langage python

Table des matières

I - Présentation du langage Python

1. Présentation du langage python

Python est un **langage** informatique **interprété** à usage généraliste , simple à apprendre et à utiliser, qui est essentiellement utilisé pour :

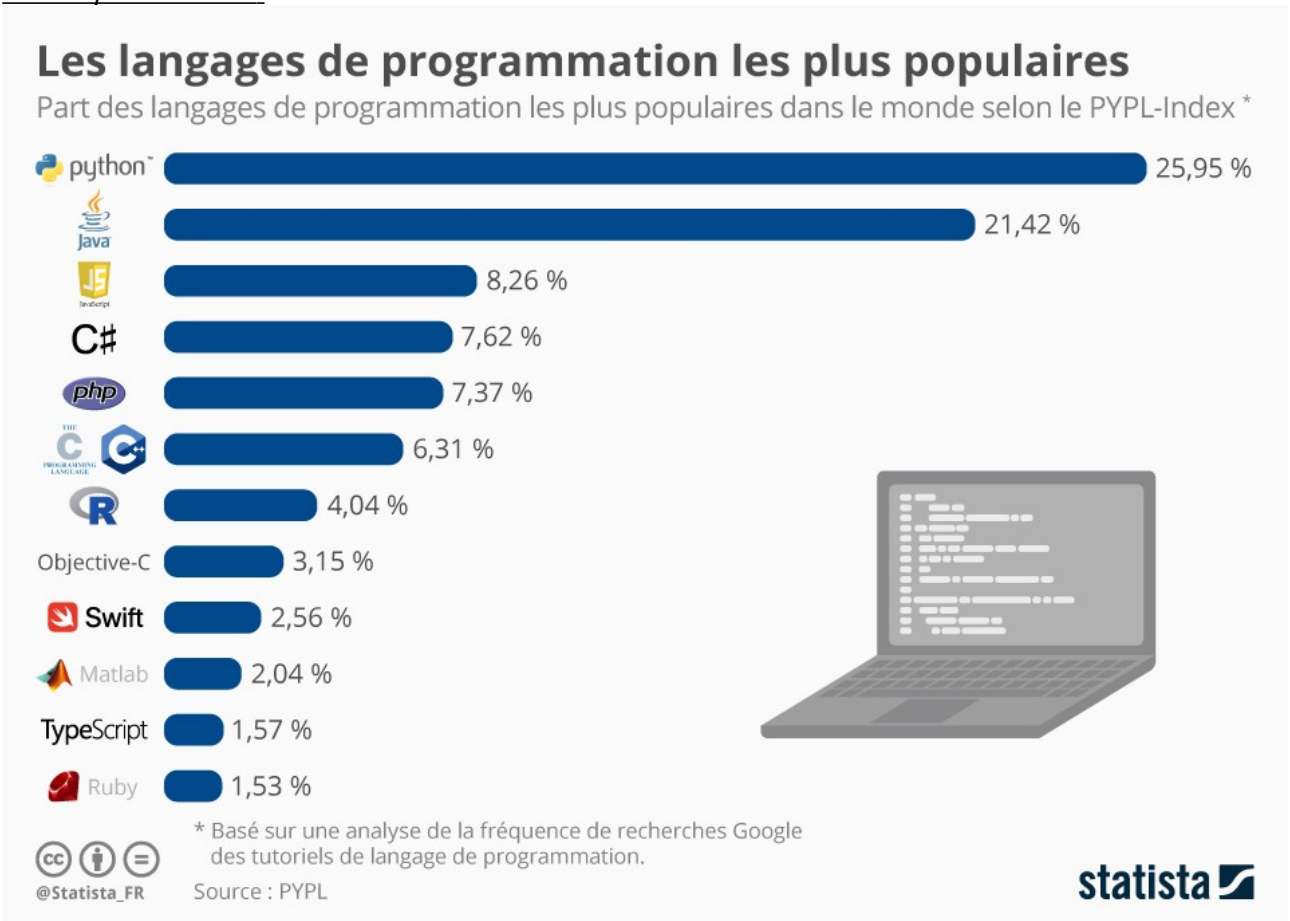
- **coder des petits scripts ou programmes simples**
- piloter/orchestrer des appels vers des fonctions prédéfinies efficaces (quelquefois codées en langage "C" de bas niveau et rapide)
- gérer des fichiers et/ou des grosses quantités de données (Big Data , Scrapping Web , ...)
- **piloter/orchestrer des calculs scientifiques**
- **analyse de données (statistiques, ...)**
- **l'intelligence artificielle (elle même basée sur des calculs mathématiques)**

Le langage python peut également être utilisé dans plein d'autres domaines (contrôles d'affichages graphiques, accès à des bases de données ou des fichiers, sites web, ...) .

1.1. Utilisation et popularité de Python

Langages de programmation les plus utilisés :

statistiques de 2019 :



Avec l'essor de l'intelligence artificielle, le langage python est de plus en plus utilisé.

1.2. Principales caractéristiques du langage python

simple	code facilement compréhensible et apprentissage rapide
interprété (et pas compilé)	comme les langages "basic" , "javascript" , "perl" , "ruby" , ... *avantages* : **souplesse et résultats immédiats** *inconvénients* : ***exécution pas rapide*** (si 100% python)
typage fort dynamique	au sein d'un programme python, les variables ont implicitement (sans devoir le préciser) des types précis (de manière automatique/dynamique en fonction des affectations).
à usage général / polyvalent	on trouve des bibliothèques de fonctions prédéfinies pour presque tous les usages (calculs , accès bases de données , affichages , ...)
multi-paradigme	code procédural et séquentiel pour scripts simples code en mode "orienté objet" pour applications élaborées mode fonctionnel avec lambda
robuste et fiable	gestion automatiques des ressources (mémoire, descripteur de fichier), gestion d'exception
mature et bien implanté	python existe depuis plus de 25 ans et est beaucoup utilisé dans le domaine scientifique et dans le cadre de l'administration système (ex : linux, ...)
open-source	accès et usage libre et gratuit . Seules quelques extensions "clefs en main" sont quelquefois payantes .
multi-plateformes	étant interprété, le langage python peut facilement être utilisé sur tout type de plateformes (Windows , Linux , Mac , smartphones, ...)
syntaxe avec indentations	contrairement à beaucoup d'autres langages qui délimitent des blocs de code via { et } ou via "begin" et "end" , le langage python n'utilise pas de délimiteur mais a une structure de code contrôlée par des indentations (décalages par rapport de débuts des lignes)

Bien qu'intéressant sur bien des points, le langage Python n'est pas interprété par les navigateurs "web/internet" . Les navigateurs "IE , Firefox , Chrome, ..." ont historiquement fait le choix d'utiliser le langage interprété javascript (bien aussi et un peu plus rapide que python) .

Licence (libre) : Python Software Foundation License

Extension des fichiers: **.py**

1.3. Historique et évolution

Première version publique officielle : 0.9.0 en **1991** développé par "Guido van Rossum" (Pays bas , Amsterdam).

De 1995 à 1999 : évolution du langage aux états-unis (CNRI, ...) , **version 1.6 en 1999**

A partir de **2001** et la version 1.6.1 , l'évolution du langage python est contrôlée par la "***Python Software Foundation***".

Python a longtemps existé en version 2.x (durant la décennie 2000-2010) .

Les version 3.x (à partir de 2008/2010) sont sur certains petits points en rupture avec les version 2.x. Il est donc conseillé aujourd'hui de ne plus utiliser l'ancienne version 2 (dans la mesure du possible)

Bonnes versions relativement récentes et stables du langage python :
3.7 (2018)
3.9 (2020)
3.11(2022)
3.13(2024)

1.4. Distributions de "python"

En tant que langage interprété , python peut être pris en charge par plusieurs variantes du moteur d'interprétation.

