IIT
INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine Privée

FASCICULE DE COURS

PROGRAMMATION PYTHON

1er Génie Informatique

Semestre 1

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

2021-2022

Introduire les concepts de base du langage Python_version **3**

Cours ±**15h** ☑

TP ±**20h** ☑

30% DS

70% Examen

Contexte : Langages de programmation

Polarité des langages de programmation

Language Ranking: IEEE Spectrum

Rank	Language	Score
1	Python	100.0
2	Java	95.4
3	C	94.7
4	C++	92.4
5	JavaScript	88.1
6	C#	82.4
7	R	81.7
8	Go	77.7
9	HTML	75.4
10	Swift	70.4

Python conforte sa place de leader

Classement des langages de programmation selon IEEE, 2021

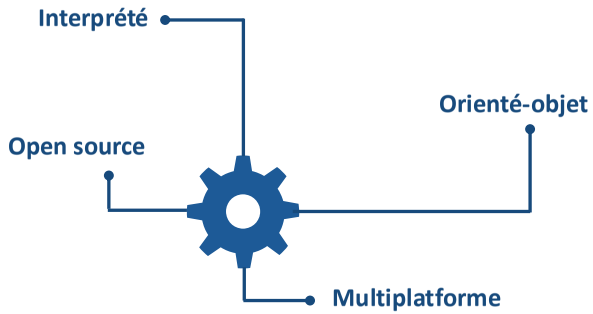
PLAN

DU COURS

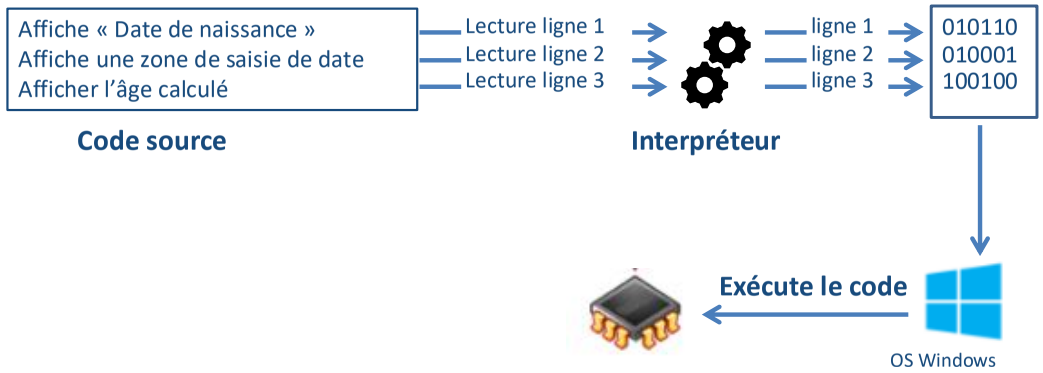
Plan du cours: PYTHON BASICS

C'est quoi Python?

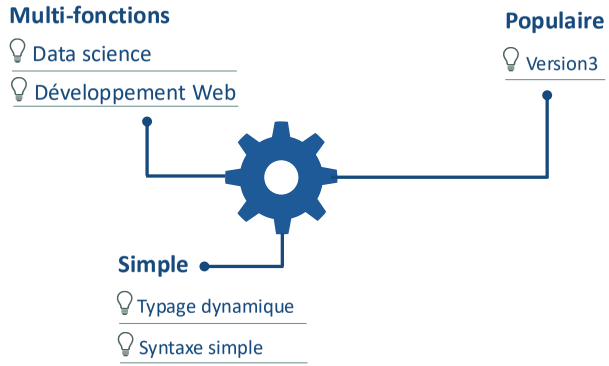
- Langage de programmation de haut niveau inventé en 1991 par Guido van Rossum
- Utilisé plus généralement en analyse de données et en calcul scientifique



Python : langage interprété



Pourquoi apprendre Python?



Environnement de développement (IDE)

IDLE

PyCharm

Spyder

Atom

PyDev

Jupyter Notebook

Jupyter Notebook

- **Jupyter Notebook** ne fonctionne pas uniquement comme un IDE, mais aussi comme un outil de présentation
  - ❑ permettant le travail en mode interactif ou fichier

> "Jupyter Notebook should be an integral part of any Python data scientist's toolbox. It's great for prototyping and sharing notebooks with visualizations"

Anaconda?

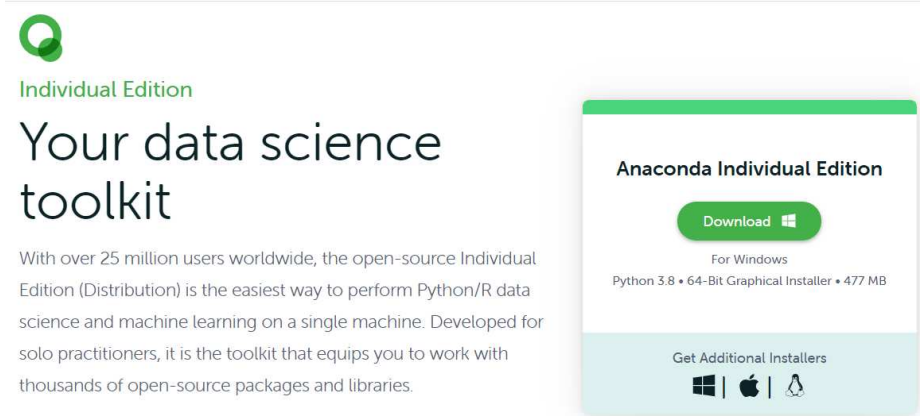
- **Anaconda** est une **distribution complète** de Python
  - Éviter les problèmes d'incompatibilités entre les différents packages
  - Proposer un outil de gestion de packages appelé **Conda** utile pour mettre à jour et installer facilement les librairies destinées au **calcul numérique** (numpy, scipy, panda, mathplotlib, etc.)

Anaconda = Python + librairies + **Jupyter notebook**

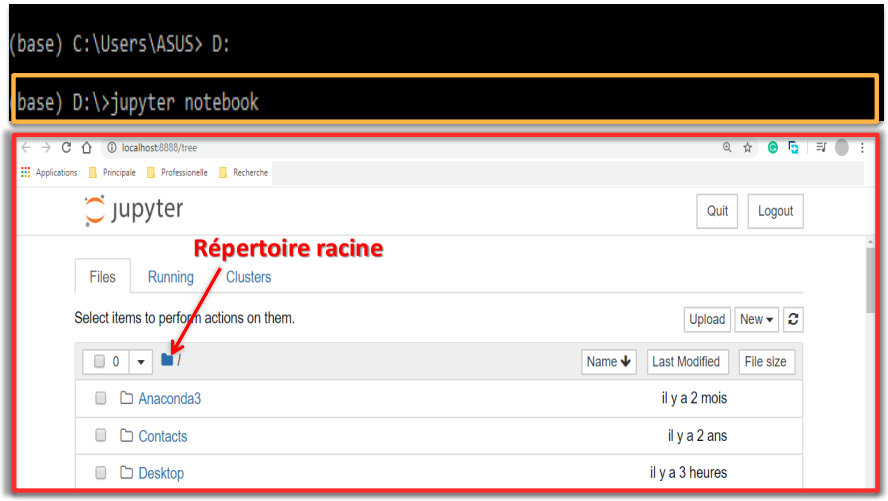
ANACONDA.

Installation d'Anaconda

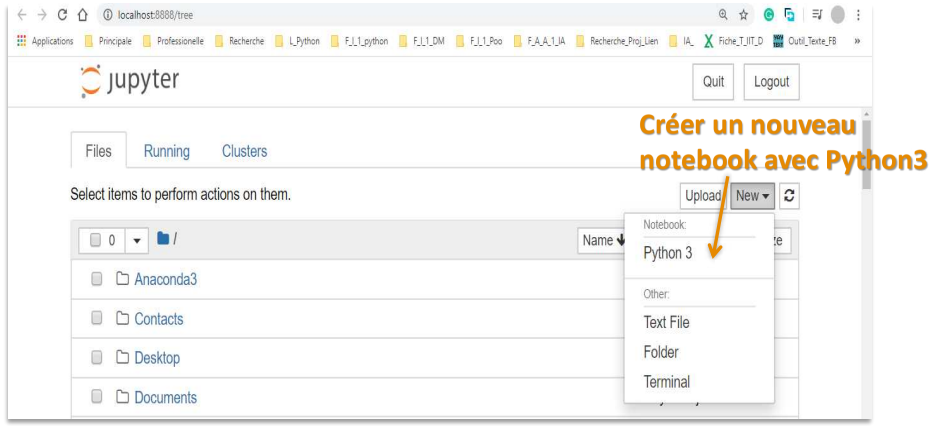
- Lien de téléchargement : **https://www.anaconda.com/distribution/**



Anaconda Prompt : Ouvrir Jupyter Notebook



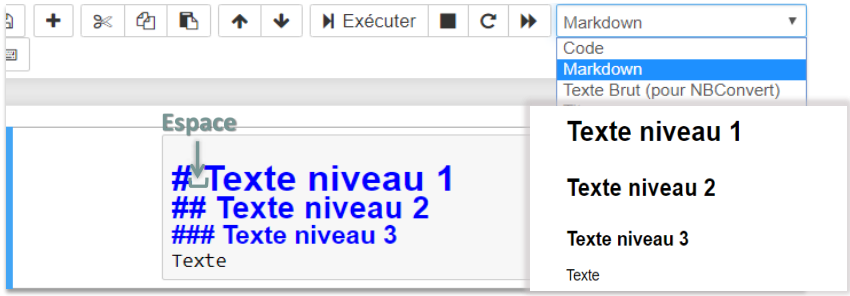
Création d'un nouveau Notebook



Le fichier résultat de notebook est d'extension **.ipynb**

Format Markdown

- Markdown : langage de balisage léger
- Rédiger du **texte formaté** (gras, italique, liens, titres, images, formules mathématiques, etc.) avec quelques balises très simples



NB. Un document Markdown peut être lu sans donner l'impression d'avoir été balisé ou formaté par des instructions particulières

Commentaire

- Un commentaire serve à :
  - ❑ Expliquer ce qui se passe dans une portion de code
  - ❑ Documenter qui a écrit le code ou d'autres informations
  - ❑ Désactiver une ligne de code

```
# commentaire sur une seule ligne
"""
commentaire sur plusieurs lignes
"""
```

Commentaire sur un seul ligne #

Commentaire sur plusieurs lignes """ """

```
'\ncommentaire sur plusieurs lignes\n'
```

?

Variables

- Une variable est caractérisée par un **identificateur** (**nom**)
- Et un **type** automatiquement attribué

- Le nom des variables peut être constitué de lettres minuscules (**a à z**), lettres majuscule (**A à Z**), nombres (**0 à 9**) et/ou caractère souligné (_)

NB.

Python est sensible à la casse, ce qui signifie que les variables « test », « Test » ou « TEST » sont différentes

→ N'utilise pas d'espace dans un nom de variable & des mots réservés

Mots réservés

```
import keyword
for e in (keyword.kwlist):
    print (e, end="\t")

print ("")
print ("Le nombre de mots réservés est: ", len(keyword.kwlist))
```

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

35 mots réservés

Type des variables

type(var) renvoie le type de la variable en argument var

id(var) renvoie un entier, représentant l'identifiant interne de la variable en argument var quel que soit son type

```
x=False
type(x)# bool
```

```
id(x)# 504140000
```

```
x=1
type(x)# int
```

```
id(x)# 504506560
```


Opérateurs

Les opérateurs arithmétiques

x+y	Addition
x-y	soustraction
x*y	Multiplication
x/y	Division réelle
x**n	Élévation à la puissance
x//y	Division entière
x%y	Reste de division entière
-x	Opposé - opérateur unaire

Les opérateurs de comparaison

x<y	Strictement inférieur
x<=y	Inférieur ou égal
x>y	Strictement supérieur
x>=y	Supérieur ou égal
x==y	Égal
x!=y	Différent
x is y	x et y représentent le même objet
x is not y	x et y ne représentent pas le même objet

Les opérateurs logiques

x or y	Ou logique
x and y	Et logique
not x	Non logique
x ^ y	Ou Exclusif logique

Les opérateurs bit à bit

x | y	Ou bit à bit
x & y	Et bit à bit
x ^ y	Ou Exclusif bit à bit
x << y	Décalage de y bits vers la gauche sur x
x >> y	Décalage de y bits vers la droite sur x

Opérateurs

Priorité des opérateurs

**	puissance
*, /, //, %	multiplication, division, division entière, reste
+, -	addition et soustraction
<<, >>	décalages bit à bit
&, ^, |	Et bit à bit, Ou Exclusif bit à bit , Ou bit à bit
is, is not	Tests d'appartenance
<, <=, >, >=, !=, ==	Comparaisons
or , and, not x, ^ y	Opérateurs logiques

Les priorités des opérateurs de la plus hausse à la plus basse

Affectation parallèle & multiple

```
x=8; y=8; z=12
```

```
x, y, z=8, 8, 12
print(x, y, z)
```

Affectation parallèle

```
x=y=8 ; z=12
print(x, y, z)
```

Affectation multiple

NB. L'ordre d'affectation est important

Activité : Étant donné x= 11 et y= 22, écrire dans une seule ligne l'instruction qui permet de permuter la valeur de x par y. Afficher les nouvelles valeurs de x et y

```
x,y=11,22
x,y=y,x
print(x,y)
```

Instruction de sortie print()

- Affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne
- Ajoute un retour à la ligne par défaut

```
print("Bonjour") # Bonjour
print("Taoufik")  # Taoufik
```

```
print("Bonjour", end="")
print("Taoufik") # BonjourTaoufik
```

Argument qui supprime le retour à la ligne et le remplace par la chaine en entrée

```
print("Bonjour", "Taoufik", sep="-")
print("Taoufik")  # Bonjour-Taoufik
```

Argument qui met un séparateur entre les valeur des autres arguments donnés en entrée

Instruction d'entrée input()

- Provoque une interruption dans le programme courant où l'utilisateur est invité à entrer des données au clavier

```
Entrée [*]: chaine = input()
age = input("Age : ")

print("La valeur saisie de chaine : ", chaine)
print("La valeur saisie d'Age : ", age)
```


```
Taoufik
Age :
```

...

```
type(age)  # str
```


Conversion de types

- Chaine de caractères (str)→ entier (int) : int("chaine")

```
annN = input("Année de naissance :")
age =  2021-int(annN) # 1991
print("Age : ", age)
type(age) # int
```

- Chaine de caractères (str) → type adéquat : eval("chaine")

```
age = eval(input("Age : ")) # 30
print("La valeur saisie d'Age : ", age)
type(age) # int
```

```
a=eval("12.5")
type(a) # float
```

a=int("12.5")?

