12'
print(s.isnumeric())
False
```

→ str.isprintable() :

Donne `True` si tous les caractères de la chaîne sont affichables ou qu'elle est vide sinon, `False`.

```
s = 'Space is a printable'

print(s)
Space is a printable
print(s.isprintable())
True

s = '\nNew Line is printable'
print(s)
New Line is printable
print(s.isprintable())
False

s = ''
print('\nEmpty string printable?', s.isprintable())
Empty string printable? True
```

→ str.isspace() :

Donne `True` s'il n'y a que des caractères blancs dans la chaîne et il y a au moins un caractère, sinon `False`.

```
s = '   \t'

print(s.isspace())
```

```
True
s = ' a '
print(s.isspace())
False
s = ''
print(s.isspace())
False
```

→ str.istitle() :

Le istitle () renvoie True si la chaîne est une chaîne titrée. Sinon, il renvoie False.

```
s = 'Python Is Good.'

print(s.istitle())
True

s = 'Python is good'
print(s.istitle())
False

s = 'This Is @ Symbol.'
print(s.istitle())
True

s = '99 Is A Number'
print(s.istitle())
True

s = 'PYTHON'
print(s.istitle())
False
```

→ str.isupper() :

Donne `True` si tous les caractères différentiables sur la casse de la chaîne sont en majuscules et il y a au moins un caractère différentiable sur la casse, sinon `False`.

```
# example string

string = "THIS IS GOOD!"
print(string.isupper());
True
```

```
# numbers in place of alphabets
string = "THIS IS ALSO G00D!"
print(string.isupper());
True

# lowercase string
string = "THIS IS not GOOD!"
print(string.isupper());
False
```

→ str.join(itérable) :

La méthode de chaîne join () renvoie une chaîne en joignant tous les éléments d'un itérable, séparés par un séparateur de chaîne.

```
numList = ['1', '2', '3', '4']
separator = ', '
print(separator.join(numList))
1, 2, 3, 4
# .join() with tuples
numTuple = ('1', '2', '3', '4')
print(separator.join(numTuple))
1, 2, 3, 4
s1 = 'abc'
s2 = '123'

# each element of s2 is separated by s1
# '1'+ 'abc'+ '2'+ 'abc'+ '3'
print('s1.join(s2):', s1.join(s2))
s1.join(s2): 1abc2abc3

# each element of s1 is separated by s2
# 'a'+ '123'+ 'b'+ '123'+ 'b'
print('s2.join(s1):', s2.join(s1))
s2.join(s1): a123b123c
```

→ str.ljust(largeur [, fillchar]) :

La méthode string ljust () renvoie une chaîne justifiée à gauche d'une largeur minimale donnée.

```
string = 'cat'
width = 5
```

```
# print left justified string
print(string.ljust(width))
cat
```

→ str.rjust(largeur [, fillchar]) :

La méthode string rjust () renvoie une chaîne justifiée à droite d'une largeur minimale donnée.

```
string = 'cat'
width = 5
fillchar = '*'

# print right justified string
print(string.rjust(width, fillchar))
**cat
```

→ `str.lower()` :

La méthode string rjust () renvoie une chaîne justifiée à droite d'une largeur minimale donnée.

```
# example string

string = "THIS SHOULD BE LOWERCASE!"
print(string.lower())
this should be lowercase!

# string with numbers
# all alphabets whould be lowercase
string = "Th!s Sh0uLd B3 L0w3rCas3!"
print(string.lower())
th!s sh0uld b3 l0w3rcas3!
```

→ lstrip([chars]) :

Renvoie une copie de la chaîne des caractères supprimés au début.

```
string = "this should be uppercase!"
print(string.upper())
THIS SHOULD BE UPPERCASE!

# string with numbers
# all alphabets whould be lowercase
```

```
string = "Th!s Sh0uLd B3 uPp3rCas3!"
print(string.upper())
TH!S SH0ULD B3 UPP3RCAS3!
```

→ Str.swapcase() :

La méthode string swapcase () convertit tous les caractères majuscules en minuscules et tous les caractères minuscules en caractères majuscules de la chaîne donnée et la renvoie.

```
string = "THIS SHOULD ALL BE LOWERCASE."
print(string.swapcase())
this should all be lowercase.
string = "this should all be uppercase."
print(string.swapcase())
THIS SHOULD ALL BE UPPERCASE.
string = "ThIs ShOuLd Be MiXeD cAsEd."
print(string.swapcase())
tHiS sHoUlD bE mIxEd CaSeD.
```

String lstrip() :

La méthode lstrip () renvoie une copie de la chaîne avec les premiers caractères supprimés (en fonction de l'argument chaîne passé).