L'implémentation de référence est **CPython** (interpréteur codé en langage C) .
Il existe aussi **Jython** (basé sur une machine virtuelle java) , ...

En outre, on trouve aujourd'hui certaines distributions packagées de python incluant "moteur d'interprétation + éditeurs + ensemble de bibliothèques + ...) .

Les principales distributions (basées sur CPython) sont :

- **WinPython**
- **Anaconda**
- ...

La plupart des distributions sont à usage scientifique et elles incluent les bibliothèques de calculs ***numPy*** et ***sciPy*** .

Autrement dit, pour installer python sur un ordinateur on peut installer que python ou bien toute une distribution telle qu'anaconda .

Néanmoins, pour un usage général et standard , il est conseillé d'installer et d'utiliser la version la plus classique de python (basée sur CPython) et accessible sur le site de référence https://www.python.org .

II - Prise en main de l'interpréteur Python

1. Installation et première utilisation de python

Avant de pouvoir utiliser le langage python il faut installer un des interpréteurs disponibles

1.1. Quelques installations possibles de python

Python est souvent installé d'office sur les distributions linux mais pas forcément dans une version récente . Par exemple sur **linux ubuntu 22.04** , le phython 3.10 (souvent pré-installé) se lance avec la commande "**python3**" ou bien l'alias "**python**" .

Pour installer python sur un ordinateur windows , on peut se connecter sur le site officiel **https://www.python.org/** et effectuer un téléchargement de l'installeur via l'url ***https://www.python.org/ftp/python/3.13.3/python-3.13.3-amd64.exe*** (environ 28 Mo) .

On peut éventuellement ajouter Python au PATH pour pouvoir ultérieurement le lancer facilement depuis une fenêtre CMD ou autre .

1.2. Prise en main de l'interpréteur python

Au sein d'une fenêtre de commande (CMD ou PowerShell ou shell linux ou ...) , la commande **python --version** permet d'afficher la version de l'interpréteur python (ex : 3.13.3) .

En lançant la commande "**python**" sans argument , on peut ainsi **lancer l'interpréteur python en mode interactif** . Celui ci nous invite alors à saisir des commandes après une invite (prompt) **">>>"** .
Toute "commande / ordre / expression" saisi(e) est alors immédiatement interprété(e) et le résultat s'affiche immédiatement .

```
python
Python 3.7.6 .....
>>> 2+3
5
>>> 4*6
24
>>>   exit
```

1.3. Ecriture et lancement d'un programme python

Un fichier de code python a par convention l'extension "**.py**" et peut être saisi avec un très grand nombre d'éditeur (nodepad++, visual studio code , ...) .

Exemple: **hello.py**

```
a=2 ; b=3 ; c=a+b
print('Hello world')
print('c=',c)
```

Pour lancer ce script (petit programme) , on peut lancer la commande

python hello.py

Ce qui provoque l'affichage suivant :

Hello world
c= 5

NB: **py** est souvent (selon le contexte) un **alias** pour **python** et on peut donc souvent lancer la commande alternative suivante :

py hello.py

Et pour automatiser un lancement depuis un double click via l'explorateur de fichiers de windows on peut éventuellement écrire un fichier de lancement tel que celui ci :

lancer_prog_python.bat

```
REM avec PATH contenant le répertoire d'installation de python
python hello.py
pause
```

1.4. Instructions élémentaires du langage python .

NB : En langage python les **commentaires** sont des **fin de lignes commençant par #**

p1.py

```
a=1  # nomVariable=valeur_a_affecter
print(a) # affiche la valeur de la variable a (ici 1)
a=2
print(a) # affiche la nouvelle valeur de la variable a (ici 2)
b=a*3+4  # variableResultat = expression d'un calcul
c=a+b
print("b=",b,"c=",c)  # affiche plusieurs choses en les séparant par des espaces
                      # affiche ici b= 10 c= 12
prenom = "alex"  # une chaine de caractères est délimitée en python par des " " ou des ' '
nom = 'Therieur'
nomComplet = prenom + ' ' + nom # concaténation (ajout bout à bout)
print(nomComplet) # affiche alex Therieur

# input('texte question') demande à saisir/renseigner une valeur
age = input ("quel est ton age ? ")
print ('age renseigné:' , age);  # affichera la valeur choisie/précisée .
```

Attention: *un + entre 2 chaînes de caractères (string) déclenche une concaténation (valeurs juxtaposées bout à bout) tandis qu'un + entre 2 nombres déclenche une addition*

*La fonction **float( ) converti une chaîne de caractères en une valeur numérique** (avec potentiellement une virgule notée "." en anglais) .*

Exemple :

```
a=input('a:') # exemple a: 2 et a vu comme '2'
b=input('b:') # exemple b: 3 et b vu comme '3'
c=a+b # '2' + '3' = '23'
print('c=a+b=',c) # affiche par exemple c=a+b= 23

a=float(input('a:')) # exemple a: 2 et a vu comme 2.0
b=float(input('b:')) # exemple b: 3 et b vu comme 3.0
c=a+b # 2.0 + 3.0 = 5.0 = 5
print('c=a+b=',c) # affiche par exemple c=a+b= 5
```

Quelques exercices :

Ecrire un petit script calculant la moyenne de 2 nombres avec des valeurs à saisir :

x=*4 ou 2 ou ...*
y=*8 ou 10 ou ...*
moyenne= *6.0 ou ...*

moyenne.py

Autres essais libres. c'est en forgeant que l'on devient forgeron.

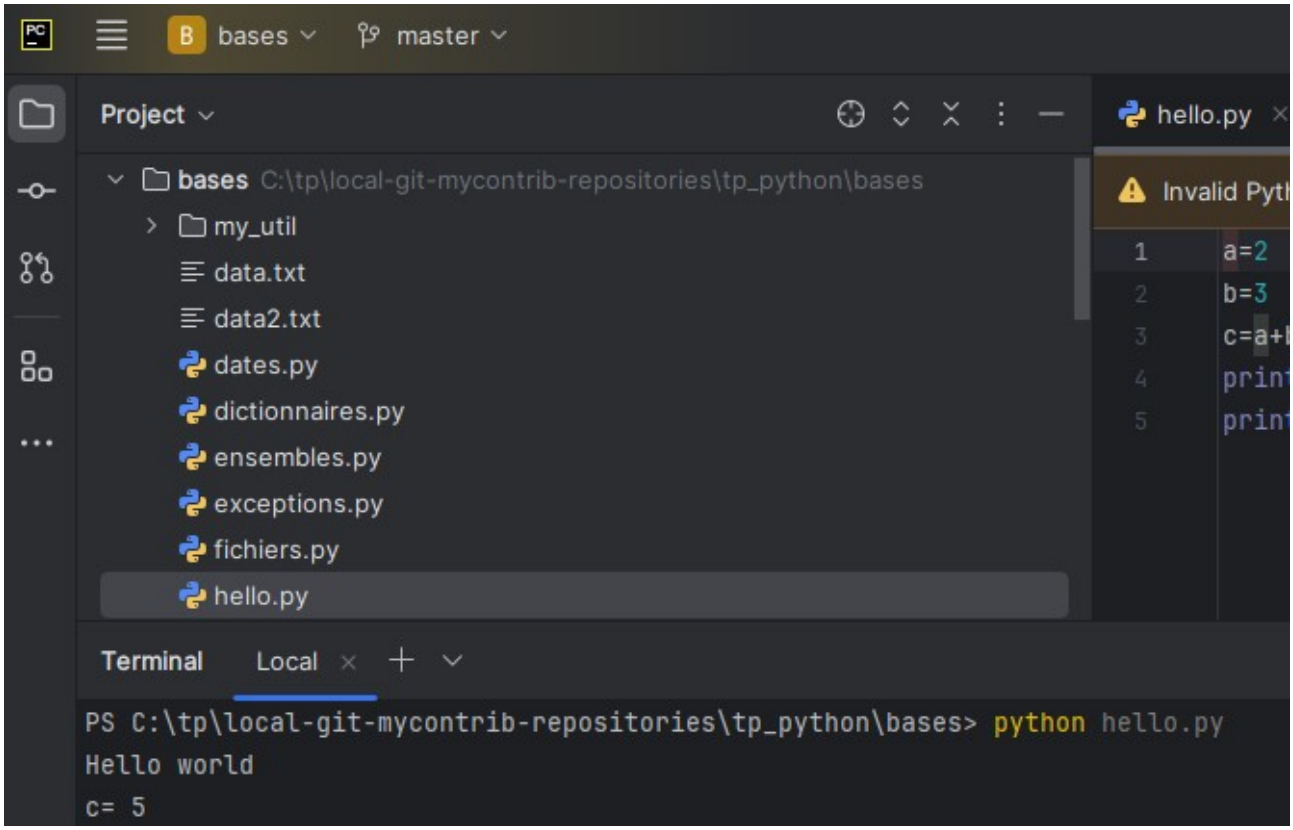
1.5. Commentaires multi-lignes et séparateur d'instructions

```
x=2 ; y=3 ; z=x*y ; print("z=x*y=",z) #  ; est un séparateur d'instructions (sur même ligne)
'''
commentaire sur plusieurs lignes
délimité par triple simple quote .
'''
print('suite')
```