Conversion de types

- Entier (int)→ réel (float) : float()
- Entier (int), réel (float) → chaine de caractères : str()

```
a=33 # <class 'int'>
b="a35" # <class 'str'>
c=111.94 # <class 'float'>
d=112 # <class 'int'>

a=float(a) # ?
b=int(b) # ?
b=int(c) # ?
d=str(d) # ?
```

Ecriture formatée Ancienne méthode

```
x = 29
nom = "Taoufik"
print("%s a %d ans" % (nom, x))# Taoufik a 29 ans
```

int

str

```
y=12.658456
print("y=%f"%y)# y=12.658456
```

float

```
print("y=%.2f"%y)  # y=12.66
```

float (avec deux chiffres après la virgule)

Conseil : Eviter cette écriture. Utiliser plutôt la fonction ch.format()

Ecriture formatée ch.format()

- Permet une meilleure organisation de l'affichage des variables
- Agit sur la chaîne de caractères ch à laquelle elle est attachée par le point

NB. Les accolades { } précisent l'endroit où le contenu de la variable doit être inséré

```
x=28; y=35.5; nom="x"
print("La valeur de {} est {}".format(nom,x))   #sans indiçage          # La valeur de x est 28
print("{1} est la valeur de {0}".format(nom,x)) #Avec indiçage          # 28 est la valeur de x
print("{} est la valeur de {nomX}".format(x,nomX=nom))                  # 28 est la valeur de x
print("La valeur de y est {:.2f}".format(y)) #Avec spécification de type # La valeur de y est 35.00
print("{:e}".format(12)) #Notation exponentielle                        # 1.200000e+01
```


Quiz

1. Python est un langage (une seule réponse)
   - ☑ interprété
   - ☐ machine
   - ☐ compilé
   - ☐ binaire
2. Python est sensible à la casse (une seule réponse)
   - ☑ Vrai
   - ☐ Faux
3. Soit **n**=12.0/4; **x**=12.0//5; **m**="13.0" (Choix multiple)
   - ☑ type(**n**) retourne *float*
   - ☑ type(**x**) retourne *float*
   - ☐ eval(**m**) convertie m en *int*
   - ☑ eval(**m**) convertie m en *float*

CONTRÔLE DE FLUX

Exécution conditionnelle

Test

if <expression> : ← **N'oubliez pas le double point**
tabulation ⟷ Traitement **l'indentation (tabulation) !**

Test à plusieurs cas

```
if <expression> :
⟷ Traitement
else :
⟷ Traitement
```

```
if <expression> :
⟷ Traitement
elif <expression> :
⟷ Traitement
else :
⟷ Traitement
```

NB. Une expression peut représenter des conditions assemblées avec les opérateurs logiques

Instructions conditionnelles imbriquées

Il est possible d'imbriquer des instructions conditionnelles les unes dans les autres, de manière a réaliser des structures de décision complexes

Exemple

```
x=eval(input())
y,z,a="KK","AA13", 10
if x==12:
    if y=="KK":
        if z=="AA13":
            print("T1")
        else:
            print("T2")
    else:
        print("T3")
else:
    print("T4")
```


L'instruction while

```
while <expression> :
⟷ Instructions à répéter
```

Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté

Si la condition est vraie, alors le corps de la boucle est répété *indéfiniment*

NB.
La boucle doit contenir au moins une instruction
Cette instruction doit changer la valeur d'une variable intervenant dans la condition, de manière à ce que cette condition puisse devenir fausse

```
n=2
while n>1:
    print("Bonjour!")
```

← **Boucle infinie**

L'instruction for

```
for <element> in <sequence> :
⟷ Instructions à répéter
```

Elle répète une séquence d'instructions autant de fois que le nombre d'éléments dans une **séquence** (**Conteneur** ou **Intervalle**)

Conteneur : chaine de caractère, liste, tuple, ensemble, ou dictionnaire

Intervalle : **range**
range(val) : Énumère les entiers de **0→val-1**
range(val1,val2) : Énumère les entiers de **val1→val2-1**
range(val1,val2,val3) : Énumère les entiers de **val1→val2-1** par **pas** de **val3**

L'instruction for

```
for x in range(3):
    print(x, end="/") # 0/1/2/

for x in range(2,6):
    print(x, end="/") # 2/3/4/5/

for x in range(2,6,2):
    print(x, end="/") # 2/4/

for x in range(6,2,-1):   # v1 v2 Pas=-1 ; v1>v2-pas
    print(x, end="/") # 6/5/4/3/

for x in range(4):
    print(x, end="/")# 0/1/2/3/
    x +=2
```

x prend les différentes valeurs imposées par range, même si elle est modifiée dans le corps de la boucle for

break & continue

break : sort violemment de la boucle et passe à la suite

```
for x in range(4):
    if x==2:
        break
    print(x, end="/")
print("YY")
# 0/1/YY
```

continue : saute directement à l'itération suivante sans exécuter la suite du bloc d'instructions de la boucle

```
for x in range(4):
    if x==2:
        continue
    print(x, end="/")
print("YY")
# 0/1/3/YY
```


Bloc d'instructions en fin de boucle

```
while <expression> :
    Instructions à répéter
else :
    Si sortie propre
```

```
for <element> in < sequence > :
    Instructions à répéter
else :
    Si sortie propre
```

NB.

On peut ajouter un bloc d'instructions en fin de boucle, qui sera réalisé si la sortie de la boucle s'est faite proprement (**sans break**), avec l'instruction **else**

Quiz

1. Quelle est la sortie du code suivant ? (une seule réponse)

```
x=True; y=z=False
if not x or y:
    print (x)
elif not x or not y and z:
    print (not x)
else: print ("None")
```

- ☐ True
- ☐ False
- ☑ None

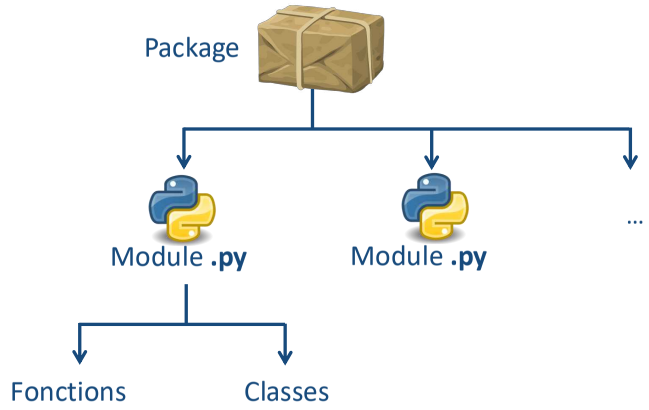
2. Par quoi remplacer le vide pour que le code affiche **13AA** ? (une seule réponse)

```
for i in range(1,4):
    if i==5//2:
        ____________________________
    print(i, end="")
else: print("AA")
```

- ☐ break
- ☑ continue
- ☐ print(i)

MODULES

Modules



Un module python est un fichier **.py** rassemblant des fonctions et des classes

Importation d'un module

Pour accéder aux fonctions ou classes d'un module existant, il faut **charger le module** avec **import**

```
from nom_module import *

from nom_module import fct1, classe1, ...

import nom_module1 , nom_module2

import nom_module as nomLocal

from nom_package.nom_module import ...

import nom_package.nom_module
```


Quelques modules standards

- **math** fournit des fonctions pour réaliser des calculs mathématiques complexes
- **datetime** fournit des fonctions pour manipuler de façon simple ou plus complexe des dates et des heures ;
- **json** permet l'encodage et le décodage de données au format JSON
- **os** fournit des fonctions pour utiliser les fonctionnalités dépendantes du système d'exploitation
- **re** fournit des opérations de *matching* des expressions régulières sur les chaines de caractères
- **random** fournit des fonctions pour simuler le hasard
- **csv** implémente des classes pour lire et écrire des données tabulaires au format CSV
- **sys** fournit un accès à certaines variables système utilisées et maintenues par l'interpréteur

⋮

Module math

Les fonctions mathématiques sont définies dans le module **math**

Pour importer **math** : **from math import ***

ou bien **import math**

factorial(x) : Retourne la factorielle de x (erreur si x n'est pas entier ou s'il est négatif)

exp(x) : Retourne e^x

log2(x) : Retourne le logarithme en base 2 de x

sqrt(x) : Retourne la racine carrée de x

ceil(x) : Retourne le plus petit entier, plus grand ou égal à x

⋮

https://docs.python.org/fr/3.5/library/math.html

Quiz

1. Pour charger le module math, il suffit de créer **from math import *** (une seule réponse)
   - ☑ Vrai
   - ☐ Faux

2. Quel module en Python supporte les expressions régulières ? (une seule réponse)
   - ☑ re
   - ☐ regex
   - ☐ pyregex
   - ☐ Aucune de ces réponses n'est correcte

3. Quelle est la sortie du code suivant ? (une seule réponse)

```
import math
print(sqrt(10))
```

   - ☐ 3.16
   - ☐ 3
   - ☑ Erreur

Conteneurs standard en Python

Les séquences

- Conteneur ordonné d'éléments indicés par des entiers
- Chaines de caractères (str), Listes (list) & Tuples (tuple)

Les ensembles (set)

- Conteneur non ordonné d'éléments sans répétition
- Sans Indiçage par des entiers

Les dictionnaires (dict)

- Tableaux associatifs
- Stocker des couples « **clé-valeur** »

Mutable vs Immutable!

Tout variable en Python=un **objet**

Un **objet** →**Type** + **Id** + **value**

Tous les objets peuvent être soit **mutables**, soit **immutable**

- MUTABLE : Objets dont **la valeur peut être modifiée** sans changer d'Id
- IMMUTABLE : Objets dont **la valeur est inchangeable une fois qu'ils sont créés** sans changer d'Id

△**Mutable** ☒list ☒ dict

△**Immutable** ☒bool ☒ int ☒ float ☒ tuple ☒ str ☒ set

CHAINE DE CARACTÈRES

Chaines de caractères

- Représentent toutes les informations qui ne sont pas numériques
- Appariaient entre guillemets simples ' ', doubles " " et triples """ """

```
chaine1 = 'aaaa'
chaine2 = "aaaaaaaa"
chaine3 = """aaaaa bbbb cccccc
ddddd eeeee ppppppp"""

print (chaine1) ; print(chaine2) ; print(chaine3)
```

Textes sur plusieurs lignes

NB.

print ('Aujourd'hui') **SyntaxError: invalid syntax**

print ('Aujourd\'hui') ou bien print ("Aujourd'hui")

Insérer le caractère antislash « \ » v

Utiliser «"» au lieu de «'»

Accès & Slicing

```
0 1
ch="Programmation Python"
-2 -1
```

Indiçage positif

Indiçage négatif

print(**ch[1]**) **# r** print(**ch[3]**) **# g**

print(**ch[-1]**) **# n** print(**ch[-20]**) **# P**

Slicing ou découpage en tranche

[ind1:ind2] → ind1… ind2-1

print(**ch[0:3]**) **# Pro 0→2** print(**ch[2:5]**) **# ogr 2→4**

print(**ch[:3]**) équivalent à print(**ch[0:3]**) **# Pro 0→2**

print(**ch[2:]**) **# ogrammation Python 2→Fin**

print(**ch[2::4]**) **# omiPo 2→Fin (pas=4)**

print(**ch[2::-1]**) **# orP 2→Fin (pas=-1)**

Concaténation, répétition et modification

- Concaténation et Répétition : + *

```
a="AA"+"BB"
b="AB"*3
print (a) # AABB
print (b) # ABABAB
```

Les chaines de caractères sont immutable (non modifiable)

- Modification d'un ou plusieurs caractère(s)

```
mot1 = "Programmation"
mot2 = " Python"
mot1=mot1+mot2
print (mot1)
```

```
mot = "pytton"
print (mot[3])
mot[3]="h"
print (mot[3])
```

NB.
Une fois une chaîne de caractères définie, vous ne pouvez plus modifier un de ses éléments, mais on peut définir une nouvelle chaine avec le même identificateur

Fonctions élémentaires

- len(ch) renvoie la longueur de ch
- ch.lower() convertit une chaine ch en minuscule
- ch.upper() convertit une chaine ch en majuscule
- ch.title() convertit en majuscule l'initiale de chaque mot dans ch
- ch.capitalize() convertit en majuscule seulement la première lettre de la chaine ch
- ch.swapcase() convertit toutes les majuscules dans ch en minuscules, et vice-versa

```
ch="Python Pour programmer"
print("Longueur de ch=",len(ch))
print(ch.lower())
print(ch.upper())
print(ch.capitalize())
print(ch.title())
print(ch.swapcase())
```

```
Longueur de ch= 22
python pour programmer
PYTHON POUR PROGRAMMER
Python pour programmer
Python Pour Programmer
pYTHON pOUR PROGRAMMER
```


Autres fonctions sur les chaines de caractères

- ch.count(sch) compte le nombre d'une sous-chaine sch dans ch
- sch in/ not in ch vérifie si une sous-chaîne sch existe dans ch
- ch.startswith(scr) vérifie si une chaîne de caractères commence par une sous-chaine sch
- ch.find(sch) cherche la position d'une sous chaine sch dans ch
- ch.index(shr) retrouve l'indice de la première occurrence de la chaine shr dans ch

```
mot = "Hi python"
print(mot.count("on"))# 1
print(mot.count("n")) # 1
```

```
print ("HI" in mot)              # False
print ("HI" not in mot)          # True
print (mot.startswith("Hi"))     # True
print (mot.startswith("Hi  "))   # True
```

```
print(mot.find ("y"))            # 4
print(mot.find ("tho"))          # 5
print(mot.find ("a"))            # -1
print(mot.find ("hoi"))          # -1
```

```
print(mot.index("y"))# 4
print(mot.index("yl"))
```

```
"pypythonp".index("p",3) # 8
```


Autres fonctions sur les chaines de caractères

- ch.split() découpe ch en plusieurs éléments selon n'importe quelle combinaisons d'espaces blancs (' ', '\n', '\t')
- ch.split(sep) découpe ch en plusieurs éléments selon une chaine séparateur sep
- ch.replace(sch1, sch2) remplace toutes les occurrences de sch1 par sch2
- ch.strip() permet de nettoyer les bords de ch
- ch.isdigit() vérifie si ch se compose uniquement de chiffres

```
ch="AA dd; bb \n c; K"
print(ch.split()) #['AA', 'dd;', 'bb', 'c;', 'K']
print(ch.split(";")) #['AA dd', ' bb \n c', ' K']
print(ch.split(";", maxsplit=1))
        #['AA dd', ' bb \n c; K']
```

Le nombre de fois qu'on souhaite découper la chaine

```
mot = "pytton"
mot=mot.replace("t","h")
print (mot) # pyhhon
```

```
mot = "pytton"
mot=mot.replace("tt","th")
print (mot) # python
```

```
ch = "AABfs2234BBDDS"
print (ch.isdigit()) # False
```

Quiz

1. Une chaine de caractère est-elle mutable ? (une seule réponse)
   - ☐ True
   - ☑ False

2. Quelle est la sortie de l'instruction **'python'[-1:-4:-1][::-1]** ? (une seule réponse)
   - ☑ 'hon'
   - ☐ 'nh'
   - ☐ 'noh'
   - ☐ ' '

3. Quelle est la sortie du code suivant ? (une seule réponse)

```
ch="AA;BDE"
for n in ch.split(";"):
    print(n, end="-")
```

   - ☐ AA BDE
   - ☐ AA-BDE
   - ☑ AA-BDE-
   - ☐ BDE

| LISTES & TUPLES

Listes

Représentent une série de valeurs (objets) qui peuvent être de types différents

NB. Une liste est déclarée par une série de valeurs, séparée par des virgules, et le tout encadré par des crochets **[ ]**

```
l1=["ABC",20,34,"M"]
print(l)  # ['ABC', 20, 34, 'M']
l2=[] équivalent à l2=list()
print(l2)  # []
```

← **Une liste vide se définit par [] ou list()**

```
l3=[20,30,5]
for i in l3:
    print(i)  # ?
```

Ou bien

```
for i in _____?_____:
    print(l3[i])
```

← **Parcours d'une liste**

Accès et Slicing

```
  0  1   2  3  4   → Indiçage positif
l= [11, 'abc', 'ddd',4,12]
 -5  -4  -3  -2 -1  ← Indiçage négatif
```

print(**l[1]**) **# abc** print(**l[-1]**) **# 12**

print(**l[:]**) équivalent à print(**l**) **# [11, 'abc', 'ddd',4,12]**

print(**l[2:]**) **# ['ddd',4,12]** (à partir de l'élément d'indice 2 vers la fin)

print(**l[:3]**) **# ['ddd',4,12]** (à partir de la 1er élément à celui d'indice 3-1=2)

print(**l[-1:-3]**) **# []** (l[val1:val2]) val1<val2

print(**l:-3**]) **# [11, 'abc']**
0→-3-1=-4

print(**l[-3:-1]**) **# ['ddd',4]**
-3→-1-1=-2

print(**l[::2]**) **# [11, 'ddd',12]** (toutes les éléments de l par pas de 2)

print(**l[1:3:2]**) **# ?**

Modification des éléments d'une liste

Les listes sont **mutables** : modifiables par l'ajout, le remplacement ou la suppression d'un objet

```
l=[20,29,13,17]

l[1]=5   ; print(l) # [20,5,13,17]

l[1:3]=[4,8]   ; print(l) # [20, 4, 8, 17]

l[1:3]=[1,2,3] ; print(l) # [20, 1, 2, 3, 17]

l[-1]=4 ; print(l) # [20, 1, 2, 3, 4]

l[::2]=[0,-2,-4] ; print(l) # [0, 1, -2, 3, -4]

l[::2]=[0,-2,-4,-6] ; print(l) # ⊗
```

```
l1=[0,1,2]
l2=["AA",2,5]

print(id(l2))  # 1828080535560
l2=l1+l2  ⟵ Concaténation
print(l2)  # [0,1,2,"AA",2,5]
print(id(l2))  # 1828080535560

l2=l1*2  ⟵ Répétition
print(l2)  # [0,1,2,0,1,2]

l2=[l1,l2]  ⟵ Liste de listes
print(l2) # [[0,1,2],["AA",2,5]]
```


Opérations sur les listes

- len(lis) renvoie la longueur de la liste lis
- min(lis) renvoie l'élément minimum dans la liste lis
- max(lis) renvoie l'élément maximum dans la liste lis
- sum(lis) renvoie la somme des éléments de la liste lis
- lis.count(elem) compte le nombre de l'élément elem dans la liste lis
- list & range : génère une liste d'entiers

```
l1=["AB",20,6,1]
print(len(l1))  # 4
print(max(l1))  ⊗

l2=[11,46,1,1]
print(max(l2))  # 46
print(sum(l2))  # 59
print(l2.count(1))  # 2

l3=list(range(1,4))
print(l3)  # [1, 2, 3]
```


Opérations sur les listes

- el in lis, el not in lis : vérifie l'appartenance d'un élément el dans lis
- lis.append(elem) ajoute l'élément elem à la fin de la liste lis
- lis.insert(ind, elem) insère l'élément elem dans l'indice ind de lis
- lis.extend(seq) ajoute le contenu d'une sequence seq à lis
- lis.sort() trie la list lis
- sep.join(lis) convertie la liste lis en une chaine de caractères en ajoutant un séparateur sep entre chaque élément

Les éléments de lis doivent être de type str

```
l=["AA",20,31]
print("AA" in l)    # True
print(13 not in l) # True

l.append(20)
print(l)  # ['AA', 20, 31, 20]
l.insert(2,"kk")
print(l)  # ['AA', 20, 'kk', 31, 20]
l.extend([1,2])# ['AA', 20, 'kk', 31, 20, 1, 2]
l.sort() ⊗
l1=["BB","DD","AA"]
l1.sort()
print(l1)# ['AA', 'BB', 'DD']
l1.sort(reverse=True)  ⟵ Ordre décroissant
print(l1)# ['DD', 'BB', 'AA']

ch="-".join(l1)
print(ch)# DD-BB-AA
```