```
random_string = '   this is good '

# Leading whitepsace are removed
print(random_string.lstrip())
this is good
# Argument doesn't contain space
# No characters are removed.
print(random_string.lstrip('sti'))
this is good
print(random_string.lstrip('s ti'))
his is good
website = 'https://www.ubuntu.com/'
print(website.lstrip('htps:/.'))
www.ubuntu.com/
```

String rstrip() :

La méthode rstrip () renvoie une copie de la chaîne avec les caractères de fin supprimés (en fonction de l'argument de chaîne passé).

```
random_string = 'this is good    '

# Trailing whitepsace are removed
print(random_string.rstrip())
c'est bon

# Argument doesn't contain 'd'
# No characters are removed.
print(random_string.rstrip('si oo'))
c'est bon
print(random_string.rstrip('sid oo'))
c'est g
website = 'www.facebook.com/'
print(website.rstrip('m/.'))
www.facebook.co
```

String strip() :

La méthode strip () renvoie une copie de la chaîne en supprimant les caractères de début et de fin (en fonction de l'argument de chaîne passé).

```
string = '  xoxo love xoxo   '

# Leading and trailing whitespaces are removed
print(string.strip())
xoxo amour xoxo

# All <whitespace>,x,o,e characters in the left
# and right of string are removed
print(string.strip(' xoe'))
aimer
# Argument doesn't contain space
# No characters are removed.
print(string.strip('stx'))
   xoxo amour xoxo
string = 'android is awesome'
print(string.strip('an'))
```

```
droïde est génial
```

String partition()

La méthode partition () divise la chaîne à la première occurrence de la chaîne d'argument et renvoie un tuple contenant la partie avant le séparateur, la chaîne d'argument et la partie après le séparateur.

```
string = "Python is fun"

# 'is' separator is found
print(string.partition('is '))
('Python ', 'is ', 'fun')
# 'not' separator is not found
print(string.partition('not '))
('Python is fun', '', '')

string = "Python is fun, isn't it"
# splits at first occurence of 'is'
print(string.partition('is'))
('Python ', 'is', " fun, isn't it")
```

String maketrans() :

La méthode chaîne maketrans () renvoie une table de mappage pour la traduction utilisable pour la méthode translate ().

```
dict = {"a": "123", "b": "456", "c": "789"}
string = "abc"
print(string.maketrans(dict))
{97: '123', 98: '456', 99: '789'}

# example dictionary
dict = {97: "123", 98: "456", 99: "789"}
string = "abc"
print(string.maketrans(dict))
{97: '123', 98: '456', 99: '789'}
```

String rpartition() :

La rpartition () divise la chaîne à la dernière occurrence de la chaîne d'argument et renvoie un tuple contenant la partie avant le séparateur, la chaîne d'argument et la partie après le séparateur.

```
string = "Python is fun"

# 'is' separator is found
print(string.rpartition('is '))
('Python ', 'is ', 'fun')

# 'not' separator is not found
print(string.rpartition('not '))
('', '', 'Python is fun')
string = "Python is fun, isn't it"

# splits at last occurence of 'is'
print(string.rpartition('is'))
('Python is fun, ', 'is', "n't it")
```

String translate ()

La méthode string translate () renvoie une chaîne dans laquelle chaque caractère est mappé à son caractère correspondant dans la table de traduction.

```
firstString = "abc"
secondString = "ghi"
thirdString = "ab"

string = "abcdef"
print("Original string:", string)
Chaîne d'origine: abcdef
translation = string.maketrans(firstString, secondString, thirdString)

# translate string
print("Translated string:", string.translate(translation))
Chaîne traduite: idef
```

String replace()

La méthode replace () retourne une copie de la chaîne où toutes les occurrences d'une sous-chaîne sont remplacées par une autre sous-chaîne.

```
song = 'cold, cold heart'

# replacing 'cold' with 'hurt'
print(song.replace('cold', 'hurt'))
hurt, hurt heart

song = 'Let it be, let it be, let it be, let it be'


# replacing only two occurences of 'let'
print(song.replace('let', "don't let", 2))
Let it be, don't let it be, don't let it be, let it be
```

Python String rfind ()

La méthode rfind () renvoie l'index le plus élevé de la sous-chaîne (si trouvé). S'il n'est pas trouvé, il renvoie -1.