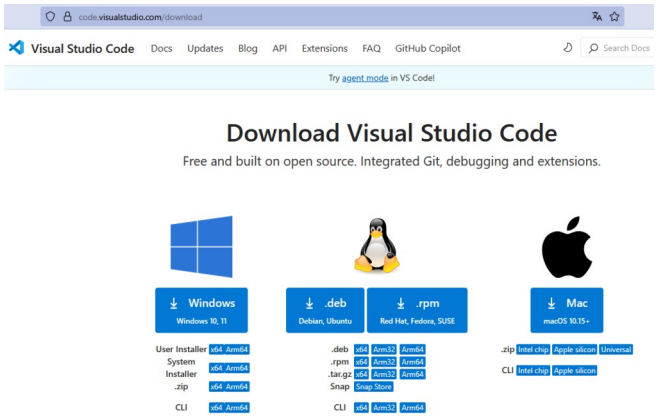
2. IDE (éditeur de code) pour python

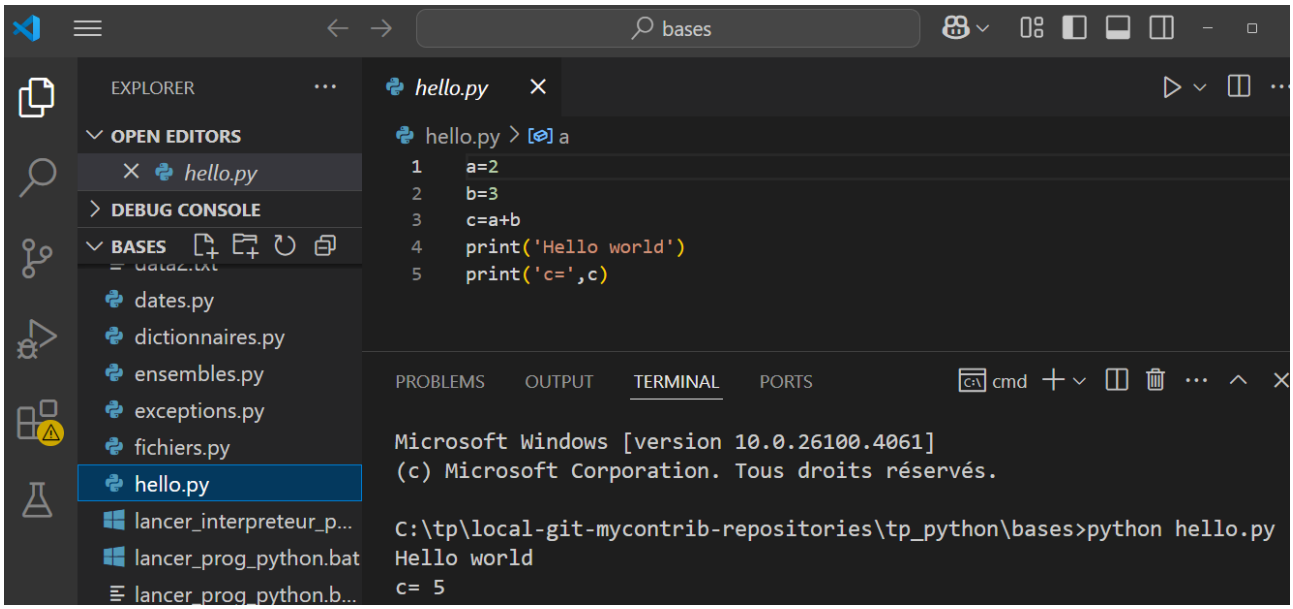
2.1. Pycharm

Bon Editeur python de l'entreprise JetBrains (comme IntelliJ, webstorm, …)



2.2. Visual Studio Code avec extension python





III - Syntaxes élémentaires , types , boucles

1. Types et syntaxes élémentaires de Python

1.1. Types de données en python 3

Comme beaucoup d'autres langages interprétés (tel que javascript) , le langage python a un ***typage dynamique*** :
Il n'est pas nécessaire de préciser le type d'une variable.
Une variable python a , à un instant donné, un type qui dépend de la valeur affectée .

Les principaux types élémentaires sont précisés dans le tableau ci-après

type	*caractéristiques*	*exemples*
int	nombre entier	**-12** **0** **234**
float	nombre à virgule flottante	**-12.5** **0.0** **234.78**
complex	nombre complexe (avec partie imaginaire)	**2+4j**
str	chaîne de caractères (string)	**'abc'** **"def"**
bool	booléen (True or False)	**True** **False**
list	liste (ou tableau redimensionnable) d'éléments	**["rouge" , "vert" , "bleu"]**

la fonction type() renvoie le type (à l'instant présent) d'une variable python .

```
a=128
type(a) # retourne int , affichage complet : <class 'int'>
b='bonjour'c1
type(a) # retourne str , affichage complet : <class 'str'>
```

NB :

- Pour les nombres complexes , la lettre j a été choisie à la place de i car i est souvent utilisé comme indice pour les boucles.
- La valeur spéciale **None** (ressemblant à null de java/javascript) est de type **NoneType** .

NB: bien que ce ne soit pas très conseillé , une variable python peut changer de type au cours du temps selon les affectations successives :

```
x=1             #int
x=1.2           # float
x=False         # bool
x="abc"         # string
listeHeterogene = [ 1 , "abc" , True , 5.5 ]  # list
```

Opérateurs arithmétiques :

```
a=5;b=float('6')
c=a+b;print('c=',c) # c= 11


x=5;y=3
z=x+y ; print(z)  #addition , z=8
z=x-y ; print(z)  #soustraction , z=2
z=x*y ; print(z)  #multiplication , z=15
z=x/y ; print(z)  #division , z=1.666666
z=x%y ; print(z)  #modulo (reste division entière) , z=2
z=x//y ; print(z)  #division entière(floor)  , z=1
z=x**y ; print(z)  #exponentiation (puissance) , z=125
```

Affectations (assignements) et combinaisons:

```
x=4 ; print(x) # x=4
x=4 ; x+=3; print(x) # x=x+3=7
x=4 ; x-=3; print(x) # x=x-3=1
x=4 ; x*=3; print(x) # x=x*3=12
x=4 ; x/=3; print(x) # x=x/3=1.333333
x=4 ; x%=3; print(x) # x=x%3=1
x=4 ; x//=3; print(x) # x=x//3=1
x=4 ; x**=3; print(x) # x=x**3=6
```

1.2. Bloc d'instructions et indentation

Dans beaucoup d'autres langages (ex : C/C++ , java , javascript, ...) , un bloc d'instruction est délimité par { et } (début et fin).