Opérations sur les listes

- lis.remove(elem) supprime l'élément elem de la liste lis (s'il existe)
- **del** lis
- lis.pop() supprime et retourne le dernier élément de lis
- lis.pop(i) supprime et retourne l'élément à la position i de lis
- lis.reverse() inverse l'ordre des éléments de la liste lis
- reversed (lis) affiche la liste inversé de lis sans l'affecter

```
l=[14, 7, 12.1, 7, "KK","DD"]
l.remove(3) ⊗
l.remove(7)
print(l)  # [14, 12.1, 7, 'KK', 'DD']

val1=l.pop()
val2=l.pop(2)
print(val1, val2, l)
  # DD 7 [14, 12.1, 'KK']
l.reverse()
print(l)  # ['KK', 12.1, 14]

reversed(l)
print(l)  # ['KK', 12.1, 14]
for i in reversed(l):
    print(i, end=" ")# 14 12.1 KK
```

```
del l # ?
del l[:] # ?
```

Listes en compréhension

Listes dont le contenu est défini par l'application d'une fonction à une séquence d'éléments (filtrage)

liste = [<expression> for <element> in <iterable> if <condition>]

Facultatif

```
liste = [2*i for i in range (0,6)]
print(liste)   # [0, 2, 4, 6, 8, 10]
liste = [i for i in range (0,10) if i%2!=0]
print(liste)   # [1, 3, 5, 7, 9]
liste = [0 if i%2!=0 else 1 for i in range (0,10) if i>0 ]
print(liste)   # [0, 1, 0, 1, 0, 1, 0, 1, 0]
liste = [[0 if i!=j else 1 for i in range (0,3)] for j in range (0,3)]
print(liste)   # [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```


Module random

Listes & Valeurs aléatoires

from random import *

randint(a, b)=randrange(a,b) génère un nombre entier aléatoirement dans l'intervalle [a,b]

randrange(a, b, **p**) génère un nombre entier aléatoirement dans l'intervalle [a,b] par un pas **p**

sample(seq, **n**) génère une liste de **n** valeurs aléatoires **sans répétition**, inclus dans une séquence seq

shuffle(lis) mélange aléatoirement lis

```
from random import *

L1 = [randrange(0,10,2) for i in range(5)]
print("L1=",L1) # L1= [6, 2, 0, 8, 4]
L2 = [randint(0,10) for i in range(5)]
print("L2=",L2) # L2= [6, 2, 0, 8, 4]

L3 = sample(range(1,30), 5)
L4 = sample([30,46,33,21,78,34,35], 5)
print("L3=",L3) # L3= [8, 2, 21, 23, 26]
print("L4=",L4) # L4= [33, 21, 35, 46, 34]
```

```
L5 = ['aaa',4,'bbbb',8,10]
shuffle(L5) ; print(L5)
        # ['aaa', 8, 10, 'bbbb', 4 ]
```


Quiz

1. Étant donné une liste **l**, l'instruction **l=l[::-1]** est équivalent à (réponse multiple)

- ☐ reversed(l)
- ☑ l.reverse()
- ☐ l=l[-1::]
- ☑ l=l[-1::-1]

2. Quelle la sortie de l'instruction **print(list("Python"))** (une seule réponse)

- ☐ ["Python"]
- ☐ "Python"
- ☑ ['P', 'y', 't', 'h', 'o', 'n']

3. Quelle est la sortie du code suivant ? (une seule réponse)

```
l = ['ap', 'ce']
for i in l:
    i=i.upper()
print(l)
```

- ☑ ['ap', 'ce']
- ☐ ['AP', 'CE']
- ☐ 'AP' 'CE'

Tuples

Utilisent des () au lieu des [], elles sont facultatives mais recommandées

Peuvent contenir des objets de types différents

t1=() #ou bien **t1=tuple()** ⟵ **Un Tuple vide se définit par () ou tuple()**

t1=(20, "ABC", 3.2) # **ou bien t1=** 20, "ABC", 3.2
print(**t1**) **# (20, 'ABC', 3.2)**

t1=20, ⟵ **Un Tuple d'un seul élément (virgule obligatoire)**

t2=(42,12,55)
t3=(t2,t3) # **ou bien t3=t2,t3** ⟵ Tuple de Tuples
print(**t3**) **# ((42, 12, 55), (2, 14))**

Tuples

```
t4=(22,"ABC",[24,13],14)
t4[0]=11
t4[2]=99
```
⊗ ⟵ Les éléments d'un Tuple sont immutables

```
t5=(2,14) ; print(id(t5))   # 1828081314184
t5=t5+t4  print(t3)  # (2, 14, 22,"ABC",[24,13],14) ⟵ concaténation
print(id(t5))  # 1828062559400
```

```
print(id(t4)) # 1870540037960
t4[2][0]=99 ; print(t4, id(t4)) # (22, 'ABC', [99, 13], 14) 1870540037960
t4[2].append("CB") ;print(t4, id(t4)) #(22, 'ABC', [99, 13, 'CB'], 14) 1870540037960
t4[2]+=["CB"]
```
⊗

Le tuple est immutable ***mais*** peut contenir des éléments qui sont mutables

Empaquetage & Dépaquetage

L'utilisation de la virgule lors de la création d'un Tuple, permettant de séparer les éléments les uns des autres, s'appelle empaquetage automatique

```
t=2, 5, 7 ; print(t)
```
⟵ Empaquetage automatique

L'utilisation de la virgule dans la partie gauche d'une assignation dont l'expression droite retourne un Tuple s'appelle dépaquetage automatique

```
a, b, c=t
print("a=", a)  # a= 2
print("b=", b)  # b= 5
print("c=", c)  # c= 7
```
⟵ Dépaquetage automatique

*var & _

Le symbole _ peut être utilisé comme un nom de variable jetable si l'on a seulement besoin de quelques éléments d'un Tuple

Un préfixe *var convertit var en une liste

```
a=1,2,3,4 ; x,_,y,_=a
print(x,y)# 1 3

a,*b,c=1,2,3,4
print(a,b,c, sep="   |   ") # 1 | [2, 3] | 4

*b,c=1,2,3,4
print(b,c, sep="   |   ")  # [1, 2, 3] | 4
```
b est une variable de type list

```
a,b=1,2,3,4 ; print(a, b)
```
⊗ Too many values to unpack (expected 2)

```
*b=1,2,3,4; print(b)
```
⊗ Starred assignment target must be in a list or tuple

```
*b,=1,2,3,4 ; print(b)   # [1,2,3,4]
```


Opérations sur les Tuples

- len(tup) calcule la longueur du Tuple tup
- min(tup) retourne l'élément le plus petit dans tup
- max(tup) retourne l'élément le plus grand dans tup
- tup.count(elem) retourne le nombre d'occurrences de l'élément elem dans tup
- tup.index(elem) : retourne la première occurrence de l'élément elem s'il existe dans tup, Erreur sinon
- sep.join(tup) convertie le Tuple de chaînes de caractères lis en une chaine de caractères en ajoutant un séparateur sep entre chaque élément

```
t=(11,29,"EDz",19.65,"EDz")
t1=(5,10)
print(len(t))   # 5
print(max(t1)) # 10
print(max(t))
print(t.count("aaa")) # 0
print(t.count("EDz")) # 2
```

```
print(t.index("E"))
```
⊗

```
t3=("DDD","FFF")
ch="--".join(t3) ; print(ch)
        # DDD--FFF
```

Quiz

1. Un Tuple est-elle mutable ? (une seule réponse)
   - ☐ True
   - ☑ False
2. Quelle est la sortie du code suivant ? (Choix multiple)

```
t=(20,(4,4)),8
t[1][0]=1
print(t)
```

   - ☑ Erreur
   - ☐ (20,(4,4),8)
   - ☐ (20,(1,4),8)
   - ☐ [20,(1,4)]
3. Soit **k,*g,m=[(i,i*i) for i in range(2)]+[98,7]** (choix multiple)
   - ☑ print(**k**) affiche (0, 0)
   - ☑ print(**g**) affiche [(1, 1), 98]
   - ☑ print(**m**) affiche 7

| Ensembles & Dictionnaires

Ensembles

Représentent une collection d'objets de **différents types**, **sans répétition** (sans duplicata) et **sans ordre** (sans Indiçage)

Un ensemble est déclaré par une série de valeurs, séparée par des virgules, et encadrée par des accolades {}

```
ens={1,12.4,3,"AA"}; print(ens) # {1,12.4,3,"AA"}
ens=set(); print(ens, type(ens)) # set() <class 'set'>
ens={}; print(ens, type(ens)) # {} <class 'dict'>
```

Un ensemble vide se définit par set() et non {}

```
ens1={1,4} ; print(id(ens1)) # 1793566107688
ens1=ens1|{1,6}   ← Concaténation (union) par | et non plus par +
print(ens1, id(ens1)) # {1, 4, 6} 1793566108584
```

Un Ensemble et ses éléments sont immutables

print (**ens1[1]**) ⊗ **pas d'indice**

Ensembles

```
ens={12,(55,"BC")}; print(ens) # {12,(55,"BC")}   ← Ensemble de Tuples
ens={12,[55,"BC"]}; print(ens) ⊗   ← Ensemble de listes
ens={12,{55,"BC"}}; print(ens) ⊗   ← Ensemble d'ensembles
```

On peut faire des ensembles de Tuples mais pas d'ensembles dont les éléments sont des listes ou des ensembles !

Un ensemble contient des objets uniques qui sont hashable

```
ens=set([1,"aa",38,"k"]) ; print(ens) # {1,"aa",38,"k"}
ens=set((1,"aa",38,"k")) ; print(ens) # {1,"aa",38,"k"}
ens= set("programme") ; print(ens) # {'a', 'g', 'p', 'o', 'm', 'e', 'r'}
ens= set(range(5)) ; print(ens) # {0, 1, 2, 3, 4}
```

Pas de duplicata
Pas d'ordre

```
ens={i for i in range(5) if i%2==0}; print(ens) # {0, 2, 4}
```

← Ensemble en compréhension

Opérations sur les ensembles

- len(ens) renvoie la longueur de l'ensemble ens
- max(ens), min(ens), sum(ens)
- ens.add(elem) ajout l'élément elem à l'ensemble ens
- ens.update(elem1,elem2,...) ajout plusieurs éléments **iterable** à l'ensemble ens
- ens.remove(elem) supprime l'élément elem de l'ensemble ens (**s'il existe**)
- ens.pop() renvoie la première élément de l'ensemble ens, et le **supprime**

```
e={10,12,4}
print(len(e)) # 3
print(max(e),min(e),sum(e),sep="--")
            # 12--4--26
e.add(13) ; print(e)
e.update("G",["Z",9]); print(e)
        # {4, 'Z', 9, 'G', 10, 12, 13}
```

```
e.add(13,"DD")      ⊗
e.update(15,[3,8])  ⊗
```

```
e.remove(12)
print(e)  # {4, 'Z', 9, 'G', 10, 13}

val=e.pop()
print(val, e) # 4 {'Z', 9, 'G', 10, 13}
```


Opérations sur les ensembles

- elem in ens, elem not in ens : vérifie l'appartenance d'un élément elem dans l'ensemble ens
- ens.copy() copie les éléments de l'ensemble ens
- ens.clear() efface le contenu de de l'ensemble ens. Elle retourne **set()**
- ens1 <= ens2 **ou bien** ens1.issubset(ens2) **ou bien** ens2.issuperset(ens1) vérifie si ens1 est sous-ensemble de ens2

  NB. Un ensemble ens1 est un sous-ensemble de ens2 si chaque élément de ens1 est également un élément de ens2

```
e={'Z', 9, 'G', 10, 13}
```

```
print(13 in e, 12 not in e)
        # True True
```

```
e1 = e.copy() ; print(e1)
    # {'Z', 9, 'G', 10, 13}
e.clear() ; print (e) # set()
```

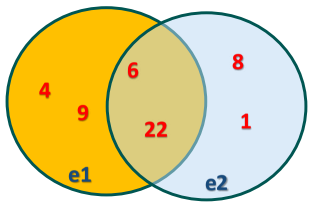
```
print(set(['Z',10]) in e) # False
```

```
ens1={10, 7}
ens2={1, 10, 7, 22}
print(ens1.issubset(ens2))# True
```


Opérations sur les ensembles

- ens1==ens2 retourne **True** si les ensembles ens1 et ens2 contiennent les mêmes éléments
- ens1!=ens2
- ens1|ens2 ou bien ens1.union(ens2) retourne un ensemble qui est l'union des ensembles ens1 et ens2
- ens1&ens2 ou bien ens1.intersection(ens2)
- ens1-ens2 ou bien ens1.difference(ens2)
- ens1^ens2 ou bien ens1.symmetric_difference(ens2)

```
e1 = set([4, 6, 9,22])
e2 = set([1, 6, 8,22])
print(e1|e2)
    # {1, 4, 6, 8, 9, 22}
print(e1&e2)
    # {6, 22}
print(e1-e2, e2-e1)
    # {9, 4} {8, 1}
print(e1^e2)
    # {1, 4, 8, 9}
```



Quiz

1. Un ensemble est-elle mutable ? (une seule réponse)
   - ☐ True
   - ☑ False
2. Quelle est la sortie du code suivant ? (une seule réponse)

```
e=set([22,"BK",22])
e.update((22,50),"Z")
e.add((13,))
print(e)
```

   - ☐ {'Z', 'BK', (22, 50), (13,), 22}
   - ☐ {'Z', 'BK', 13, (22, 50), 22}
   - ☐ {'Z', 'BK', 50, 22, (13,), 22}
   - ☑ {'Z', 'BK', 50, 22, (13,)}
3. Par quoi remplacer le vide pour que le code affiche **{2,3}**? (réponse multiple)

```
d={1,2,3}
_____________________
print(d)
```

   - ☑ d.intersection({2,3,4,5})
   - ☑ d &{2,3,4,5}
   - ☐ d|{2,3,4,5}

Dictionnaires

Représentent une structure de type **clé**:valeur

```
d={"a":2, 12:"c"}; print(d) # {'a': 2, 12: 'c'}
```

```
d=dict(); print(d, type(d)) # {} <class 'dict'>
d={}; print(d, type(d)) # {} <class 'dict'>
```

Un dictionnaire vide se définit par {} ou dict()

Le **seul moyen d'accès** à une valeur particulière est par l'intermédiaire de sa **clé**

dic[cle] renvoie la valeur associée à la clé cle du dictionnaire dic

```
d1={"z":2, "d":8, 12:["AA",3]}
print(d1["d"]) # 8
print(d1[12]) # ['AA', 3]
print(d1["GG"]) ⊗
```


Opérations sur les dictionnaires

Ajout et MAJ d'un valeur

dic[cle]=elem modifie la valeur associée à la clé cle à elem (si cle existe) **ou bien** ajoute cle:elem au dictionnaire dic (sinon)

```
d1={"a":2, "b":8}; print(d1, id(d1)) # {'a': 2, 'b': 8} 1793565906984
d1["c"]=1 ← Ajout d'un élément de clé "c" et de valeur 1
d1["a"]=4 ← MAJ le valeur de l'élément de clé "a" à 4
print(d1, id(d1)) #{'a': 4, 'b': 8, 'c': 1} 1793565906984
```

Un dictionnaire et ses éléments sont mutables

dic.update({cle:val}) ajoute un élément {cle:val} au dictionnaire dic si la clé n'existe pas

```
d1.update({'a':10}); print(d1) # {'a': 10, 'b': 8, 'c': 1}
d1.update({'m':10}); print(d1) # {'a': 10, 'b': 8, 'c': 1, 'm': 10}
d1.update({'a':10},{'K':3}) ⊗
```


Opérations sur les dictionnaires

- len(dic) renvoie la longueur du dictionnaire dic (nombre de clés)
- cle in dic renvoie True si la clé cle est présente dans le dictionnaire dic, False sinon
- dic.copy() renvoie une copie du dictionnaire dic, indépendante de l'original
- dic.pop(cle) renvoie la valeur et supprime la paire **cle:valeur**
- del dic[cle] efface une paire cle:valeur
- dic.clear() efface le contenu du dictionnaire dic

```
d={"chien":"Dog", "souris":"mouse"}
print(len(d)) # 2
print("Chien" in d, "chien" in d)
 # False True
d_c=d.copy()
print(id(d), id(d_c))
 # 1296344277352 1296344345840
print(d_c)# {'chien': 'Dog', 'souris': 'mouse'}

val=d.pop("chien")
print(val,d) # Dog {'souris': 'mouse'}

del d_c["chien"]
print(d_c) # {'souris': 'mouse'}

d_c.clear()
print(d_c) # {}
```


Parcours d'un dictionnaire

- dic.items() renvoie un **itérable** pour décrire les couples **cle**:**valeur** du dictionnaire dic dans une boucle for