```
quote = 'Let it be, let it be, let it be'

result = quote.rfind('let it')
print("Substring 'let it':", result)
Sous-chaîne 'laissez-le': 22

result = quote.rfind('small')
print("Substring 'small ':", result)
Sous-chaîne 'petite': -1

result = quote.rfind('be,')
if  (result != -1):
   print("Highest index where 'be,' occurs:", result)
else:
   print("Doesn't contain substring")
Contient la sous-chaîne 'be,'
```

String rindex ()

La méthode rindex () retourne l'index le plus élevé de la sous-chaîne à l'intérieur de la chaîne (si trouvé). Si la sous-chaîne n'est pas trouvée, cela déclenche une exception.

```
quote = 'Let it be, let it be, let it be'

result = quote.rindex('let it')
print("Substring 'let it':", result)
Sous-chaîne 'laissez-le': 22

result = quote.rindex('small')
print("Substring 'small ':", result)
Traceback (dernier appel le plus récent):
   Fichier "...", ligne 6, dans <module>
      result = quote.rindex ('petit')
ValueError: sous-chaîne introuvable
```

Python String split ()

La méthode split () rompt une chaîne au séparateur spécifié et renvoie une liste de chaînes.

```
text= 'Love thy neighbor'
# splits at space
print(text.split())
['Amour', 'ton', 'voisin']

grocery = 'Milk, Chicken, Bread'

# splits at ','
print(grocery.split(', '))
["Lait", "Poulet", "Pain"]

# Splitting at ':'
print(grocery.split(':'))
["Lait, Poulet, Pain"]
```

Python String rsplit()

La méthode rsplit () sépare la chaîne de la droite au séparateur spécifié et renvoie une liste de chaînes.

```
text= 'Love thy neighbor'

# splits at space
print(text.rsplit())
```

```
['Love', 'thy', 'neighbor']
grocery = 'Milk, Chicken, Bread'

# splits at ','
print(grocery.rsplit(', '))
['Milk', 'Chicken', 'Bread']

# Splitting at ':'
print(grocery.rsplit(':'))
['Milk, Chicken, Bread']
```

Python String splitlines()

La méthode splitlines () divise la chaîne aux sauts de ligne et renvoie une liste de lignes dans la chaîne.

```
grocery = 'Milk\nChicken\r\nBread\rButter'
print(grocery.splitlines())
['Milk', 'Chicken', 'Bread', 'Butter']

print(grocery.splitlines(True))
['Milk\n', 'Chicken\r\n', 'Bread\r', 'Butter']

grocery = 'Milk Chicken Bread Butter'
print(grocery.splitlines())
['Milk Chicken Bread Butter']
```

String startswith()

La méthode startswith () renvoie True si une chaîne commence par le préfixe spécifié (chaîne). Sinon, il renvoie False.

```
text = "Python is easy to learn."
result = text.startswith('is easy')
```

```
# returns False
print(result)
False

result = text.startswith('Python is ')
# returns True
print(result)
True

result = text.startswith('Python is easy to learn.')
# returns True
print(result)
True
```

→ String title()

La méthode title () renvoie une chaîne avec la première lettre de chaque mot en majuscule; une chaîne de titre en casse.

```
text = 'My favorite number is 25.'
print(text.title())
My Favorite Number Is 25.

text = '234 k3l2 *43 fun'
print(text.title())
234 K3L2 *43 Fun
```

String zfill()

La méthode zfill () renvoie une copie de la chaîne avec les caractères '0' remplis vers la gauche.

```
text = "program is fun"

print(text.zfill(15))
0program is fun

print(text.zfill(20))
```

```
000000program is fun

print(text.zfill(10))
program is fun
```

- **Les listes :**

  Exemple sur les listes :

```
>>> squares = [1, 4, 9, 16, 25]   # est une liste des entiers

>>> squares       #on saisi le nom de la liste sur la console

[1, 4, 9, 16, 25]   # il m'affiche la liste.

>>> squares[0]  #si on tape nom_de_liste[indice] il va m'afficher
                #l'element qui correspond l'indice

1

>>> squares[-1]

25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

→ on peut concaténer 2 listes en ajoutant le (+) entre les deux elles.

```
>>> squares + [36, 49, 64, 81, 100]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

→ on peut ajouter un element à une liste avec (append())

```
>>> Nums = [1, 8, 27, 65, 125]
>>> Nums.append(216)  # add the cube of 6
```

Il est possible d'imbriquer des listes (de créer des listes contenant d'autres listes), par exemple :

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

→ On peut calculer la longueur de la liste avec len().

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

.

.

.

.