Le langage python a quant à lui choisi de délimiter un bloc d'instruction par un niveau d'indentation (un décalage homogène par rapport au début de ligne) : Toutes les instructions décalées via un même nombre d'espaces seront considérées comme appartenant à un même bloc . On conseil généralement 4 espaces pour différencier un sous-bloc de son bloc parent .

exemple en langage **C**	exemple en langage **python**

`int factorielle(int n)`
`{`
`if (n < 2) { return 1; }`
`else {`
`return n * factorielle(n - 1);`
`}`
`}`
	`def factorielle(n):`
	`if n < 2:`
	`return 1`
	`else:`
	`return n * factorielle(n - 1)`

NB :
- Python conseil l'utilisation de plusieurs espaces consécutifs (idéalement 4).
- Des tabulations peuvent éventuellement être utilisées à la place mais il faut choisir entre série d'espaces et tabulations : un mélange des 2 styles est normalement interdit par python3.

1.3. Tests et opérateurs logiques (comparaisons, ...)

```
if condition :
   bloc d'instruction déclenché si condition vérifiée
```

ou

```
if condition :
   bloc d'instruction déclenché si condition vérifiée
else :
   bloc d'instruction déclenché sinon (si condition non vérifiée)
```

Une condition (à vérifier) est généralement formulée comme un test de comparaison :

x == y	égal à
x != y	différent de
x > y	strictement supérieur à
x >= y	supérieur ou égal à
x < y	strictement inférieur à
x <= y	inférieur ou égal à

Exemples :

```
age=20
if age>=18:
    print('majeur pour age=',age)
    print('pas mineur')
print('suite dans tous les cas')

age=16
if age>=18:
    print('majeur pour age=',age)
else :
    print('mineur pour age=',age)
print('suite dans tous les cas')
```

==>

majeur pour age= 20
pas mineur
suite dans tous les cas
mineur pour age= 16
suite dans tous les cas

Une condition peut quelquefois être exprimée comme une combinaison logique de sous condition.
Le mot clef **and** correspond à un "et logique" : les 2 sous conditions doivent être vraies
Le mot clef **or** correspond à un "ou logique" : au moins une des 2 sous conditions doit être vraie

Exemples :

```
age=30
if (age>=18) and (age<=42) :
        print('age entre 18 et 42 ans')
print('suite dans tous les cas')

age=16
if (age<18) or (age>=65) :
        print('enfant ou bien personne agée')
print('suite dans tous les cas')
```

Instruction **pass**

```
"""
L'instruction if ne peut pas être vide . si pour une raison quelconque on souhaite y placer
une instruction sans contenu , on peut utiliser l'instruction pass pour éviter une erreur
"""
x1=5; x2=6
if x2 > x1 :
        pass # ne rien faire pour l'instant (version 1 temporaire)
print("suite apres if ne faisant rien pour l'instant")
```

avec **elif** (~~elseif~~):

```
heure=15
if heure<=12 :
    print("bonjour!")
elif heure <=18 :
    print("bon après midi!")
else :
    print('bonsoir!')
```

NB: L'instruction match/case n'a été introduite en python que très tardivement (version 3.10 , vers 2021) . Dans la plupart des programmes python en production on s'appuiera donc sur if et elif .

Pour toutes les versions de python	Seulement avec python ≥ 3.10
couleur = "vert" **if** couleur == "rouge" : print("red") **elif** couleur=="vert" : print("green") **elif** couleur=="bleu" : print("blue") **else**: print("other color")	couleur = "vert" **match** couleur: **case** "rouge" : print("red") **case** "vert" : print("green") **case** "bleu" : print("blue") **case** _ : print("other color / *default*")

affectation ternaire :

```
age=33
etat="majeur" if age >=18 else "mineur"
# équivalent en c,c++,c#,java,javascript: etat=(age>=18)?"majeur":"mineur"
print("etat=",etat , "pour age" , age) # etat= majeur pour age 33

age=15
etat="majeur" if age >=18 else "mineur"
print("etat=",etat, "pour age" , age) # etat= mineur pour age 15
```

condition avec imbrications :

```
x=25
if x >= 20 :
        print("x >= 20")
        if x >= 30 :
                print("et aussi x >=30")
        else :
```

```
                print("et x < 30 (pas >= 30)")
```

1.4. 4 grands types de Collections en python

collections	caractéristiques	exemples
list	liste/tableau ordonné et modifiable	l1 = [2 , 5 , 2 , 4]
tuple	tableau ordonné non modifiable	t1 = (8 ,6, 3)
set	Ensemble sans doublon non ordonné, pas indéxé	s1 = { "rouge", "vert" , "bleu" }
dictionary	Table d'association sans doublon et modifiable	d1 = { 'label' : "c1" , 'solde' : 200}

1.5. Listes et fonction len() pour connaître la taille/longueur

Un ***tableau redimensionnable*** (appelé **list** en python) est ***une collection de valeurs consécutives*** dont les **positions** (appelés **indices**) vont de **0 à n-1** .

La fonction prédéfinie **len**(liste) retourne la **taille (ou longueur) d'une liste ou d'un tableau**
La syntaxe *listeXy*[*positionN*] permet d'accéder directement à un élément de la liste .

Exemples :

```
listeDeCouleurs =        [ 'rouge' , 'vert' , 'bleu' , 'noir' , 'blanc' ]
# indices ou positions :      0        1        2        3        4
print('la taille de liste de couleurs est ' , len(listeDeCouleurs) ) # affiche 5
print('la première couleur est ' , listeDeCouleurs[0] ) # affiche rouge
print('la couleurs du milieu est ' , listeDeCouleurs[2] ) # affiche bleu
print('la dernière couleur est ' , listeDeCouleurs[4] ) # affiche blanc
print('en dernier ' , listeDeCouleurs[ len(listeDeCouleurs)-1 ] ) #  blanc
```

Syntaxes spéciales pour plage d'index :

print('plage **1 inclus à 3 exclus** :' , listeDeCouleurs**[1:3]**) # *['vert', 'bleu']*

print('plage **début à 3 exclus** :',listeDeCouleurs**[:3]**) # *['rouge', 'vert', 'bleu']*

print('plage **2 inclus à fin** :',listeDeCouleurs**[2:]**) # *['bleu', 'noir', 'blanc']*

1.6. range() , séquence et généralités sur les boucles

Une **séquence** (appelée également ***tuple***) est une **succession de valeur séparée par des virgules** et est assez souvent générée par la fonction **range**()

La fonction **list**(1 , 2 , 3) **construit une liste** (ici **[1, 2, 3]**) **à partir d'une séquence .**

La fonction prédéfinie **range**(**posDebut,posArret,pas**) ***construit la séquence***
posDebut , posDebut+pas , posDebut+2*pas , ...
qui s'arrête lorsque ***posDebut+n*pas*** n'est plus strictement inférieur à ***posArret*** .