```
d={"chien":"Dog", "souris": "Mouse"}
```

```
for i in d.items():
    print(i)
```

```
('chien', 'Dog')
('souris', 'Mouse')
```

```
for i,j in d.items():
    print([i,j])
```

```
['chien', 'Dog']
['Souris', 'Mouse']
```

- dic.keys() renvoie un **itérable** pour décrire les **clés** de dic dans une boucle for
- dic.values() renvoie un **itérable** pour décrire les valeurs de dic dans une boucle for

```
for i in d.keys():
    print(i, end="-") # chien-souris
for i in d.values():
    print(i, end="-") # Dog-Mouse
```

Quiz

1. Par quoi remplacer le vide pour que le code affiche le dictionnaire **{2: 5, 1: (7, 10)}** ? (une seule réponse)

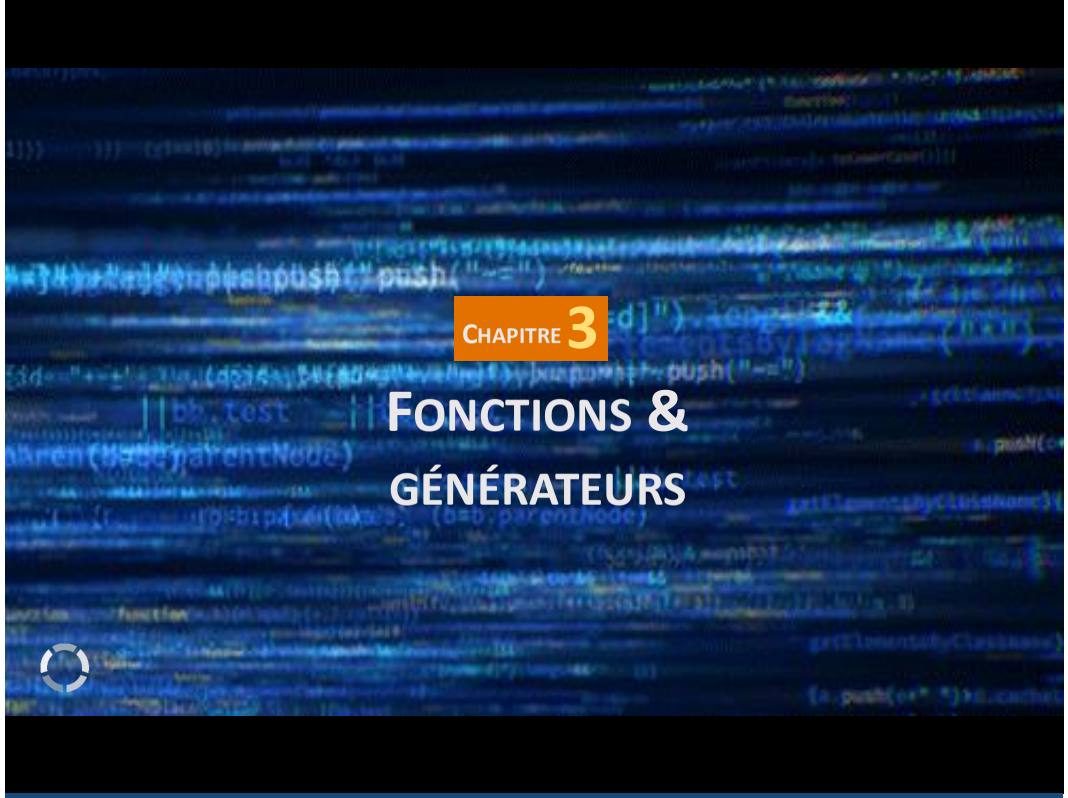
```
x=____________________
x[2]=5
x[1]=7,10
print(x)
```

- ☐ set()
- ☐ {i for i in [2,5,1,(7,10)]}
- ☐ {'2':5, '1':(7,10)}
- ☑ {}

2. Par quoi remplacer le vide pour que le code affiche [1, 2] ? (réponse multiple)

```
d={1:"AA", 2:"CC"}
print(____________________)
```

- ☐ [i for i in d.items()]
- ☑ [j for j,i in d.items()]
- ☑ list(d)
- ☐ [[i for i in d.keys()]]



Fonctions

- Représentent des constructions pour structurer les programmes
- permettent d'utiliser le code à plusieurs endroits d'un programme

```
def <non_fonction>(<Liste_paramètres>) :
⟷ instructions
```

```
def calculer(x,y):
        return x*y
def text():
        print("XX")
```

Signature

Fonctions

- Pas de distinction entre **procédure** et **fonction**
- Le mot clé *return* est **facultatif**. Il termine l'exécution de l'appel de fonction et renvoie le résultat
- S'il n'y a pas *return*, la fonction renvoie l'objet ***None***

Définition de la fonction

```
def f1(x,y,z):
    return x+y+z
```

Paramètres
Les noms que l'on spécifie dans la signature de la fonction

Invocation de la fonction

```
a=f1(5,4,1)
print(a)
```

Arguments
Les valeurs passées à l'appel de la fonction

Paramètres & Arguments

Deux types de paramètres :

- **Optionnels** : qui ont une valeur attribuée par défaut
- **Obligatoires** : qui n'ont pas une valeur attribuée par défaut

```
def f1(p1, p2='val', p3=3):
    print(p1, p2, p3)
```

Deux types d'arguments :

- **Nommés (keyword arguments)** : sont précédés par un identifiant
- **Positionnels (positional arguments)** : qui ne sont pas nommés

```
f1(20,30,50) # 20 30 50
f1(p1=20,p2=30,p3=50) # 20 30 50
f1(20,30,p3=50) # 20 30 50
```


Propriétés (1/2)

1. Les arguments correspondants aux paramètres optionnels, s'ils sont fournis; Leurs valeurs seront adaptées, sinon elles gardent les valeurs attribuées par défaut

2. Un argument peut être positionnel ou nommé que ce soit il correspond à un paramètre optionnel ou obligatoire

```
def calculer(a, b=5,c=3):
    print(a+b+c)

calculer(10) # 18
calculer(a=12,b=2,c=1) # 15
calculer(12,b=2,c=1) # 15
calculer(12,2,c=1) # 15
calculer(12,2,1) # 15
```

3. Les arguments correspondants aux paramètres obligatoires doivent être fournis lors de l'invocation (appel) de la fonction

calculer() ⊗

Propriétés (2/2)

4. Les arguments positionnels doit être fournis avant les arguments nommés

calculer(a=12,b=2, 1) ⊗ **L'ordre des arguments est très important (obligatoires avant optionnels) !**

5. Un argument optionnel doit avoir le même nom de paramètre fourni lors de la définition de la fonction

calculer(z=12, e=10, f=15) ⊗

```
def calculer(a, b=5,c=3):
    ⋮
```


Passage du nombre variable d'arguments!

```
def f1(*args):
    print(args)
```

Convertie les arguments non nommés en Tuple

```
f1(5,20,30) # (5, 20, 30)
f1(a=5,b=20,c=30) ⊗
```

```
def f1(**args):
    print(args)
```

Convertie les arguments nommés en dictionnaire

```
f1(5,20,30) ⊗
f1(a=5,b=20,c=30) # {'a': 5, 'b': 20, 'c': 30}
```

NB. Les paramètres extensibles doivent être en dernier

~~def f2(*args, x):~~

Passage des arguments étoilés

```
def f1(a,b,c=1,d="XY"):
    print("a=",a ,"b=",b, "c=",c, "d=",d)
```

f1(20,30) **#a= 20 b= 30 c= 1 d= XY**
Équivalent à
f1(*[20,30]) **#a= 20 b= 30 c= 1 d= XY** ←

Lors de l'appel d'une fonction, il est possible de passer les arguments **non nommés** sous forme d'une **séquence** préfixée d'une *

f1(**{'a':20,'b':30}) **#a= 20 b= 30 c= 1 d= XY** ←

```
def f1(a,b,c=1,d="XY"):
        ⋮
```

Lors de l'appel de fonction, il est possible de passer les arguments **nommés** sous la forme d'un **dictionnaire** préfixé de **

NB. Chaque clé du dictionnaire correspond au même nom de paramètre fourni lors de la définition de la fonction

Signature universelle

```
def f(*args, **kwargs):
    print("args=", args,"kwargs=",kwargs)
```

 # Signature qui accepte tout type de paramètres, passé de n'importe quelle manière

f(1,2, a=3, b=5) **# args= (1, 2) kwargs= {'a': 3, 'b': 5}**

f(3,4,*[1,5],**{'z':1,'f':2},a=10) **# args= (3, 4, 1, 5) kwargs= {'z': 1, 'f': 2, 'a': 10}**

f(**{'z':1,'f':2},a=10) **# args= () kwargs= {'z': 1, 'f': 2, 'a': 10}**

Quiz

1. Quelle est la sortie du code suivant ? (une seule réponse)

```
def f1(x):
    x = list(set(x))
    return x
def f2(z,*a):
    print(z*sum(a))
x = 1, 2, 2 ; f2(f1(x),1,2)
```

- ☐ 3
- ☐ 6
- ☐ [1,2,1,2]
- ☑ [1,2,1,2,1,2]

2. Par quoi remplacer le vide pour que le code affiche (1, 2, 3) ? (réponse multiple)

```
def f(*args,x=0,y=0,z=0):
    print(args)
_________________
```

- ☐ f(**{'x':1,'y':2, 'z':3})
- ☑ f(1,2,3)
- ☑ f(*[1,2,3])
- ☐ f(*[[1,2,3]])

Activité1

Écrire une fonction **g()** qui se comporte de la façon suivante :

g() # Hello
g('Salut') # Salut
g('Salut', Ahmed=3) # Salut Ahmed Salut Ahmed Salut Ahmed
g('Ciao', Ahmed=2, Karim=2) #Ciao Ahmed Ciao Ahmed Ciao Karim Ciao Karim

```
def g(a="Hello",**args):
    if args=={}: # ou bien # not args
        print(a)
    else:
        for i,j in args.items():
            print((a+" "+i+" ")*j, end="")
        print()
```

```
def g(**args, a="Hello"): #?
    ⋮
```

Activité2

Écrire une fonction **pair()** qui accepte plusieurs nombres en argument et renvoie une liste avec les nombres pairs. Si on ne passe pas d'argument, elle renvoie None

```
def pair(*args):
    L=[i for i in args if i%2==0]
    if L!=[]:
        return L
```

```
print(pair(1,4,6))   # [4,6]
print(pair(*[1,4,6])) # [4,6]
print(pair(4))   # [4]

print(pair()) # None
```


Variable locale/ globale

- **Variable locale** : Elle est créée dans une fonction; Elle n'est pas visible à l'extérieur d'une fonction

```
def afficher():
    y=5          <-- Variable locale
    print(locals()) <-- retourne le dictionnaire des objets locaux de la fonction courante
afficher() # {'y': 5}
print(y)  ⊗
```

- **Variable globale** : Elle est créée dans le programme principal; Elle sera visible partout dans le programme

```
def afficher():
    print(y+1)
y=5
afficher() # 6
print(y) # 5
```


Variable globale

1. Une variable globale ne peut pas être modifiée sans le mot clé **global**

```
def afficher(e):
    global c
    c+=1
    print(c,e)
c=3
afficher(10) # 4 10
```

2. La modification d'un objet **mutable** dans une fonction créera un pointer sur même id vers la variable globale

```
def ajouter(val, L=[3,4]):
    L+=[val]      <-- locale
    return L      <-- globale
L=ajouter(5); print(L, id(L))
# [3, 4, 5] 2661477592456
L=ajouter(6); print(L, id(L))
# [3, 4, 5, 6] 2661477592456
```


Fonctions anonymes

- Un autre moyen de créer des fonctions **courtes** et **pratique**, limitées à une seule instruction

lambda par1, par2, ..., parN: instruction de retour

```
f=lambda x : x*2 ; print(f)
    # <function <lambda> at 0x00000242391AE7B8>
print(f(3)) # 6

f=lambda x : 10 if x >0  else -10 ; print(f)
    # <function <lambda> at 0x00000242391AE598 >
print(f(20)) # 10

val = (lambda x,y : sum([v for v in range(x,y)]))(2,6)
print(val) # 14
```

[2,3,4,5]

Fonction map

map(fct,it) applique une fonction fct sur toutes les valeurs renvoyés par un itérable it. Elle retourne l'itérateur sur les valeurs fct(i) pour i parcourant it

```
f= lambda x:x**2
m=map(f,range(1,4))
print(m)
# <map object at 0x00000180F2AF5780>
print(list(m))
# [1, 4, 9]
m1=map(f,range(1,4))
for j in m1:
    print(j,end=" ")
 # 1 4 9
#print(list(m1)) # []
```

```
numbers = (1, 2, 3, 4)
result = map(lambda x: x + x, numbers)
print(list(result))   # [2, 4, 6, 8]
```

```
l1=[1,2,3,11]
t1=4,5,6
result=map(lambda x,y: x+y ,l1,t1)
print(list(result))# [5, 7, 9]
```


help(nom_de_la_ fonction)

help(nom_de_la_ fonction) affiche le **docstrings** de la fonction

```
def calculer(x,z):
    """retourne le produit de deux entiers donnés en paramètres"""
    return x*y
help(calculer)
```

docstrings

```
Help on function calculer in module __main__:

calculer(x, z)
    retourne le produit de deux entiers donnés en paramètres
```


Quiz

1. Laquelle des instructions suivantes est équivalent à **list(map(lambda x: x**2, [1, 2, 3]))** ? (une seule réponse)

- ☐ list(range(1,4)**2)
- ☐ [x**2 for x in [[1,2,3]]]
- ☑ [x**2 for x in [1,2,3]]
- ☐ [x for x in lambda x: x**2]

2. Quelle est la sortie du code suivant ? (une seule réponse)

```
def f(i, l = []):
    l.append(i) ; return l
f(1); f(2) ; x = f(3) ; print(x)
```

- ☑ [1,2,3]
- ☐ [3]
- ☐ [[1],[2],[3]]

3. Est-ce que le code suivant affiche **3** ? (une seule réponse)

```
def fun(f, val):
    print(f(val))
fun(max, [1, 2, 3])
```

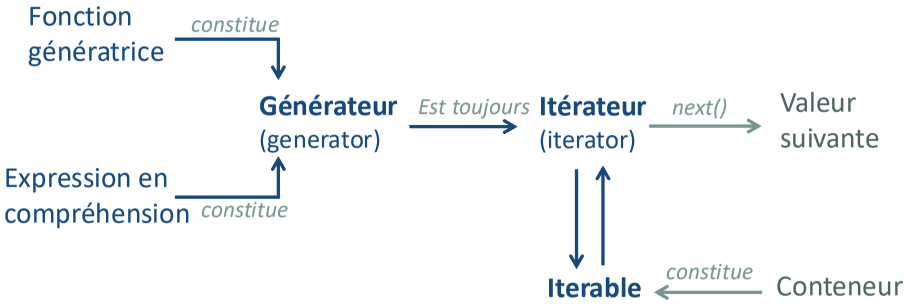
- ☑ Vrai
- ☐ Faux

Activité

Écrire une **fonction anonyme** qui permet de **supprimer à partir d'un mot donné tous les chiffres de 0 à 9**. Tester cette fonction

```
f=lambda mot:"".join([c for c in mot if not c.isdigit()])

f("sdf23 5ssf")
```

Itérateur & Générateurs



Un itérateur permet de parcourir des objets itérables

Un générateur constitue un moyen plus pratique de créer et manipuler des itérateurs

Fonction générateur (1/3)

```
def gen():
    yield "a"
    yield "b"
```

a. Un générateur commence son exécution et renvoie la valeur donnée par l'instruction **yield**, puis, attend qu'on lui redonne la main pour calculer la *valeur suivante*

```
g=gen() ; print(type(g)) # <class 'generator'>
print(next(g)) # a
```

b. À chaque appel de la fonction prédéfinie next(), le code sera exécuté jusqu'au prochain **yield**

```
print(next(g)) # b
```

c. À la fin de l'exécution de la fonction générateur, Python lève automatiquement l'exception **StopIteration**

```
print(next(g)) # ?
```


Fonction générateur (2/3)

```
def gen1():
    yield 1
    yield 2
    yield 3

g1=gen1()
for i in g1:
    print(i) # ?
```

La boucle **for** appelle implicitement **next()** et reconnait à travers l'exception **StopIteration** que le générateur n'a plus de valeurs

→Pas d'erreur

Fonction générateur (3/3)

```
def gen2():
    for i in range(2):
        yield
        yield i
        return 0

g2=gen2()
for j in g2:
    print(j)# ?
```

La valeur retournée par le générateur sera celle apposée au **yield**, ou **None** dans le cas où aucune valeur n'est spécifiée à **yield**

return aura pour effet de lever **StopIteration**

yield from

yield from <expr> : utilisé dans le corps d'un générateur

<expr> doit être une expression évaluée en itérable

yield from <expr> est disponible à partir de la version 3.3 de Python

```
def gen4():
    for j in "Python":
        yield j

#Équivalent à
def gen5():
    yield from "Python"