Au sein de la fonction range(*posDebut*,**posArret**,*pas*) , le paramètre facultatif *posDebut* a la valeur par défaut *0* s'il n'est pas précisé et le paramètre facultatif *pas* vaut 1 par défaut .

Concrètement :

- La fonction prédéfinie **range**(**n**) construit la séquence **0 , 1 , 2 , ... , n-1**
- **range(3,6)** construit la séquence **3 , ... , 6-1**
- **range(3,-1,-1)** construit la séquence inversée **3 , 2 , 1 , 0**

Exemples :

```
print('range(4)=de 0 à 3<4 =', list(range(4))) # affiche [ 0,1,2, 3 ]
print('range(3,6)=de 3 à 5<6 = ', list(range(3,6))) # affiche [ 3, 4 , 5 ]
print('range(4-1,0-1,-1)= séquence inversée = ', list(range(3,-1,-1))) # affiche [ 3, 2 , 1 , 0 ]
```

En programmation, une boucle permet d'exécuter plusieurs fois un même bloc d'instructions (avec souvent une petite variante) .

En python la boucle ***for*** signifie "***pour chaque éléments de ...***" ou bien "***pour chaque indice dans un certain intervalle***"
La boucle ***while*** signifie "***tant que la condition est vraie***"

1.7. Boucle for (pour chaque élement de ...)

```
#indicesDe0a3= 0,1,2,3
indicesDe0a3= range(4) # range(4) construit 0, 1, 2, 3
for i in indicesDe0a3 :
        print("i=",i)
print('suite apres la boucle')
```

affiche

```
i= 0
i= 1
i= 2
i= 3
suite apres la boucle
```

```
print('boucle de 2 à 8<9 par pas de 2 soit de 2 en 2 :');
for i in range(2,9,2) :
        print("i=",i)
print('suite apres la boucle')
```

==>

```
i= 2
i= 4
i= 6
i= 8
suite apres la boucle
```

```
joursDeLaSemaine = ['lundi','mardi','mercredi','jeudi','vendredi','samedi','dimanche']
# la liste (ou tableau) joursDeLaSemaine comporte 7 éléments dont les indices vont de 0 à 6
for jour in joursDeLaSemaine :
        print(jour,'est un element de joursDeLaSemaine')
print('suite apres la boucle')
```

==>

```
lundi est un element de joursDeLaSemaine
mardi est un element de joursDeLaSemaine
mercredi est un element de joursDeLaSemaine
jeudi est un element de joursDeLaSemaine
vendredi est un element de joursDeLaSemaine
samedi est un element de joursDeLaSemaine
dimanche est un element de joursDeLaSemaine
```

```
#nbJoursDansUneSemaine=7
nbJoursDansUneSemaine=len(joursDeLaSemaine) # la fonction len() retourne la taille/longueur
                                            #(length en englais)
for i in range(nbJoursDansUneSemaine) :
        print("i=",i,"jour=",joursDeLaSemaine[i])
print('suite apres la boucle')
```

==>

```
i= 0 jour= lundi
i= 1 jour= mardi
i= 2 jour= mercredi
i= 3 jour= jeudi
i= 4 jour= vendredi
i= 5 jour= samedi
i= 6 jour= dimanche
```

En exercices :

1. coder et exécuter avec python tous les exemples ci dessus (sans les commentaires et sans copier/coller) pour se familiariser avec la syntaxe .
2. calculer la somme et la moyenne de la liste [12 , 48 , 32 , 8 , 24] . *solution en fin de chapitre*

Boucle for avec else :

```
languages=["python" , "javascript"];
for l in langages :
        print("l=",l)
else:
        print("--- fin de liste ----")
```

Boucle for avec break :

```
valeurs=[-1, -8, 6 , -9 , 12,-3, 4]
premierPositif=None
for v in valeurs :
        if v >=0 :
                premierPositif=v
                break # fin anticipée de boucle (premierPositif déjà trouvé)
print("premierPositif=",premierPositif) # 6
```

Boucle for avec continue :

```
valeurs=[-1, -8, 6 , -9 , 12,-3, 4]
for v in valeurs :
        if v <0 :
                continue # passer directement à l'itération suivante
        print("positive v=",v) #instruction declenchée que si pas continue avant
```

==>positive v= 6 positive v= 12 positive v= 4

Boucles imbriquées :

```
matrice = [
        [ 4 , 0 , 5] ,
        [ 2 , 6 , 1] ,
        [ 0 , 3 , 0]
]
for i in range(3) :
        for j in range(3):
                print("matrice("+str(i)+","+str(j)+")=" , matrice[i][j])
```

⇒

matrice(0,0)= 4 matrice(0,1)= 0 matrice(0,2)= 5
matrice(1,0)= 2 matrice(1,1)= 6 matrice(1,2)= 1
matrice(2,0)= 0 matrice(2,1)= 3 matrice(2,2)= 0

1.8. Boucle while (tant que ...)

```
x=0
while x <= 3 :
        print ('x=',x,'x*x=',x*x)
        x=x+1
print('suite apres la boucle')
```

==>

```
x= 0 x*x= 0
x= 1 x*x= 1
x= 2 x*x= 4
x= 3 x*x= 9
suite après la boucle
```

Attention: de pas oublier x=x+1 sinon boucle infinie qui ne s'arrête jamais !!!
pas de until , ni do … while en python .

```
listeCouleurs = [ "rouge" , "noir" ]
listeCouleurs.append("vert") # ajoute l'élément "vert" à la liste
listeCouleurs.append("bleu") # ajoute l'élément "bleu" à la liste
print("listeCouleurs=",listeCouleurs); # ['rouge', 'noir', 'vert', 'bleu']
```

```
listeValeurs = []
valeurOuFin=input("val=")
while (valeurOuFin != 'fin' ) and (valeurOuFin != '' ) :
        nouvelleValeur = float(valeurOuFin)
        listeValeurs.append(nouvelleValeur)
        valeurOuFin=input("val=")
print('listeValeurs=',listeValeurs)
```

==>

```
val=4
val=6
val=
listeValeurs= [4.0, 6.0]
```

En exercices :

3. coder et exécuter avec python tous les exemples ci dessus (sans les commentaires et sans copier/coller) pour se familiariser avec la syntaxe .
4. calculer la somme et la moyenne de la liste [12 , 48 , 32 , 8 , 24] sans utiliser la boucle for mais en utilisant la boucle while et en initialisant une variable i=0 avant la boucle .
5. transformer ["hiver" , "printemps" , "ete" , "automne"] en liste de valeurs en majuscules via une boucle au choix (for ou while) puis afficher toute la liste transformée .

NB : Solutions des "mini-exercices" sur page suivante.

Solutions des mini-exercices :

```
l1 = [ 12 , 48 , 32 , 8 , 24 ]
somme=0
taille=len(l1)
for i in range(taille):
    somme += l1[i]
print('somme=',somme)
print('moyenne=',somme/taille)
```

ou bien

```
l2 = [ 12 , 48 , 32 , 8 , 24 ]
somme=0
for val in l2:
    somme += val
print('somme de l2=',somme)
print('moyenne de l2=',somme/len(l2))
```

somme= 124
moyenne= 24.8

```
# avec while :
l1 = [ 12 , 48 , 32 , 8 , 24 ]
somme=0 ; taille=len(l1)
i=0
while i < taille :
    somme += l1[i]
    i+=1
```

```
saisons = ["hiver" , "printemps" , "ete" , "automne" ]
saisonsMaj = []
for i in range(len(saisons)) :
    saisonsMaj.append(saisons[i].upper())

print("saisonsMaj=",saisonsMaj)
```

saisonsMaj= ['HIVER', 'PRINTEMPS', 'ETE', 'AUTOMNE']

IV - Structures de données (liste,dictionnaire,...)