--------
for i in gen4():
    print(i, end=" ") # P y t h o n
for i in gen5():
    print(i, end=" ") # P y t h o n
```


Expression en compréhension

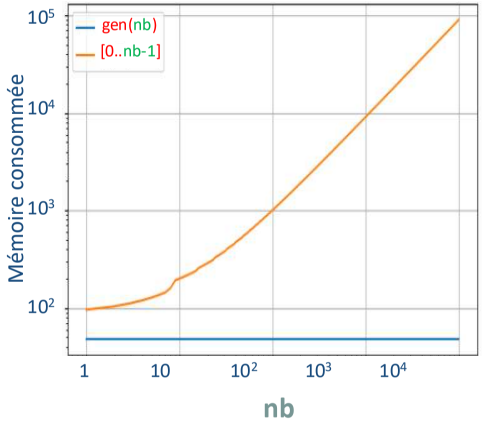
```
nb=3
l=[i for i in range(nb)]
print(type(l)) # <class 'list'>
#vs.
g=(i for i in range(nb))
print(type(g)) # <class 'generator'>
---
for j in l:
    print(j) #?
for j in g:
    print(j) #?
```

Expression génératrice

NB.

Pour un **générateur**, la mémoire consommée est **indépendante** de nb

Pour une **liste**, la mémoire consommée **croît linéairement** de nb



Quiz

1. Quelle est la sortie de l'instruction suivante ? (une seule réponse)

```
print("".join([str(j) for j in (i for i in range(1,4,2))]))
```

- ☐ 123
- ☐ [1,2,3]
- ☑ 13
- ☐ [1,3]

2. Quelle est la sortie du code suivant ? (une seule réponse)

```
def gen(g):
    yield from g
def gen1():
    for i in ["aa","ab"]:
        yield i ; return 1
for j in gen1():
    v=gen(j)
    print(next(v))
```

- ☐ ab
- ☐ aa
- ☑ a
- ☐ b

Activité

Créer un générateur **gen_lettre_chaine(ch)** représentant les lettres d'une chaine de caractère **ch** données en paramètre. Les espaces blancs ne seront pas considérées comme des lettres

Par exemple, le parcours sur **gen_lettre_chaine('A aa BB C ')** retourne A puis a puis a puis B puis B puis C

```
def gen_lettre_chaine(ch):
    ch=ch.replace(" ","")
    yield from ch

for i in gen_lettre_chaine('A aa BB C '):
    print(i)
```

Accès à un fichier

- Assuré par l'intermédiaire d'un objet **Fichier**, crée par un appel à la fonction **open()**

```
unFichier = open('chemin/nom_fichier.txt', mode ='r' )
```

	Lecture	Ecriture	Ajout	
'r'	☑	☒	☒	Pointeur placé en début du fichier
'r+'	☑	☑	☒	
'w'	☒	☑	☒	Ancien contenu perdu Fichier créé s'il n'existe pas
'w+'	☑	☑	☒	
'a'	☒	☒	☑	Pointeur placé en fin du fichier Fichier créé s'il n'existe pas
'a+'	☑	☒	☑	

Fermeture d'un fichier

- f.close() ferme un fichier f (libérer les ressources allouées pour la gestion de fichiers)

NB. N'oubliez pas de fermer un fichier après l'avoir ouvert !

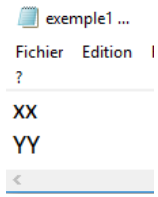
Si d'autres applications, ou d'autres partie de code, souhaitent accéder à ce fichier, ils ne pourront pas car le fichier sera déjà ouvert

```
f=open('chemin/exemple.txt')
# Traitement
....
f.close()
```


Lecture d'un fichier (1/2)

- f.read() lit un fichier f et renvoie sous forme d'une chaine de caractères **toutes les lignes** de f
- f.read(nb) lit **au maximum** un nombre nb de caractères dans f

```
f=open('chemin/exemple1.txt')
# Équivalent à f=open('chemin/exemple.txt', 'r' )
print(type(f))   # <class '_io.TextIOWrapper'>
contenu=f.read()
contenu #'xx\nYY'

print(contenu) #?
f.close()
------
f=open('chemin/exemple.txt')
f.read(4) #'xx\nY'
-------
f.close()
```



Lecture d'un fichier (2/2)

- f.readlines() lit un fichier f et renvoie une liste de chaine de caractères dont chaque élément représente une ligne
- f.readline() lit une ligne du fichier f et la renvoie sous forme d'une chaine de caractères

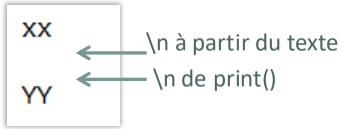
```
f=open('chemin/exemple1.txt')
contenu=f.readlines()
print(contenu)  # ['xx\n', 'YY']
f.close()
# Tant qu'il y a un saut de ligne dans le fichier, il est marqué par \n à la fin de chaque élément de la liste
```

```
f=open('chemin/exemple1.txt')
f.readline()
'xx\n'
f.readline()
'YY'
f.readline()
''
f.close()
# À chaque nouvel appel de readline() le curseur passe à la ligne suivante et la renvoie
# Si la fin du fichier est atteinte la fonction readline() retourne une chaine vide
```


Parcours d'un fichier

- Un fichier un type itérable
- Il peut être directement lu ligne par ligne par une boucle **for**

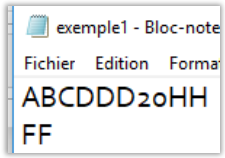
```
f=open('chemin/exemple1.txt', 'r' )
for ligne in f:
    print(ligne)

f.close()
```



Ecriture d'un fichier

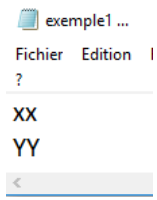
- f.write(ch) écrit la chaine ch dans le fichier f
- f.writelines(lis) écrit les éléments de la liste lis transmise (un par un) (**pas de retour à la ligne automatique à la fin de chaque élément**)

```
f=open('chemin/exemple1.txt', 'w' )
f.write('ABC')
f.write('DDD')
f.writelines(['20', 'HH\n', 'FF'])

f.writelines([20,10])

f.close()

# Les éléments de la liste transmise à un fichier doivent être des chaines de caractères
# si le fichier n'est pas fermé, alors aucun élément sera ajouté
```



Gestion du curseur dans un fichier

- f.tell() : renvoie la **position actuelle** du curseur de lecture / écriture dans le fichier f
- f.seek(i) : met le curseur de lecture / écriture dans le fichier f à la position i




```
f=open('chemin/exemple1.txt')
print(f.tell(), f.readline())
print(f.tell(), f.readline())
f.close()
```

```
0 XX

4 YY
```

```
f=open('chemin/exemple1.txt')
print(f.read())
f.seek(5)
print(f.read())
f.close()
```

```
XX
YY
Y
```

Méthode optimisée d'ouverture et fermeture

L'utilisation de **with** avec **open()** garantit la bonne fermeture du fichier

```
f=open('chemin/exemple2.txt', 'w+' )
f.write('ABC\n')
f.write('DDD')
f.seek(0)
print(f.read())
f.close()
```

 # Équivalent à

```
with open('chemin/exemple2.txt', 'w+' ) as f :
    f.write('ABC\n')
    f.write('DDD')
    f.seek(0)
    print(f.read())
```


Module csv Lecture

```
from csv import *

f=open('chemin/exemple2.csv', 'r+')
data= reader(f, delimiter=",")
# Par défaut delimiter=","
for c in data:
    print(c) # retourne une liste qui comporte les éléments
             de la ligne courante

# Il faut obligatoirement fermer le fichier après utilisation
f.close()

# Équivalent à
with open('chemin/exemple2.csv', 'r+') as f:
    data= reader(f, delimiter=",")
    for c in data:
        print(c)
```

exemple2.csv

```
xx,zz,ee
YY,14,17
```

```
['xx', 'zz', 'ee']
['YY', '14', '17']
```


Module csv Ecriture

```
from csv import *

with open('chemin/exemple3.csv', 'w', newline='') as f:
    wr= writer(f, delimiter=';')
    wr.writerow(['XXX','ZZZ',14])
    wr. writerow(['RRR', '2AZE', 123,77])
    #Si newline='' n'est pas spécifié, une ligne vide sera ajouter après
     chaque appel de writerow()
```

* Ecriture à partir d'un dictionnaire

```
with open('chemin/exemple4.csv', 'w', newline='') as f:
    wr= DictWriter(f, fieldnames=['col_1', 'col_2'], delimiter=";")
    wr.writeheader()
    wr.writerow({'col_1': 'XXX', 'col_2': 'YYY'})
    wr.writerow({'col_1': '456', 'col_2': '123'})
```


Quiz

1. Quelle fonction est utilisée pour ouvrir le fichier en lecture en Python ?
(une seule réponse)

- ☐ fopen(file_name, mode)
- ☑ open(file_name, mode)
- ☐ openfile(file_name, mode)
- ☐ open_file(file_name, mode)

2. Étant donné le code suivant (réponse multiple)

```
with open('chemin/exemple1.txt', 'w') as f:
    for ligne in f:
        print(ligne)
```

- ☑ Le code engendre une erreur
- ☑ La fermeture du fichier se fait automatiquement
- ☑ Il faut utiliser le mode 'r' ou 'w+' pour que le code sera fonctionnel
- ☑ Le parcours du fichier se fait ligne par ligne

Exception

- Une exception correspond à une erreur détectée durant l'exécution
- Une exception interrompue l'exécution de programme→ Un message d'erreur plus ou moins explicite est affiché
- Une exception n'est pas toujours fatale

```
a=20/0 # ZeroDivisionError: division by zero
a=c+33 # NameError: name 'c' is not defined
a=int('1f3') # SyntaxError: invalid syntax
d={'r':11,'f':33}; print(d['k']) # keyError: 'k'
                    ⋮
```

NB. Gérer une exception permet d'intercepter une erreur pour éviter un arrêt du programme

Exception

- Toutes les exceptions levées par Python appartiennent à un ensemble d'exceptions nommé **Exception**
- Toutes les exceptions sont typées (avec un nom de type d'exception) pour éviter des erreurs silencieuses

ZeroDivisionError
NameError
keyError
IndexError
AttributeError
StopIteration
IndentationError
TypeError
ValueError
⋮

https://docs.python.org/3/library/exceptions.html

Syntaxe complète d'une exception

```
try:
        #Séquence normale d'exécution
except <exception_1> as e1:
        #Traitement de l'exception 1
except <exception_2> as e2:
        #Traitement de l'exception 2
else:
        #Clause exécutée en absence d'erreur
finally:
        #Clause toujours exécutée
```

← Facultatif

Gérer une exception!

Le mécanisme de gestion des exceptions s'effectue donc en **deux phases** :

1- La levée d'exception lors de la détection d'erreur (au niveau du bloc **try**)

```
try:
    #Séquence normale d'exécution
```

2- Le traitement approprié (au niveau du bloc **except**)

```
except <exception> as e:
    #Traitement de l'exception
```

Si **une erreur est détectée** (levée d'exception), elle est traitée dans le bloc **except** approprié (le gestionnaire d'exception)

Exemples (1/2)

```
try:
    x=2/0
except ZeroDivisionError as e1:
    print(e1)
except ValueError as e2:
    print(e2)
else:
    print("pas d'erreurs")
finally:
    print("toujours exécutée")
```

```
division by zero
toujours exécutée
```

```
try:
    int("zz3")
    y=3
except ZeroDivisionError as e1:
    print(e1)
except ValueError as e2:
    print(e2)
else:
    print("pas d'erreurs")
finally:
    print("toujours exécutée")
```

```
invalid literal for int() with base 10: 'zz3'
toujours exécutée
```

```
try:
    y=3
except ZeroDivisionError as e1:
    print(e1)
except ValueError as e2:
    print(e2)
else:
    print("pas d'erreurs")
finally:
    print("toujours exécutée")
```

```
pas d'erreurs
toujours exécutée
```


Exemples (2/2)

```
try:
    print(b)
    print (a/2)
except (ZeroDivisionError,NameError) as e:
    print(e)
```

Les parenthèses sont obligatoires

```
name 'b' is not defined
```

 # Équivalent à ≡

```
try:
    print(b)
    print (a/2)
except ZeroDivisionError as e1:
    print(e1)
except NameError as e2:
    print(e2)
```

```
name 'b' is not defined
```


Déclencher une exception!

L'instruction **raise** permet de lever volontairement une exception

```
x = -2
if not(x>=0) :
    raise ValueError("x est négatif")
```

```
ValueError
<ipython-input-4-86bab1b9d2f0> in <module>
      1 x = -2
      2 if not(x>=0) :
----> 3     raise ValueError("x est négatif")

ValueError: x est négatif
```

```
try:
    x = -2
    if not(x>=0) :
        raise ValueError("x est négatif")
except Exception as e:
    print(e)
```

```
x est négatif
```

NB. L'instruction **raise** peut être trouver à tout endroit du code, pas seulement dans un bloc **try**

AssertionError

- Une **assertion**, décrit par **l'instruction assert**, est un moyen de s'assurer, *avant de continuer*, qu'une **condition** est respectée
- **AssertionError** ne peut se produire qu'en cas d'échec d'une assertion

  →Il ne doit jamais être une surprise ou apparaître d'une façon inattendue

```
try :
    if (x<0) :
        raise ValueError("x est négatif")
    assert x>3
except ValueError :
    print("x est négatif")
except AssertionError :
    print("x doit être >3")
```

x=-3

```
x est négatif
```

x=2

```
x doit être >3
```

Si la condition dans **assert** renvoie **True**, l'exécution se poursuit normalement, sinon, une exception **AssertionError** est levée

Quiz

1. Quelle est la sortie du code suivant ? (une seule réponse)

```
try:
    x=a+2
    raise ValueError("Erreur1")
except Exception as e:
        print(e)
```

- ☐ Erreur1
- ☑ name 'a' is not defined
- ☐ ValueError
- ☐ Aucune réponse n'est correcte

2. Quelle est la sortie du code suivant ? (une seule réponse)

```
try:
    x=1; assert x==2
except AssertionError as e:
        print("x!=2")
```

- ☑ x!=2
- ☐ ValueError
- ☐ Rien va être afficher
- ☐ Aucune réponse n'est correcte

Activité

Écrire une fonction **nb_el()** qui affiche le *nombre d'éléments* d'une **liste *L*** **donnée en paramètre**. Une **Exception** est déclenchée si **aucun argument n'est donnée lors de l'invocation de nb_el()**
Cette Exception renvoie le message "Aucun argument n'est donné"

```
def nb_el(L=[]):
    if L==[]:
        raise Exception("Aucun argument n'est donné")
    else:
        print(len(L))

try:
    nb_el()
except Exception as e:
    print(e)
```


BIBLIOGRAPHIE

[1] **G. Moruzzi**. **Python basics and the interactive mode. Essential Python for the Physicist**. Springer, Cham, **2020**

[2] **S. Nagar**. **Introduction to Python for engineers and scientists**, Berkeley, CA, **2018**

[3] **S. Doty**. **Python Basics**. Computer Science, **2008**

MERCI DE VOTRE ATTENTION

شُكرًا على المُتابَعة

© **Travaux pratiques n°1**			**AU :** 2021-2022
Matière	Programmation Python	**Semestre :** 1	
Discipline	1ère année Génie Informatique		
Enseignants	Dr. Taoufik Ben Abdallah / Dr. Rania Rebaï Boukhriss		
Nombre de pages : 2			

Exécution conditionnelles, boucles & chaines de caractères

Etapes à suivre :

- Télécharger le fichier ***TP1_PP_2021.zip*** à partir de Moodle, puis extraire le fichier dedans ***TP1_PP.ipynb***
- Ouvrir Jupyter notebook et télécharger (upload) le fichier ***TP1_PP.ipynb***, puis cliquer sur Téléverser
- Répondre aux questions de TP dans l'espace réservé de chaque question

Exercice 1

Ecrire un programme Python permettant de :

1. Saisir trois entiers dans les zones identifiées respectivement par n_1, n_2, et n_3. Afficher le contenu de n_1, n_2, et n_3

2. Permuter le contenu de n_1, n_2, et n_3 dans le sens contraire des aiguilles d'une montre. Afficher le contenu de n_1, n_2, et n_3 après permutation

Exercice 2

Ecrire un programme Python permettant de :

1. Saisir deux entiers : l'heure (h) et les minutes (m) avec $h \in [0 \dots 23]$ et $m \in [0 \dots 59]$. Répéter la saisie jusqu'à h et m seront valides

2. Afficher l'heure qu'il sera une minute plus tard

Exemple 1 : étant donné $h = 21$ et $m = 32$, le message affiché est `"Dans une minute, il sera 21:33"`
Exemple 2 : étant donné $h = 14$ et $m = 59$, le message affiché est `"Dans une minute, il sera 15:0"`

Exercice 3

Étant donné la chaine de caractères $ch =$ **" Bonjour Python2021 "**, Écrire un programme Python permettant de :

1. Afficher le nombre de mots

2. Afficher le nombre d'occurrences de la lettre **"o"**

3. Remplacer le mot **"Bonjour"** par **"Bienvenue"**. Afficher ch

4. Supprimer les espaces de bord de ch. Afficher ch

Exercice 4

Écrire un programme Python permettant de vérifier si une chaine de caractères ch donnée est palindrome ou non. Un palindrome est un mot ou une phrase dont l'ordre des lettres reste le même si on le lit de gauche à droite ou de droite à gauche. Afficher `"palindrome"` si ch est palindrome, et `"n'est pas palindrome"` sinon

Exercice 5

Étant donné la chaine de caractères $ch =$ `"Aap193th23zyt889D3on"`, écrire un programme Python permettant de :

1. Remplacer les sous-chaines **composées de trois chiffres successifs** dans ch par le symbole`"@"`. Afficher ch

2. Calculer la somme des chiffres restants qui apparaissent dans ch

3. Convertir les caractères majuscules en des caractères minuscules, et inversement. Afficher ch

4. Vérifier si la chaine M ="python" est composable à partir de ch. Un mot est composable à partir d'une séquence de lettres si la séquence contient toutes les lettres du mot. Chaque lettre de la séquence ne peut être utilisée qu'une seule fois

Exercice 6

Écrire un programme Python permettant de :

1. Saisir une chaine de caractères mp, désignant un mot de passe, de **maximum de 10 caractères** dont au moins un caractère correspond à la lettre `"A"` ou `"a"`, et qui ne comporte pas d'espaces. Afficher `None` si les conditions ne sont pas vérifiées et mp sinon

2. Afficher le nombre de lettres en majuscules dans mp

3. Afficher le nombre de caractères non redondants dans mp

4. *Dans une seule ligne de code*, remplacer les caractères `"A"` ou `"a"` de mp par `"?"`, inverser la chaine, et afficher la nouvelle chaine de caractères obtenue

Exemple : étant donné mp =`"a@A2f1Am4"`, alors le programme affiche `"4m?1f2?@?"`

5. Créer une chaine mp_T qui comporte les caractères de mp décalés de 1 cran dans l'alphabet. Afficher mp_T

Bon Travail ♣

PROGRAMMATION PYTHON

CORRECTION TP1

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

2021-2022

Travaux pratiques n°1 : Exercice2 (1/2)

Ecrire un programme Python permettant de :

1) Saisir deux entiers : l'heure (h) et les minutes (m) avec $h \in [0 \dots 23]$ et $m \in [0 \dots 59]$. Répéter la saisie jusqu'à h et m seront valides

```
while True:
    h=eval(input("h="))
    if h>=0 and h<=23:
        break
```

```
while True:
    m=eval(input("m="))
    if m>=0 and m<=59:
        break
```


Travaux pratiques n°1 : Exercice2 (2/2)

2) Afficher l'heure qu'il sera une minute plus tard

Exemple 1 : étant donné $h = 21$ et $m = 32$, le message affiché est "Dans une minute, il sera 21:33"
Exemple 2 : étant donné $h = 14$ et $m = 59$, le message affiché est "Dans une minute, il sera 15:0"

```
if m==59:
    if h==23:
        h=0
        m=0
    else :
        h+=1
        m=0
else:
    m=m+1
print("{}:{}".format(h,m))
```


Travaux pratiques n°1 : Exercice3

Étant donné la chaine de caractères $ch =$ " Bonjour Python2021 ", Écrire un programme Python permettant de :

1) Afficher le nombre de mots

```
print(ch.strip().count(' ')+1)
```

2) Afficher le nombre d'occurrences de la lettre "o"

```
print("Nombre de o dans ch=", ch.count("o"))
```

3) Remplacer le mot "Bonjour" par "Bienvenue". Afficher ch

```
ch=ch.replace("Bonjour","Bienvenue")
print(ch)
```

4) Supprimer les espaces de bord de ch. Afficher ch

```
ch=ch.strip()
print(ch)
```

Travaux pratiques n°1 : Exercice4

Écrire un programme Python permettant de vérifier si une chaine de caractères ch donnée est palindrome ou non.

Un palindrome est un mot ou une phrase dont l'ordre des lettres reste le même si on le lit de gauche à droite ou de droite à gauche.

Afficher "palindrome" si ch est palindrome, et "n'est pas palindrome" sinon

```
ch=input("ch=")
if ch.upper()==ch[::-1].upper():
    print(ch, "est palindrome")
else:
    print(ch, "n'est pas palindrome")
```


Travaux pratiques n°1 : Exercice5 (1/2)

Étant donné la chaine de caractères ch = "Aap123th23zyt789D3on", écrire un programme Python permettant de :

1) Remplacer les sous-chaines composées de trois chiffres successifs dans ch par le symbole"@". Afficher ch

```
for i in range(len(ch)-2):
    if ch[i:i+3].isdigit():
        ch=ch.replace(ch[i:i+3],"@")
print(ch)
```

2) Calculer la somme des chiffres restants qui apparaissent dans ch

```
s=0
for i in ch:
    if i.isdigit():
        s+= int(i)
print(s)
```


Travaux pratiques n°1 : Exercice5 (2/2)

3) Convertir les caractères majuscules en des caractères minuscules, et inversement. Afficher ch

```
ch=ch.swapcase()
print(ch)
```

4) Vérifier si la chaine M ="python" est composable à partir de ch. Un mot est composable à partir d'une séquence de lettres si la séquence contient toutes les lettres du mot.

Chaque lettre de la séquence ne peut être utilisée qu'une seule fois

```
M="python"
for i in M.lower():
    if  M.count(i)<ch.lower().count(i):
        print(ch, "n'est pas composable de", M)
        break
else:
    print(ch, "est composable de", M)
```


Travaux pratiques n°1 : Exercice6 (1/3)

1) Saisir une chaine de caractères mp, désignant un mot de passe, de maximum de 10 caractères dont au moins un caractère correspond à la lettre "A" ou "a", et qui ne comporte pas d'espaces.

Afficher **None** si les conditions ne sont pas vérifiées et mp sinon

```
mp=input("mp= ")
if (len(mp)>10 or "A" not in mp.upper() or " " in mp):
    print(None)
```

2) Afficher le nombre de lettres en majuscules dans mp

```
count=0
for i in mp:
    if i.isupper():
        count+=1
print(count)
```

Travaux pratiques n°1 : Exercice6 (2/3)

3) Afficher le nombre de caractères non redondants dans mp

```
count=0
for i in mp:
    if mp.count(i)==1:
        count+=1
print(count)
```

4) Dans une seule ligne de code, remplacer les caractères "A" ou "a" de mp par "?", inverser la chaine, et afficher la nouvelle chaine de caractères obtenue

Exemple : étant donné mp ="a@A2f1Am4", alors le programme affiche "4m?1f2?@?"

```
print(mp.replace("A","?").replace("a","?")[::-1])
```


Travaux pratiques n°1 : Exercice6 (3/3)

5) Créer une chaine mp_T qui comporte les caractères de mp décalés de 1 cran dans l'alphabet. Afficher mp_T

```
mp_T=""
for i in mp:
    mp_T+=chr(ord(i)+1)
print(mp_T)
```

© **Travaux pratiques n°2**			**AU :** 2021-2022
Matière	Programmation Python	**Semestre :** 1	
Discipline	1ère année Génie Informatique		
Enseignants	Dr. Taoufik Ben Abdallah / Dr. Rania Rebaï Boukhriss		
Nombre de pages : 2			

Listes & Tuples

Etapes à suivre :

- Télécharger le fichier ***TP2_PP_2021.zip*** à partir de Moodle, puis extraire le fichier dedans ***TP2_PP.ipynb***
- Ouvrir Jupyter notebook et télécharger (upload) le fichier ***TP2_PP.ipynb***, puis cliquer sur Téléverser
- Répondre aux questions de TP dans l'espace réservé de chaque question

Exercice 1

Étant donné une liste $L = [20, 15, 8, 6, 9, 4]$;

1. Renverser ***L*** puis l'afficher

2. Calculer et afficher la somme des éléments de ***L***

3. Calculer et afficher le nombre des éléments de ***L***

4. Calculer et afficher le minimum, et la moyenne des éléments de ***L***

5. Tester l'appartenance de la valeur de la moyenne parmi les éléments de ***L***

6. Ajouter à la fois les éléments **1**, **4**, et **8** à la fin de la liste ***L***. Afficher ***L***

7. Supprimer les éléments qui se répètent plus qu'une fois dans la liste ***L***. Afficher ***L***

8. Ecrire une liste en compréhension ***L1*** qui contient les éléments d'indices pairs de ***L***. Afficher ***L1***

9. Trier en ordre décroissant les éléments de ***L1***. Afficher ***L1***

Exercice 2

Étant donnée ***ch1*** =`"abc"`, et ***ch2*** =`"de"`, écrire une seule instruction python qui permet d'extraire à partir de ces deux chaînes une liste contenant toutes les combinaisons possibles constitué de deux caractères : un caractère de ***ch1***, et l'autre de ***ch2***. Le résultat à afficher est `['ad', 'ae', 'bd', 'be', 'cd', 'ce']`

Exercice 3

Ecrire un programme qui permet de saisir deux entiers m et n, et de générer une liste ***L_a*** de $m \times n$ **nombres binaires aléatoires** (**0** ou **1**). Transformer la liste ***L_a*** en une liste de m listes de n éléments

Exemple : si $m = 3, n = 2$, et ***L_a*** $= [0, 1, 0, 0, 1, 1]$ alors le programme doit afficher $[[0, 1], [0, 0], [1, 1]]$

Exercice 4

Étant donné la chaine de caractères ***ch*** suivante :

```
ch ="""Python est est est populaire
Python populaire est simple"""
```

1. Créer une liste, nommée ***L_ch***, contenant des listes dont chacun comporte **deux éléments** : le premier représente un mot de ***ch***, et le deuxième correspond à son nombre d'occurrence. Le résultat à afficher est ***L_ch*** = `[['Python', 2], ['est', 4], ['populaire', 2], ['simple', 1]]`

2. Mélanger aléatoirement les éléments de ***L_ch***

Exemple : ***L_ch*** = `[['Python', 2], ['populaire', 2], ['est', 4], ['simple', 1]]`

3. Transformer les listes (de format **[*mot* , *val*]**) de ***L_ch*** en des listes contenant le mot ***mot*** répété ***val*** fois

Exemple : étant donné ***L_ch*** = `[['Python', 2], ['populaire', 2], ['est', 4], ['simple', 1]]` ; alors le programme affiche ***L_ch*** = `[['Python', 'Python'], ['populaire', 'populaire'], ['est', 'est', 'est', 'est'], ['simple']]`

4. Transformer ***L_ch*** en une chaine de caractères, nommée ***ch1*** dont les mots sont séparés par **"*"**

Exemple : étant donné ***L_ch*** = `[['Python', 'Python'], ['populaire', 'populaire'], ['est', 'est', 'est', 'est'], ['simple']]` ; alors le programme affiche ***ch1*** = `"Python*Python*populaire*populaire*est*est*est*est*simple"`

Exercice 5

1. Saisir un entier positif $N \leq 20$, puis générer aléatoirement un tuple, nommé ***Tup_chiffres***, de ***N*** entiers qui varient entre 1 et 10. **Répéter la saisie de *N* si la valeur donnée est erronée**

2. Extraire à partir de ***Tup_chiffres*** un tuple, nommé ***Tup_seg***, qui contient ***M*** listes dont chacun comporte ***K*** chiffres ordonnées dans l'ordre décroissant

Exemple : étant donné ***Tup_chiffres*** = $(2, 2, 1, 3, 4, 6, 1)$; alors le programme affiche ***Tup_seg*** = $([2, 2, 1], [3], [4], [6, 1])$

3. Ajouter à la première position de chaque liste de ***Tup_seg*** une valeur représentant la somme de ses éléments

Exemple : étant donné ***Tup_seg*** = $([2, 2, 1], [3], [4], [6, 1])$; alors le programme affiche ***Tup_seg*** = $([5, 2, 2, 1], [3, 3], [4, 4], [7, 6, 1])$

4. Transformer ***Tup_seg*** en une liste, nommée ***L_nb***

Exemple : étant donné ***Tup_seg*** = $([5, 2, 2, 1], [3, 3], [4, 4], [7, 6, 1])$; alors le programme affiche ***L_nb*** = $[5, 2, 2, 1, 3, 3, 4, 4, 7, 6, 1]$

5. Ecrire **dans une seule instruction** un programme qui permet de convertir les valeurs non dupliqués en une chaine de caractère, nommée ***ch_nb***

Exemple : étant donné ***L_nb*** = $[5, 2, 2, 1, 3, 3, 4, 4, 7, 6, 1]$; alors le programme affiche ***ch_nb*** = "576"

Bon Travail ♣

IIT INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine privée

PROGRAMMATION PYTHON

CORRECTION TP2

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

2021-2022

Travaux pratiques n°2 : Exercice1 (1/2)

Étant donné une liste $L = [20, 15, 8, 6, 9, 4]$;

1. Renverser L puis l'afficher

```
L.reverse() ; print(L)
```

2. Calculer et afficher la somme des éléments de L

```
print(sum(L))
```

3. Calculer et afficher le nombre des éléments de L

```
print(len(L))
```

4. Calculer et afficher le minimum, et la moyenne des éléments de L

```
print(min(L))
```

```
from statistics import mean
m=mean(L); print(m)
```

```
#ou bien
m=sum(L)/len(L); print(m)
```

5. Tester l'appartenance de la valeur de la moyenne parmi les éléments de L

```
m in L
```


Travaux pratiques n°2 : Exercice1 (1/2)

6. Ajouter à la fois les éléments 1, 4, et 8 à la fin de la liste L. Afficher L

```
L.extend([1,4,8])
print(L)
```

7. Supprimer les éléments qui se répètent plus qu'une fois dans la liste L. Afficher L

```
for el in L:
    nb=L.count(el)
    if nb>1:
        for i in range(nb):
            L.remove(el)
print(L)
```

8. Ecrire une liste en compréhension $L1$ qui contient les éléments d'indices pairs de L. Afficher L

```
L1=[L[i] for i in range (0,len(L),2)] ; print(L1)
```

9. Trier en ordre décroissant les éléments de $L1$. Afficher $L1$

```
L1.sort(reverse=True) ; print(L1)
```


Travaux pratiques n°2 : Exercice2

Étant donnée $ch1$ ="abc", et $ch2$ ="de", écrire **une seule instruction** python qui permet d'extraire à partir de ces deux chaînes une liste contenant toutes les combinaisons possibles constitué de deux caractères : un caractère de $ch1$, et l'autre de $ch2$. Le résultat à afficher est `['ad', 'ae', 'bd', 'be', 'cd', 'ce']`

```
ch1="abc"; ch2="de"

print([c1+c2 for c1 in ch1 for c2 in ch2] )
```

Travaux pratiques n°2 : Exercice3

Ecrire un programme qui permet de saisir deux entiers m et n, et de générer une liste L_a de $m \times n$ nombres binaires aléatoires (0 ou 1)

```
m=int(input("m="))
n=int(input("n="))

from random import randint
L_a=[randint(0,1) for i in range(m*n)] ; print(L_a)

L_a=[ L_a[j:j+n] for j in range(0,len(L_a),n)]
print(L_a)
```


Travaux pratiques n°2 : Exercice4 (1/3)

Étant donné la chaine de caractères ch suivante :
ch ="""Python est est est populaire
Python populaire est simple"""

1. Créer une liste, nommée L_ch, contenant des listes dont chacun comporte **deux éléments** : le premier représente un mot de ch, et le deuxième correspond à son nombre d'occurrence. Le résultat à afficher est L_ch = [['Python', 2], ['est', 4], ['populaire', 2], ['simple', 1]]

```
L=ch.strip().split()
L_ch=[[L[i],L.count(L[i])] for i in range(len(L))
                          if L[i] not in L[:i]]

print(L_ch)
```


Travaux pratiques n°2 : Exercice4 (2/3)

2. Mélanger aléatoirement les éléments de L_ch

Exemple : L_ch = [['Python', 2], ['populaire', 2], ['est', 4], ['simple', 1]]

```
from random import shuffle
shuffle(L_ch)
print(L_ch)
```

3. Transformer les listes (de format [mot , val]) de L_ch en des listes contenant le mot mot répété val fois

Exemple :
étant donné L_ch = [['Python', 2], ['populaire', 2], ['est', 4], ['simple', 1]] ;
alors le programme affiche
L_ch = [['Python', 'Python'], ['populaire', 'populaire'], ['est', 'est', 'est', 'est'], ['simple']]

```
L_ch=[ [el[0] for compt in range(el[1])]  for el in L_ch]
print(L_ch)
```


Travaux pratiques n°2 : Exercice4 (3/3)

4. Transformer L_ch en une chaine de caractères, nommée $ch1$ dont les mots sont séparés par "*"

Exemple : étant donné
L_ch = [['Python', 'Python'], ['populaire', 'populaire'], ['est', 'est', 'est', 'est'], ['simple']] ; alors le programme affiche
$ch1$ = "Python*Python*populaire*populaire*est*est*est*est*simple"

```
print("*".join(["*".join(el) for el in L_ch]))
```

Travaux pratiques n°2 : Exercice5 (1/4)

1. Saisir un entier positif $N \leq 20$, puis générer aléatoirement un tuple, nommé $Tup\_chiffres$, de N entiers qui varient entre 1 et 10. Répéter la saisie de N si la valeur donnée est erronée

```
from random import randint
while True:
    N=int(input("N="))
    if N in range(0,21):
        break

Tup_chiffres=tuple([randint(0,10) for i in range(N)])
```


Travaux pratiques n°2 : Exercice5 (2/4)

2. Extraire à partir de $Tup\_chiffres$ un tuple, nommé Tup_seg, qui contient M listes dont chacun comporte K chiffres ordonnées dans l'ordre décroissant

Exemple : étant donné $Tup\_chiffres = (2,2,1,3,4,6,1)$; alors le programme affiche $Tup\_seg = ([2,2,1],[3],[4],[6,1])$