1. Manipulation de listes en python

Le langage python gère des listes comme des tableaux redimensionnables .
Une liste python ressemble à un tableau javascript ou une ArrayList de java.
Au sein d'une liste python, les éléments sont ordonnés et les indices valides vont de 0 à n-1.

```
listeVide=[]

listeInitiale=[1,2,3]

liste=listeInitiale
liste.append(4) # ajoute un nouvel élément en fin de liste
print("liste=",liste) # affiche [1,2,3,4]

print("premier élément:",liste[0]) # affiche 1
liste[0]=1.1
print("premier élément modifié:",liste[0]) # affiche 1.1

#liste[4]=5  --> IndexError: list assignment index out of range

print("dernier élément:",liste[-1]) # affiche 4 (dernier élément)
```

```
liste2 = [ "a" , "b" , "c" ]
del liste2[1] # suppression de l'élément d'indice 1 (0,1,2)
print("liste2=",liste2) # affiche [ 'a' , 'c' ]

liste3 = [ "rouge" , "vert" , "bleu" ]
liste3.remove("vert") # suppression de l'élément dont la VALEUR est "vert"
print("liste3=",liste3) # affiche [ 'rouge' , 'bleu' ]
```

```
liste4=[1,2,3,4];
liste4.reverse() # inverse l'ordre des éléments de la liste
print("liste4=",liste4) # affiche [ 4,3,2,1 ]
```

```
nbElements=len(liste4)
print("longueur (nbElements) de liste4=",nbElements) # affiche 4
```

```
liste5=['a' , 'b' , 'a' , 'b' , 'c' , 'a' ]
nbOccurencesDeA = liste5.count('a')
print("nbOccurences de 'a' dans liste5=",nbOccurencesDeA) # affiche 3
```

```
indiceBdansListe5= liste5.index('b')
print("indiceBdansListe5",indiceBdansListe5) # 1 (premier trouvé)

indiceCdansListe5= liste5.index('c')
print("indiceCdansListe5",indiceCdansListe5) # 4
```

```
#liste5.index('e') --> ValueError: 'e' is not in list
```

```
autreListe = [ 'a' , 'b' , 'c1', 'd' , 'e']
autreListe.insert(3,'c2') #insertion avant l'élément actuel d'indice 3
                          #insertion du nouvel élément en pos 3 et en décalant la fin
print('autreListe apres insertion:' , autreListe)
#autreListe apres insertion: ['a', 'b', 'c1', 'c2', 'd', 'e']
```

Boucle sur indices (rappel):

```
liste = [ 'a' , 'b', 'c' ]
for i in range(len(liste)) :
        print('i=',i,'liste[i]=',liste[i])
# i= 0 liste[i]= a
# i= 1 liste[i]= b
# i= 2 liste[i]= c
```

Boucle directement et uniquement sur valeurs :

```
liste = [ 'a' , 'b', 'c' ]

#boucle sur valeur des éléments:
for val in liste :
        print(val)
# a
# b
# c
```

Boucle directe à la fois sur indices et valeurs des éléments:

```
for tuple_indice_val in enumerate(liste) :
        print(tuple_indice_val , tuple_indice_val[0] , tuple_indice_val[1] )
# (0,'a') 0 a
# (1,'b') 1 b
# (2,'c') 2 c
```

```
liste6 = [ 2 , 4 , 6 ]
refListe = liste6 # refListe référence la même liste que celle référencée par liste6
refListe[0]=2.2 # même effet que liste6[0] = 2.2
print("liste6=",liste6) # affiche [ 2.2 , 4 , 6 ]

liste7 = [ 1 , 3 , 5]
liste8 = liste7.copy() # copie/duplication d'une liste
liste8[0]=1.1
print("liste7=",liste7) # affiche [1, 3 , 5]
print("liste8=",liste8) # affiche [1.1, 3 , 5 ]
```

```
maPile = [ 1 , 2 ,  3 , 4]
dernierElementRetire  = maPile.pop(); print(dernierElementRetire); # 4
dernierElementRetire  = maPile.pop(); print(dernierElementRetire); # 3
print(maPile); # [ 1, 2 ]
```

```
troisCouleurs="rouge;vert;bleu" # grande chaîne de caractères avec sous parties séparées par ";"
listeCouleurs = troisCouleurs.split(";")
print("listeCouleurs=",listeCouleurs) # ['rouge', 'vert', 'bleu']

mesCouleurs=";".join(listeCouleurs) # transforme liste en chaîne de caractères
#attention : très différent de java/javascript (listeJs.join(";"))
print("mesCouleurs=",mesCouleurs) # affiche la chaîne rouge;vert;bleu
```

```
liste10=[ 'a' , 'b' , 'c' ]
if 'a' in liste10 :
        print('liste10 comporte a')
else :
        print('liste10 ne comporte pas a')

if 'd' in liste10 :
        print('liste10 comporte d')
else :
        print('liste10 ne comporte pas d')
```

2. Set (ensembles)

```
ensembleVide={}

#dans un ensemble, les éléments ne sont pas ordonnés (sans index stable)
#dans un ensemble, chaque élément est unique (sans duplication possible)

ensembleDeFruits={"apple", "banana", "cherry"}

print("ensembleDeFruits=",ensembleDeFruits) # {'banana', 'cherry', 'apple'}
```

```
for f in ensembleDeFruits:
        print(f)
#banana
#cherry
#apple
```

```
#NB: une fois qu'un ensemble a été créé/initialisé , on ne peut pas modifier
# ses éléments mais on peut ajouter un nouvel élément via .add()
# ou bien de nouveaux éléments via .update()

ensembleDeFruits.add("orange") # ajout (sans notion d'ordre)
print(ensembleDeFruits) # {'apple', 'orange', 'banana', 'cherry'}

ensembleDeFruits={"apple", "banana", "cherry"}
ensembleDeFruits.update({"orange" , "peach"}) # ajout de plusieurs éléments
print(ensembleDeFruits) # {'apple', 'peach', 'banana', 'orange', 'cherry'}
```

```
ensembleDeFruits={"apple", "banana", "cherry"}
ensembleDeFruits.remove("banana") # supprime un élément s'il existe , erreur sinon
```

```
print(ensembleDeFruits)  # {'cherry', 'apple'}
ensembleDeFruits.discard("banana") # supprime un élément s'il existe toujours sans erreur

ensembleDeFruits.clear() # vide l'ensemble
print(ensembleDeFruits)  # {}
```

```
set1 = {"a", "b" , "c"}
set2 = {"d", "e", "f"}

set3 = set1.union(set2)
print(set3) # {'b', 'd', 'a', 'e', 'c', 'f'}

set4 = {"a", "b" , "c" , "d"}
set5 = {"c", "d" , "e" , "f"}

set6 = set4.intersection(set5)
print(set6) # {'d', 'c'}

#il existe également .isdisjoint() , .issubset() , .difference() , ...
```

Transformations de set en liste , tuple et vice versa :

```
set1 = {"a",  "b" , "c" , }
listeFromSet1 = list(listeFromSet1)
print("listeFromSet1=",listeFromSet1,type(listeFromSet1))
#listeFromSet1= ['b' 'c', 'a'] <class 'list'> # NB: ordre aléatoire/variable
```

```
tuple2 = ("a", "b" , "c"  )
listeFromTuple2 = list(tuple2)
print("listeFromTuple2",listeFromTuple2,type(listeFromTuple2))
# listeFromTuple2 ['a', 'b', 'c'] <class 'list'>
```

```
l1 = ["a", "doublon", "b" , "c" , "doublon"]
setFromL1 = set(l1)
print("setFromL1",setFromL1,type(setFromL1))
# setFromL1 {'b', 'c', 'doublon', 'a'} <class 'set'>
```

```
tuple1 = ("a", "doublon", "b" , "c" , "doublon")
setFromTuple1 = set(tuple1)
print("setFromTuple1 =", setFromTuple1 ,type(setFromTuple1))
# setFromTuple1 = {'b', 'c', 'a', 'doublon'} <class 'set'>
```