```
#Tup_chiffres=(2,2,1,3,4,6,1)

j=0
el_seg=[]
for i in range(len(Tup_chiffres)):
    if(i==len(Tup_chiffres)-1 or Tup_chiffres[i]<Tup_chiffres[i+1]) :
        el_seg.append(list(Tup_chiffres[j:i+1]))
        j=i+1
Tup_seg=tuple(el_seg)
print(Tup_seg)
```


Travaux pratiques n°2 : Exercice5 (3/4)

3. Ajouter à la première position de chaque liste de Tup_seg une valeur représentant la somme de ses éléments

Exemple : étant donné $Tup\_seg = ([2,2,1],[3],[4],[6,1])$; alors le programme affiche $Tup\_seg = ([5,2,2,1],[3,3],[4,4],[7,6,1])$

```
#Tup_seg=([2, 2, 1], [3], [4], [6, 1])

for i in range(len(Tup_seg)):
    Tup_seg[i].insert(0,sum(Tup_seg[i]))
print(Tup_seg)
```


Travaux pratiques n°2 : Exercice5 (4/4)

4. Transformer Tup_seg en une liste, nommée L_nb

Exemple : étant donné $Tup\_seg = ([5,2,2,1],[3,3],[4,4],[7,6,1])$; alors le programme affiche $L\_nb = [5,2,2,1,3,3,4,4,7,6,1]$

```
L_nb=[j  for el in Tup_seg for j in el]
```

5. Ecrire dans une seule instruction un programme qui permet de convertir les valeurs non dupliqués en une chaine de caractère, nommée ch_nb

Exemple : étant donné $L\_nb = [5,2,2,1,3,3,4,4,7,6,1]$; alors le programme affiche $ch\_nb =$ "576"

```
#L_nb=[5, 2, 2, 1, 3, 3, 4, 4, 7, 6, 1]

ch_nb="".join([str(i) for i in L_nb if L_nb.count(i)==1])
print(ch_nb)
```

© **Travaux pratiques n°3**			**AU :** 2021-2022
Matière	Programmation Python	**Semestre :** 1	
Discipline	1ère année Génie Informatique		
Enseignants	Dr. Taoufik Ben Abdallah / Dr. Rania Rebaï Boukhriss		
Nombre de pages : 2			

Ensembles & Dictionnaires

Etapes à suivre :

- Télécharger le fichier ***TP3_PP_2021.zip*** à partir de Moodle, puis extraire le fichier dedans ***TP3_PP.ipynb***
- Ouvrir Jupyter notebook et télécharger (upload) le fichier ***TP3_PP.ipynb***, puis cliquer sur Téléverser
- Répondre aux questions de TP dans l'espace réservé de chaque question

Exercice 1

Étant donné deux chaines de caractères $ch1$ = `'abc'` et $ch2$ = `'dc'`;

1. Créer l'ensemble $X1$ qui comporte les caractères de $ch1$ et $ch2$; Le résultat à afficher est $X1$ = {`'d','c','b','a'`}

2. Créer l'ensemble $X2$ qui comporte la combinaison des chaines de deux caractères en éliminant toutes redondances (par exemple `'ba'` et `'ab'` sont redondantes) ; Le résultat à afficher est $X2$ = `{'ac', 'ab', 'cd', 'ad', 'cb', 'bd'}`

3. Créer l'ensemble $X3$ qui comporte uniquement la combinaison des chaines de deux caractères qui supportent `'a'` ; Le résultat à afficher est $X3$ =`{'ad', 'ab', 'ac'}`

4. Créer l'ensemble $X4$ qui comporte uniquement la combinaison des chaines de deux caractères **qui ne supportent pas** `'a'` ; Le résultat à afficher est $X4$ =`{'bd', 'cb', 'cd'}`

Exercice 2

Étant donné un dictionnaire d =`{'prenom1' : 'Tawfik', 'nom1' : 'Ben Abdallah', 'age' : 31}`;

1. Créer l'instruction qui permet de corriger **la valeur de clé** `'prenom1'` de `'Tawfik'` à `'Taoufik'`. Afficher d

2. Afficher la liste des clés de d

3. Afficher la liste des valeurs de d

4. Créer la liste $L1$ des paires clé/valeur du dictionnaire correspondant à cette format `[[clé1,valeur1], [clé2,valeur2], ...]`. Afficher $L1$

Exercice 3

Étant donné la chaine de caractère seq = `"ACCTAGCCCTA"`, écrire un programme qui affiche un dictionnaire d_s contenant le nombre d'occurrence de toutes les combinaisons possibles des mots de deux lettres successives qui existent dans seq. Le résultat à afficher est `{'AC' : 1, 'CC' : 3, 'CT' : 2, 'TA' : 2, 'AG' : 1, 'GC' : 1}`

Exercice 4

Étant donné l'ensemble $jardin$ est de format `{('id_jardin1', 'proprietaire_jardin1'), ('id_jardin2', 'proprietaire_jardin2'),...}`, et la liste $entretien$ est de format `[['id_jardin1', 'date_entretien1_jardin1', coût_entretien1_jardin1], ['id_jardin1', 'date_entretien2_jardin1', coût_entretien2_jardin1],..., ['id_jardin2', 'date_entretien1_jardin2', coût_entretien1_jardin2],...]` ; **Chaque jardin peut avoir un ou plusieurs entretien(s)**.

Soit $jardin$=`{('j001', 'AA'), ('j002', 'DD')}`, et $entretien$=`[['j001', '12/06/2021', 100], ['j001', '22/09/2021', 211], ['j002', '27/05/2021', 65]]` ;

À partir de $jardin$ et $entretien$, créer **dans une seule instruction** un dictionnaire, nommé $jardin\_entretien$ de ce format `{'id_jardin1' :[('date_entretien1_jardin1', coût_entretien1_jardin1), ('date_entretien2_jardin1', coût_entretien2_jardin1)], 'id_jardin2' :[('date_entretien1_jardin1', coût_entretien1_jardin1)],... }`.

Le résultat à afficher est {`'j001' :[('12/06/2021', 100), ('22/09/2021', 211)], 'j002' :[('27/05/2021', 65)]`

Exercice 5

Étant donné un annuaire téléphonique qui est présenté sous forme d'un dictionnaire nommé d_a avec

$d\_a =$

```
{'AAA': '74 445 000',
'BBB': ['74 067 129', '74 067 145'],
'EEE': '71 991 883',
'CCC': ['71 179 002', '71 178 001', '71 179 110'] }
```

Les clés représentent les noms des personnes dans l'annuaire

Les valeurs représentent une chaine de caractère ou une liste de chaines correspondant à un ou plusieurs numéros de téléphones

1. Modifier le numéro de téléphone de `'BBB'` par `'74 999 999'`. Afficher d_a

2. Affecter le numéro de téléphone `'71 179 145'` à `'EEE'`. Afficher d_a

3. À partir de d_a, supprimer la personne `'BBB'`. Afficher d_a

4. À partir de d_a, créer, **dans une seule instruction**, une liste L_n qui contient les noms de personnes. Afficher L_n

5. À partir de d_a, créer, **dans une seule instruction**, un **ensemble de tuples** ens_t de format `{(Nom de la personne1, Numéro de téléphone1, Numéro de téléphone2), (Nom de la personne2, Numéro de téléphone1,…)}`. Afficher ens_t

6. Vérifier si l'ensemble `{'AAA', '74 445 000'}` est un sous ensemble de ens_t. **Afficher True si c'est le cas et False sinon**

7. Vider d_a et ens_t puis les afficher

Bon Travail ♣

PROGRAMMATION PYTHON

CORRECTION TP3

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

2021-2022

Travaux pratiques n°3 : Exercice1 (1/1)

$ch1$ = 'abc' ; $ch2$ = 'dc'

1. Créer l'ensemble $X1$ qui comporte les caractères de $ch1$ et $ch2$

```
X1=set(ch1)|set(ch2)
print(X1)
```

2. Créer l'ensemble $X2$ qui comporte la combinaison des chaines de deux caractères en éliminant toutes redondances (par exemple 'ba' et 'ab' sont redondantes)

```
l=list(X1)
X2={i+j for i in X1 for j in l[l.index(i)+1:]}
print(X2)
```

3. Créer l'ensemble $X3$ qui comporte uniquement la combinaison des chaines de deux caractères qui supportent 'a' ; Le résultat à afficher est $X3$ ={'ad', 'ab', 'ac'}

```
X3={i for i in X2 if 'a' in i} ; print(X3)
```

4. Créer l'ensemble $X4$ qui comporte uniquement la combinaison des chaines de deux caractères qui ne supportent pas 'a'

```
X4=X2-X3 ; print(X4)
```


Travaux pratiques n°3 : Exercice2 (1/1)

d ={'prenom1' : 'Tawfik', 'nom1' : 'Ben Abdallah', 'age' : 31}

1. Créer l'instruction qui permet de corriger la valeur de clé 'prenom1' de 'Tawfik' à 'Taoufik'. Afficher d

```
d['prenom1']="Taoufik"
print(d)
```

2. Afficher la liste des clés de d

```
print(list(d.keys()))
```

3. Afficher la liste des valeurs de d

```
print(list(d.values()))
```

4. Créer la liste $L1$ des paires clé/valeur du dictionnaire correspondant à cette format [[clé1,valeur1], [clé2,valeur2], ...]. Afficher $L1$

```
L1=[list(i) for i in d.items()]
print(L1)
```


Travaux pratiques n°3 : Exercice3 (1/1)

seq = "ACCTAGCCCTA"

Ecrire un programme qui affiche un dictionnaire d_s contenant le nombre d'occurrence de toutes les combinaisons possibles des mots de deux lettres successives qui existent dans seq

Le résultat à afficher est **{'AC' : 1, 'CC' : 3, 'CT' : 2, 'TA' : 2, 'AG' : 1, 'GC' : 1}**

```
l=[seq[i:i+2] for i in range(len(seq)-1)]
d_s={i:l.count(i) for i in l}
print(d_s)
```

Travaux pratiques n°3 : Exercice4 (1/1)

$jardin$ ={('id_jardin1', 'proprietaire_jardin1'), ('id_jardin2', 'proprietaire_jardin2'),...}
$entretien$=[['id_jardin1', 'date_entretien1_jardin1', coût_entretien1_jardin1], ['id_jardin1', 'date_entretien2_jardin1', coût_entretien2_jardin1],..., ['id_jardin2', 'date_entretien1_jardin2', coût_entretien1_jardin2],...]
Chaque jardin peut avoir un ou plusieurs entretien(s).

$jardin$={('j001', 'AA'), ('j002', 'DD')}
$entretien$=[['j001', '12/06/2021', 100], ['j001', '22/09/2021', 211], ['j002', '27/05/2021', 65]]

À partir de $jardin$ et $entretien$, créer dans une seule instruction un dictionnaire, nommé $jardin\_entretien$ de ce format

{'id_jardin1' :[('date_entretien1_jardin1', coût_entretien1_jardin1), ('date_entretien2_jardin1', coût_entretien2_jardin1)], 'id_jardin2' :[('date_entretien1_jardin1', coût_entretien1_jardin1)],... }

Le résultat à afficher est {'j001' :[('12/06/2021', 100), ('22/09/2021', 211)], 'j002' :[('27/05/2021', 65)]

```
jardin=list(jardin)
jardin.sort()
jardin_entretien={c[0]:[(i[1],i[2])
                        for i in entretien if i[0]==c[0]] for c in jardin}
print(jardin_entretien)
```


Travaux pratiques n°3 : Exercice5 (1/3)

```
d_a ={'AAA': '74 445 000',
'BBB': ['74 067 129', '74 067 145'],
'EEE': '71 991 883',
'CCC': ['71 179 002', '71 178 001', '71 179 110'] }
```

1. Modifier le numéro de téléphone de 'BBB' par '74 999 999'. Afficher d_a

```
d_a['BBB']='74 999 999'
print(d_a)
```

2. Affecter le numéro de téléphone '71 179 145' à 'EEE'. Afficher d_a

```
if type(d_a["EEE"])==list:
    d_a["EEE"]+=['71 179 145']
else:
    d_a["EEE"]=[d_a["EEE"], '71 179 145']

print(d_a)
```


Travaux pratiques n°3 : Exercice5 (2/3)

3. À partir de d_a, supprimer la personne 'BBB'. Afficher d_a

```
del d_a['BBB']
print(d_a)
```

4. À partir de d_a, créer, dans une seule instruction, une liste L_n qui contient les noms de personnes. Afficher L_n

```
L_n=list(d_a) # L_n=list(d_a.keys())
print(L_n)
```

5. À partir de d_a, créer, dans une seule instruction, un ensemble de tuples ens_t de format {(Nom de la personne1, Numéro de téléphone1, Numéro de téléphone2), (Nom de la personne2, Numéro de téléphone1,⋯)}. Afficher ens_t

```
ens_t={(cle,val) if type(val)!=list else (cle,)+tuple(val)
       for cle,val in d_a.items()}
print(ens_t)
```


Travaux pratiques n°3 : Exercice5 (3/3)

6. Vérifier si l'ensemble {'AAA', '74 445 000'} est un sous ensemble de ens_t. Afficher **True** si c'est le cas et **False** sinon

```
print({'AAA', '74 445 000'}.issubset(ens_t))

# ou bien # print({'AAA', '74 445 000'}<=(ens_t))
```

7. Vider d_a et ens_t puis les afficher

```
d_a.clear()
ens_t.clear()
print(d_a)
print(ens_t)
```

© **Travaux pratiques n°4**			**AU :** 2021-2022
Matière	Programmation Python	**Semestre :** 1	
Discipline	1ère année Génie Informatique		
Enseignants	Dr. Taoufik Ben Abdallah / Dr. Rania Rebaï Boukhriss		
Nombre de pages : 2			

Fonctions & Générateurs

Etapes à suivre :

- Télécharger le fichier ***TP4_PP_2021.zip*** à partir de Moodle, puis extraire le fichier dedans ***TP4_PP.ipynb***
- Ouvrir Jupyter notebook et télécharger (upload) le fichier ***TP4_PP.ipynb***, puis cliquer sur Téléverser
- Répondre aux questions de TP dans l'espace réservé de chaque question

Exercice 1

Soit la base des étudiants, stockée dans une liste de dictionnaires, nommée ***etudiants***, et définie dans le programme principal comme suit

etudiants =[

```
{'mat' :'E019741', 'nom' : 'AAA', 'pren' :'BBB', 'date_n': '12/08/1997', 'classe': '1prepa'},
{'mat' :'E0A7891', 'nom' : 'CCC', 'pren' :'DDD', 'date_n': '01/01/1996', 'classe': '1GI'},
{'mat' :'E119841', 'nom' : 'EEE', 'pren' :'FFF', 'date_n': '11/02/1998', 'classe': '1prepa'},
{'mat' :'E765123', 'nom' : 'EEE', 'pren' :'FFF', 'date_n': '18/09/1997', 'classe': '1GI'},
{'mat' :'E111354', 'nom' : 'GGG', 'pren' :'HHH', 'date_n': '06/10/1995', 'classe': '2GI'}]
```

Un étudiant est caractérisé par son matricule (`mat`), son nom, son prénom (`pren`), sa date de naissance (`date_n`), et la classe dans laquelle il est inscrit.

On peut se baser sur le programme principal suivant pour appeler les fonctions à développer :

```
L_e= transformer()
ajouter(1, mat="E76A123", nom="VVV", pren="ZZZ", date_n="11/12/1996, classe="1GI")
determiner_age()
trier()
print(L_e)
print(texte())
```

Travail à faire :

1. Écrire un **générateur** ***gen_etudiants***(***etudiants***) qui permet de renvoyer tous les listes possibles correspondants à un étudiants de format `[mat, nom, pren, date_n, classe]`

2. Écrire une fonction ***transformer***() qui transforme le dictionnaire ***etudiants*** en une liste de listes de ce format `[['mat de l'étudiant 1', 'nom de l'étudiant 1', 'pren de l'étudiant 1', 'date_n de l'étudiant 1', 'classe de l'étudiant 1'], ['mat de l'étudiant 2', 'nom de l'étudiant 2', 'pren de l'étudiant 2', 'date_n de l'étudiant 2', 'classe de l'étudiant 2'],…]` (utiliser le générateur ***gen_etudiants***(***etudiants***))

3. Écrire une fonction ***ajouter***(***pos***,** ***e***) qui ajoute dans la position pos de ***L_e*** l'étudiant décrit par ***e***

4. Écrire une fonction ***determiner age***(***ann c* = 2020**) qui ajoute à la dernière position de chaque liste de ***L_e*** l'âge de l'étudiant

5. Écrire une fonction ***trier***() qui permet de trier dans l'ordre décroissant les listes de ***L_e*** selon l'âge de chaque étudiant

6. Écrire une fonction ***texte***() qui retourne une chaine de caractère contenant le nom et le prénom de chaque étudiant de ***L_e***, sépare par `";"`. **Le traitement de cette fonction doit être effectué sans faire recours à une structure itérative**

Exercice 2

Soit la base des ***stock_produits***, décrit par la liste $prod$ qui est définie dans le programme principal comme suit

$prod$ =[