3. Dictionnaires (associations)

```
dictionnaireVide={}
dictionnaire2 = dict()
```

Un dictionnaire python est une table d'association (Map) : (ensemble de couples (clef,valeur))
La syntaxe d'un dictionnaire python est très proche de JSON (javascript object notation) . Par

défaut, les clefs d'un dictionnaire python sont cependant entourées de simples quotes ('clefPython' plutôt que "clefJson") .

```
dictionnaireCouleurs={
        'red':"#ff0000",
        'green':"#00ff00",
        'blue':"#0000ff"
}

rgbRed=dictionnaireCouleurs["red"];
print("rgbRed=",rgbRed); # #ff0000
dictionnaireCouleurs["red"]="#FF0000"; # change value
print(dictionnaireCouleurs["red"]); # #FF0000

rgbGreen=dictionnaireCouleurs.get("green");
print("rgbGreen=",rgbGreen); # #00ff00
```

```
dictionnaireCouleurs["black"]="#000000"; # add new pair (key ,value )
print(dictionnaireCouleurs["black"]); # #000000

dictionnaireCouleurs.update({"white":"#FFFFFF"}); # add new {key : value } pair
print(dictionnaireCouleurs["white"]); # #FFFFFF

dictionnaireCouleurs.update({"white":"#ffffff" , "yellow" : "#ffff00"});
                        # fusionner autre dictionnaire

print(dictionnaireCouleurs["yellow"]); # #ffff00
```

```
del dictionnaireCouleurs["white"]; #ou bien dictionnaireCouleurs.pop("white")
print(dictionnaireCouleurs); # affichage après suppression de l'association "white"
# {'red': '#FF0000', 'green': '#00ff00', 'blue': '#0000ff', 'black': '#000000', 'yellow': '#ffff00'}
```

```
dicoPers= { "nom" : "Bon" , "age" : 45 }
print("nom=" + dicoPers.get("nom")); # Bon
print("nom=" + dicoPers.get("prenom","prenomParDefaut")); # prenomParDefaut
```

```
dicoPers= { "nom" : "Bon" , "age" : 45 }
for key in dicoPers :
        print(key)
# nom
# age
```

ou bien

```
dicoPers= { "nom" : "Bon" , "age" : 45 }
for clef in dicoPers.keys() :
        print(clef)
# nom
# age
```

```
for val in dicoPers.values() :
        print(val)
# Bon
# 45
```

```
for clef,val in dicoPers.items() :
        print(clef,val)
# nom Bon
# age 45
```

```
dicoPers= { "nom" : "Bon" , "age" : 45 }
print("nbAssociations=" , len(dicoPers)) # 2

dicoDuplique=dicoPers.copy(); #ou bien dicoDuplique=dict(dicoPers);
dicoDuplique["age"]=30
print(dicoPers) # { "nom" : "Bon" , "age" : 45 }
print(dicoDuplique) # { "nom" : "Bon" , "age" : 30 }

dicoDuplique.clear() # vide le contenu du dictionnaire
```

```
if "nom" in dicoPers :
        print("dicoPers comporte la clef nom")
else :
        print("dicoPers ne comporte pas la clef nom")
```

```
#NB: au sein d'un dictionnaire une valeur peut être
#une liste ou un dictionnaire imbriqué:
dicoPers={
 "nom" : "Bon" ,
 "age" : 45 ,
 "fou" : False ,
 "adresse" : {
      "rue" : "12 rue elle",
                "codePostal" : "75008",
                "ville" : "Paris"
                } ,
  "sports" : [ "vélo" , "foot" ]
 }
print("dicoPers très proche du format JSON:" , dicoPers);
```

4. Manipulation de chaînes de caractères

```
s1="Hello"
s2=' World'
s3=s1+s2 # concaténation
print(s3) # Hello World
```

```
age=33
message = "mon age est " + age #TypeError: can only concatenate str (not "int") to str
message = "mon age est " + str(age)
print("message=",message) # mon age est 33
```

NB: la fonction ***str***() effectue une conversion de type (quelquefois appelée casting)

```
s4='''ile de
france'''
print(s4) # affiche une chaîne multi-lignes
```

```
s5="Bonjour"
premierCaractere=s5[0]
print(premierCaractere) # B

dernierCaractere=s5[-1]
print(dernierCaractere) # r
```

```
s="abc"
for c in s :
        print("c=",c)
```

=> c= a c= b c= c

```
# [i,j] means .substring(i,j) included i and excluding j

troisPremiersCaracteres=s5[0:3] # s5[0:3] means s5[0:3[ !!!
print(troisPremiersCaracteres) # Bon

n=len(s5) # length of string (7)

troisDerniersCaracteres=s5[n-3:n] # s5[n-3:n] means s5[n-3:n[ !!!
print(troisDerniersCaracteres) # our
```

```
s6=" Ile De France "
s6Bis=s6.strip() #like trim of other language --> supprime espaces inutiles
            #au début ou à la fin
print(s6Bis) # "Ile De France"
```

```
s7="Mont Saint Michel"
s7Maj = s7.upper(); print(s7Maj) # MONT SAINT MICHEL
s7Min = s7.lower(); print(s7Min) # mont saint michel
```

```
s7="Mont Saint Michel"
s7Bis=s7.replace(' ','-') # replace substring with another string
print(s7Bis) # Mont-Saint-Michel
```

```
s8="partie1;partie2;partie3"
listeParties=s8.split(';')
print(listeParties) # ['partie1', 'partie2', 'partie3']
```

```
s9="un deux trois"
if "deux" in s9 :
        print("s9 comporte deux")
else :
        print("s9 ne comporte pas deux")

#il existe aussi le test if "deux" not in s9
```

```
nom="toto"
age=30
taille=1.80
# .format remplace {0} , {1} , {2} , ... par les 1er,2eme,3eme arguments
description="{0} a {1} an(s) et mesure {2} m".format(nom,age,taille)
print(description) # toto a 30 an(s) et mesure 1.8 m
```

Variantes pour .format() :

```
age=33
ville='Rouen'
#message= j'ai 33 ans et j'habite Rouen

message='j\'ai {} ans et j\'habite {}'.format(age , ville)
print("message=",message)

message='j\'ai {1} ans et j\'habite {0}'.format(ville , age)
print("message=",message)

message='j\'ai {age} ans et j\'habite {ville}'.format(age=age , ville=ville)
print("message=",message)
```

```
#caractères spéciaux : \n = new line , \t = tabulation , ...
#escape : \\ means \

s10="\tHello" ; print(s10); #      Hello
s11="surLigne1\nsurLigne2" ; print(s11);
# surLigne1
# surLigne2
```

```
s12="dupond";
s12Bis=s12.capitalize() # transforme première lettre en Majuscule
print(s12Bis); # Dupond
```

```
fileName="p2.py"
```

```
dotIndex = fileName.find(".")
# .index() retourne position de la chaine recherchée et erreur si pas trouvée
# .find() retourne position de la chaine recherchée et -1 si pas trouvée
print("position . =" , dotIndex) # 2
```

```
s13="phrase finissant par un point."
if s13.endswith(".") :
        print("s13 se termine par '.' ")
```

```
s14="123"
if s14.isdigit() :
        print("s14 ne comporte que des caractères numériques ")
```

Premier aperçu sur les dates (python) :

```
import datetime

d = datetime.datetime.now()
print(d) # exemple: 2020-05-03 00:18:51.608375
```

--> le module *datetime* sera approfondi ultérieurement

Premier aperçu sur le module math (python) :

```
import math

x=2
s= math.sin(x)
p= 2 * math.pi * x
y = math.pow(x,3) # 2 puissance 3 = 8
```

--> le module *math* sera approfondi ultérieurement

V - Fonctions, lambda , modules , exceptions

1. Fonctions (python)

Un **programme** bien **structuré** est généralement constitué de **blocs de code réutilisables** appelés "**fonctions**" .