```
({'ref':'P001'}, {'Des':'AXRTF'}, {'qte_stock':22}, {'prix_unitaire': 0.9}),
({'ref':'P002'}, {'Des':'FRCDR'}, {'qte_stock':6}, {'prix_unitaire': 10}),
({'ref':'P003'}, {'Des':'TTRUI'}, {'qte_stock':13}, {'prix_unitaire': 1.1}),
({'ref':'P004'}, {'Des':'MLITD'}, {'qte_stock':8}, {'prix_unitaire': 4.6}),
({'ref':'P005'}, {'Des':'CAAEP'}, {'qte_stock':0}, {'prix_unitaire': 11})]
```

Un produit est caractérisé par sa référence (`ref`), sa description (`Des`), la quantité en stock restante (`qte_stock`) et le prix d'une pièce (`prix_unitaire`).

On peut se baser sur le programme principal suivant pour appeler les fonctions à développer :

```
g=gen_produits()
print(next(g)) #['P001', 'AXRTF', 22, 0.9]
g1=gen_prix()
print(next(g1)) #['P001', 'AXRTF', 19.8]
prod_T=transformer_produits()
print(existe_produit('P003')) #True
print(existe_produit('P006')) #False
print(f(*['ref','Des','qte_stock','prix_unitaire']))
```

Travail à faire :

1. Écrire un générateur $gen\_produits$() qui permet de renvoyer toutes les listes possibles correspondantes à un produit de format `[ref, Des, qte_stock, prix_unitaire]`

2. Écrire un générateur gen_prix () qui calcule **le prix global** de vente de tout le stock de chaque produit de $prod$, et renvoie toutes les listes possibles correspondantes à un produit de format `[ref, Des, prix_global]`

3. Écrire une fonction $transformer\_produits$() qui transforme $prod$ en un tuple de listes de ce format : `([ref_produit1, Des_produit1, prix_unitaire_produit1, prix_global_produit1], [ref_produit2, Des_produit2, prix_unitaire_produit2, prix_global_produit2], …)` (Utiliser le générateur $gen\_produits$)

4. Écrire une fonction $existe\_produit(ref)$ qui vérifie l'existence d'un produit ayant comme référence ref dans $prod\_T$. Cette fonction retourne **l'index de la liste contenant le produit** dans $prod\_T$ s'il existe et **-1** sinon

5. Écrire une fonction $transaction$ (**type** = "**I**",** p_stock) qui permet de mettre à jour le stock de produit p_stock selon le type : **type** = "**I**" (par défaut) désigne une transaction d'entrée dans le stock ou **type** = "**O**" désigne une transaction de sortie dans le stock

6. Écrire une **fonction anonyme** qui transforme $prod\_T$ en une liste de même format que celle de $prod$

Bon Travail ♣

IIT INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine privée

PROGRAMMATION PYTHON

CORRECTION TP4

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

2021-2022

Travaux pratiques n°4 : Exercice1 (1/4)

```
etudiants=[
  {'mat':'E019741', 'nom': 'AAA', 'pren':'BBB', 'date_n': '12/08/1997','classe': '1prepa'},
  {'mat':'10A7891', 'nom': 'CCC', 'pren':'DDD', 'date_n': '01/01/1996','classe': '1GI'},
  {'mat':'E119841', 'nom': 'EEE', 'pren':'FFF', 'date_n': '11/02/1998','classe': '1prepa'},
  {'mat':'E765123', 'nom': 'EEE', 'pren':'FFF', 'date_n': '18/09/1997','classe': '1GI'},
  {'mat':'E111354', 'nom': 'GGG', 'pren':'HHH', 'date_n': '06/10/1995','classe': '2GI'}
]
```

1. Écrire un **générateur** $gen\_etudiants(etudiants)$ qui permet de renvoyer tous les listes possibles correspondants à un étudiants de format `[mat, nom, pren, date_n, classe]`

```
def gen_etudiants(etudiants):
    yield from [list(e.values()) for e in etudiants]
```

```
next(gen_etudiants(etudiants))
```


Travaux pratiques n°4 : Exercice1 (2/4)

2. Écrire une fonction $transformer()$ qui transforme le dictionnaire $etudiants$ en une liste de listes de ce format `[['mat de l'étudiant 1', 'nom de l'étudiant 1', 'pren de l'étudiant 1', 'date_n de l'étudiant 1', 'classe de l'étudiant 1'], ['mat de l'étudiant 2', 'nom de l'étudiant 2', 'pren de l'étudiant 2', 'date_n de l'étudiant 2', 'classe de l'étudiant 2'],…]`
(utiliser le générateur $gen\_etudiants(etudiants)$)

```
def transformer():
    return [j for j in gen_etudiants(etudiants)]
```

```
L_e=transformer()
L_e
```

3. Écrire une fonction $ajouter(pos, ** e)$ qui ajoute dans la position pos de L_e l'étudiant décrit par e

```
def ajouter(pos, **e):
    L_e.insert(pos,list(e.values()))
```

```
ajouter(1, mat="E76A123", nom="VVV", pren="ZZZ", date_n="11/12/1996",
classe="1GI")
L_e
```


Travaux pratiques n°4 : Exercice1 (3/4)

4. Écrire une fonction $determiner\ age(ann\ c = 2020)$ qui ajoute à la dernière position de chaque liste de L_e l'âge de l'étudiant

```
def determiner_age(ann_c=2020):
    for el in L_e:
        el.append(ann_c-int(el[3].split('/')[2]))
```

```
determiner_age()
L_e
```

5. Écrire une fonction $trier()$ qui permet de trier dans l'ordre décroissant les listes de L_e selon l'âge de chaque étudiant

```
def trier():
    L_e.sort(key=lambda l:l[-1],reverse=True)
```

```
trier()
L_e
```

Travaux pratiques n°4 : Exercice1 (4/4)

6. Écrire une fonction ***texte***() qui retourne une chaine de caractère contenant le nom et le prénom de chaque étudiant de ***L_e***, sépare par ";". **Le traitement de cette fonction doit être effectué sans faire recours à une structure itérative**

```
def texte():
    return ";".join(list(map(lambda e: " ".join(e[1:3]),L_e)))
```

```
texte()
```


Travaux pratiques n°4 : Exercice2 (1/5)

1. Écrire un générateur ***gen_produits***() qui permet de renvoyer toutes les listes possibles correspondantes à un produit de format `[ref, Des, qte_stock, prix_unitaire]`

```
def gen_produits():
    yield from [[list(j.values())[0] for j in i] for i in prod]
```

```
g=gen_produits()
print(next(g)) # ['P001', 'AXRTF', 22, 0.9]
```

2. Écrire un générateur ***gen_prix*** () qui calcule **le prix global** de vente de tout le stock de chaque produit de ***prod***, et renvoie toutes les listes possibles correspondantes à un produit de format `[ref, Des, prix_global]`

```
def gen_prix():
    yield from [i[:2]+[i[-1]*i[-2]] for i in gen_produits()]
```

```
g1=gen_prix()
print(next(g1)) # ['P001', 'AXRTF', 19.8]
```


Travaux pratiques n°4 : Exercice2 (2/5)

3. Écrire une fonction ***transformer_produits***() qui transforme ***prod*** en un tuple de listes de ce format : `([ref_produit1, Des_produit1, prix_unitaire_produit1, prix_global_produit1], [ref_produit2, Des_produit2, prix_unitaire_produit2, prix_global_produit2], …)` (Utiliser le générateur ***gen_produits***)

```
def transformer_produits():
    g1=gen_prix()
    return [p+[next(g1)[-1]] for p in gen_produits()]
```

```
prod_T=transformer_produits()
prod_T
```


Travaux pratiques n°4 : Exercice2 (3/5)

4. Écrire une fonction ***existe_produit***(***ref***) qui vérifie l'existence d'un produit ayant comme référence ***ref*** dans ***prod_T***. Cette fonction retourne **l'index de la liste contenant le produit** dans ***prod_T*** s'il existe et **-1** sinon

```
def existe_produit(ref):
    for i in range(len(prod_T)):
        if ref==prod_T[i][0]:
            return i
    else: return -1
```

```
print(existe_produit('P003')) #2
print(existe_produit('P006')) #-1
```

Travaux pratiques n°4 : Exercice2 (4/5)

5. Écrire une fonction $transaction$ ($\mathbf{type} = \text{"I"}$,** p_stock) qui permet de mettre à jour le stock de produit p_stock selon le type : $\mathbf{type} = \text{"I"}$ (par défaut) désigne une transaction d'entrée dans le stock ou $\mathbf{type} = \text{"O"}$ désigne une transaction de sortie dans le stock

```
def transaction(type="I", **p_stock):
    l=list(p_stock.values())
    i=existe_produit(l[0])
    if i!=-1:
        if type=="I":
            prod_T[i][-3]+=l[1]
        else: prod_T[i][-3]-=l[1]
        prod_T[i][-1]=prod_T[i][-3]* prod_T[i][-2]
```

```
transaction(ref="P001", qte=10)
transaction(type="O", ref="P003", qte=3)

prod_T
```


Travaux pratiques n°4 : Exercice2 (4/5)

6. Écrire une **fonction anonyme** qui transforme $prod\_T$ en une liste de même format que celle de $prod$

```
f= lambda *keys: [tuple({keys[i]: el[i]} for i in range(len(keys)))
                                for el in prod_T]
```

```
print(f(*['ref','Des','qte_stock','prix_unitaire']))
```

© **Travaux pratiques n°5**			**AU :** 2021-2022
Matière	Programmation Python	**Semestre :** 1	
Discipline	1ère année Génie Informatique		
Enseignants	Dr. Taoufik Ben Abdallah / Dr. Rania Rebaï Boukhriss		
Nombre de pages : 1			

Gestion de Fichiers & Exceptions

Etapes à suivre :

- Télécharger le fichier ***TP5_PP_2021.zip*** à partir de Moodle, puis extraire le fichier dedans ***TP5_PP.ipynb***
- Ouvrir Jupyter notebook et télécharger (upload) le fichier ***TP5_PP.ipynb***, puis cliquer sur Téléverser
- Répondre aux questions de TP dans l'espace réservé de chaque question

Soit la base des voitures dans une agence, stockée dans le fichier ***voitures.txt***. Une voiture est caractérisée par sa marque, son modèle, son matricule, et son couleur

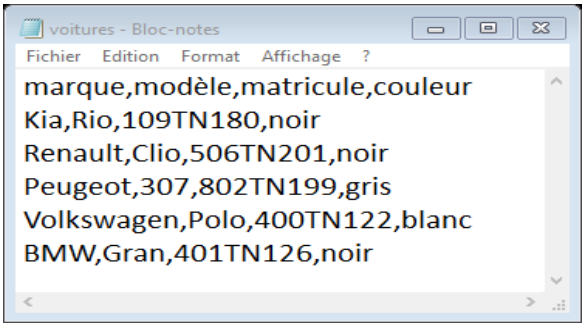


Figure 1 : voitures.txt

On peut se baser sur le programme principal suivant pour appeler les fonctions à développer :

```
Ajouter_voiture(marque='Peugeot', modèle='206', matricule='100TN210', couleur='blanc')
print(chercher())
to_csv()
entete(*['marque', 'modèle', 'matricule', 'couleur'])
try:
    print(nombre_voiture('vert'))
except AssertionError : print("Erreur!"")
```

Travail à faire :

1. Écrire une fonction ***ajouter_voiture***(** ***v***) qui ajoute la voiture ***v*** à la dernière ligne de ***voitures.txt***

2. Écrire un **générateur** ***gen_voiture***() qui permet de renvoyer toutes les listes possibles de format `[marque, modèle, matricule, couleur]` correspondants à la base des voitures du fichier ***voitures.txt*** (**la première ligne du fichier n'est pas incluse**)

3. Écrire une fonction ***chercher***(***c*** = `'noir'`) qui récupère les voitures de couleur ***c*** , et retourne un dictionnaire de ce format `{1:['marque', 'modèle', 'matricule'], 2:['marque', 'modèle', 'matricule'],...}` (utiliser ***gen_voiture***() pour récupérer les lignes du fichier ***voitures.txt***)

4. Écrire une fonction ***to_csv***(***mode*** =`'w'`) qui permet de créer un fichier nommé ***voitures.csv*** comportant toutes les lignes du fichier ***voitures.txt*** dont les éléments sont séparés par `";"` au lieu de `","` (utiliser ***gen_voiture***() pour récupérer les lignes du fichier ***voitures.txt***)

5. Écrire une fonction ***entete***(* ***h***) qui ajoute les éléments de ***h*** à la première ligne du fichier ***voitures.csv*** séparés par `";"`

6. Écrire une fonction ***nombre_voiture***(***c***= ***'noir'***) qui retourne le nombre de voiture de couleur ***c***. Une Exception de type ***AssertionError*** est levée s'il n'existe aucune voiture de couleur ***c*** dans ***voiture.txt***.

Bon Travail ♣

IIT INSTITUT INTERNATIONAL TECHNOLOGIE
Université Nord Américaine privée

PROGRAMMATION PYTHON

CORRECTION TP5

PRÉSENTÉ PAR

Dr. Taoufik BEN ABDALLAH
taoufik.benabdallah@iit.ens.tn

Dr. Rania REBAI BOUKHRISS
rania.rebai@iit.ens.tn

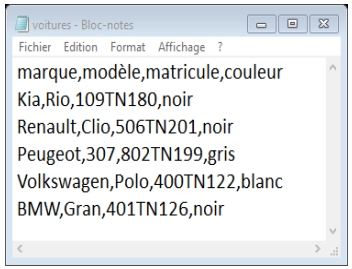
2021-2022

Travaux pratiques n°5 (1/6)

1. Écrire une fonction $ajouter\_voiture(**v)$ qui ajoute la voiture v à la dernière ligne de ***voitures.txt***

```
def ajouter_voiture(**v):
    chemin='D:/--------'
    with open(chemin+'/voitures1.txt','a') as f:
        f.write("\n"+",".join(list(v.values())))
```

```
ajouter_voiture(marque='Peugeot', modèle='206',
matricule='100TN210', couleur='blanc')
```

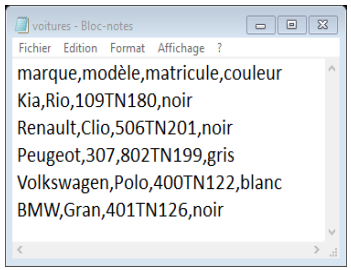


Travaux pratiques n°5 (2/6)

2. Écrire un **générateur** $gen\_voiture()$ qui permet de renvoyer toutes les listes possibles de format **[marque, modèle, matricule, couleur]** correspondants à la base des voitures du fichier ***voitures.txt*** (**la première ligne du fichier n'est pas incluse**)

```
def gen_voiture():
    chemin='D:/--------'
    with open(chemin+'/voitures1.txt') as f:
        f.readline()
        yield from [ligne.strip().split(',')
                    for ligne in f.readlines()]
```

```
for i in gen_voiture():
    print(i)
```



Travaux pratiques n°5 (3/6)

3. Écrire une fonction $chercher(c = \text{'noir'})$ qui récupère les voitures de matricule c , et retourne un dictionnaire de ce format **{1:['marque', 'modèle', 'matricule'], 2:['marque', 'modèle', 'matricule'],...}** (utiliser $gen\_voiture()$ pour récupérer les lignes du fichier ***voitures.txt***)

```
def chercher(c= 'noir'):
    g_v=gen_voiture()
    i=1
    d={}
    for v in g_v:
        if v[3]==c:
            d[i]=v[:3]
            i+=1
    return d
```

```
print(chercher())
```

Travaux pratiques n°5 (4/6)

4. Écrire une fonction $to\_csv(mode$ ='w') qui permet de créer un fichier nommé ***voitures.csv*** comportant toutes les lignes du fichier ***voitures.txt*** dont les éléments sont séparés par ";" au lieu de "," (utiliser $gen\_voiture()$ pour récupérer les lignes du fichier ***voitures.txt***)

```
from csv import *
def to_csv(mode='w'):
    chemin='D:/--------'
    with open(chemin+'/voitures.csv', mode, newline='') as f:
        wr= writer(f, delimiter=';')
        for i in gen_voiture():
            wr.writerow(i)
```

```
to_csv()
```


Travaux pratiques n°5 (5/6)

5. Écrire une fonction $entete(*h)$ qui ajoute les éléments de h à la première ligne du fichier ***voitures.csv*** séparés par ";"

```
def entête(*h):
 chemin='D:/--------'
    with open(chemin+'/voitures.csv', w, newline='') as f:
        wr= DictWriter(f, fieldnames=h, delimiter=";")
        wr.writeheader()
    to_csv(mode='a')
```

```
entête(*['marque', 'modèle', 'matricule', 'couleur'])
```


Travaux pratiques n°5 (6/6)

6. Écrire une fonction $nombre\_voiture(c = 'noir')$ qui retourne le nombre de voiture de couleur c. Une Exception de type ***AssertionError*** est levée s'il n'existe aucune voiture de couleur c dans ***voiture.txt***.

```
def nombre_voiture(c):
    d_v=chercher(c)
    assert len(d_v)!=0
    return len(d_v)
```

```
try:
    print(nombre_voiture('vert'))
except AssertionError :
    print("Erreur!")
```