Au sein du langage python :

- la définition d'une fonction est introduite par le mot clef **def** .
- une fonction a toujours des **parenthèses** (au sein de la définition et des appels)
- une fonction peut avoir (ou pas) des paramètres et une valeur calculée et retournée via le mot clef **return** .

1.1. Fonctions élémentaires

```
#fonction basique sans paramètre retournant une valeur fixe:
def genererMessageSalutation() :
        message="bonjour"
        return message
```

```
#fonction/procédure basique ne retournant aucune valeur
#mais exécutant une action:
def saluer():
        salutation=genererMessageSalutation() #appel de sous-fonction
        print(salutation)
```

```
saluer() #l'appel de la fonction déclenche l'affichage de bonjour
```

1.2. Fonctions avec paramètres

```
#fonction multiplier avec 2 paramètres formels a et b
def multiplier(a,b):
        return a*b
```

```
x=4; y=5;
res = multiplier(x,y); # appel de la fonction multiplier en passant
                       # les valeurs des paramètres effectifs x et y
                       # lors de l'appel a est une copie de x
                       # et b est une copie de y

print("res=",res) # affiche res=20
```

1.3. Lambda (fonction anonyme)

```
def carre(x):
        return x*x

print("carre(4)=", carre(4)) #16
```

```
lambdaCarre = lambda x : x*x
print("type(lambdaCarre)=", type(lambdaCarre)) #<class 'function'>
print("lambdaCarre(4)=", lambdaCarre(4)) #16
```

```
lambdaMult = lambda x,y : x*y
print("lambdaMult(2,3)=", lambdaMult(2,3)) #6
```

```
import datetime;
lambdaReturnNow = lambda  maintenant=datetime.datetime.now() : maintenant.time()
print("heure=" , lambdaReturnNow()) # 18:16:54.789809
```

```
# le paramètre fCalcul est une référence sur une fonction de calcul à déclencher
#  le paramètre fPrefixage est une référence sur une fonction de préfixage à déclencher
def enchainerCalculEtAffichageAvecPrefixage(x, fCalcul ,fPrefixage):
        res = fCalcul(x)
        print(fPrefixage(res))

enchainerCalculEtAffichageAvecPrefixage(6 ,
                lambda x:x*x ,lambda expr : '** ' + str(expr))          # ** 36

enchainerCalculEtAffichageAvecPrefixage(6 ,
                        lambda x:x+x ,lambda expr : '>> ' + str(expr))          # >> 12
```

1.4. Paramètres nommés et facultatifs sur fonctions

Les paramètres d'une fonction python sont en interne gérés comme un dictionnaire où chaque paramètre à un nom , une valeur (renseignée ou bien par défaut) .

```
#paramètres optionnels (selon = default_value )et nommés

def displayVal(val , color="blue" ,comment ="no comment"):
        message=str(val)+ " color="+color + " comment="+comment
        print(message)

displayVal(5,"red","with_all_params")
#5 color=red comment=with_all_params

displayVal(8,"green") #with default comment
#8 color=green comment=no comment

displayVal(7) # with default color and comment
#7 color=blue comment=no comment

displayVal(comment="with_named_params",val=9) #with named params
#9 color=blue comment=with_named_params
```

1.5. Fonction avec nombre d'arguments variables

```
####### fonction à nombre d'arguments variables (*args, **kwargs)
# l'étoile est appelée opérateur splat en python
# **kwargs for keyword args or **kvargs for keyValue args (c'est un double splat)

def displayArgs(*args):
        for a in args:
                print("a=",a)

def displayKeyValueArgs(**kvargs):
        for k,v in kvargs.items():
                print("k=",k,"v=",v)

def displayArgsAndKeywordArgs(*args,**kwargs):
        for a in args:
                print("a=",a)
        for k,v in kwargs.items():
                print("k=",k,"v=",v)


def sommeArgs(*args):
        s=0
        for a in args:
                s+=a
        return s

def sommeAnyArgs(*args,**kwargs):
        s=0
        for a in args:
                s+=a
        for v in kwargs.values():
                s+=v
        return s

displayArgs(2,6,8) # a=  , a= 6 , a= 8
print("sommeArgs=",sommeArgs(2,6,8)) # 16

displayKeyValueArgs(a=1,b=3,c=5)#k= a v= 1 , k= b v= 3 , k= c v= 5
displayArgsAndKeywordArgs(2,6,8,a=1,b=3,c=5) #ok
#displayArgsAndKeywordArgs(a=1,b=3,c=5,2,6,8) #error , positional args should be first

print("sommeAnyArgs=",sommeAnyArgs(2,6,8,a=1,b=3,c=5)) # 25
```

2. Modules et packages (python)

Dès qu'un programme comporte beaucoup de lignes de code, il est conseillé de **répartir les instructions dans plusieurs fichiers complémentaires** .
Certains fichiers (utilisés par d'autres) constitueront ainsi des modules de code prêts à être réutilisés.

2.1. importations de fonctions (et autres éléments)

my_fct.py

```
def doubleDe(x):
        return 2*x

def moitieDe(x):
        return x/2
```

my_app.py

```
# importation de toutes les fonctions du fichier my_fct.py
# pour pouvoir les appeler depuis ce fichier my_app.py
from my_fct import *

x=12;
print(doubleDe(x)) # affiche 24
print(moitieDe(x)) # affiche 6
```

ou bien (avec nom du module en préfixe)

my_app.py

```
import my_fct

# --> appels de my_fct.doubleDe(x) et my_fct.moitieDe(x)

x=12;
print(my_fct.doubleDe(x)) # affiche 24
print(my_fct.moitieDe(x)) # affiche 6
```

ou bien (avec préfixe renommé)

my_app.py

```
import my_fct as m

# --> appels de m.doubleDe(x) et m.moitieDe(x)

x=12;
print(m.doubleDe(x)) # affiche 24
print(m.moitieDe(x)) # affiche 6
```

2.2. packages de fichiers réutilisables (modules/librairies)

On range souvent dans un ***répertoire*** (ex : ***my_util***) un paquet de fichiers "python" réutilisables :

Exemple :

my_util/op3.py

```
def tripleDe(x):
        return 3*x

def tiersDe(x):
        return x/3
```

my_util/op4.py

```
def quatreFois(x):
        return 4*x

def quartDe(x):
        return x/4
```

my_app.py

```
#from my_util.op3 import *
from my_util.op3 import tripleDe
from my_util.op4 import *


x=12 ;
print(tripleDe(x)) # affiche 36
print(quartDe(x)) # affiche 3.0
```

2.3. Principaux modules prédéfinis du langage python

modules	***fonctionnalités***
random	nombres aléatoires et autres
math	fonctions mathématiques (sin, cos, ...)
os	fonctions pour accès uniformes aux fonctionnalités du système d'exploitation (windows, linux, ...)
datetime	classes pour manipuler les dates et les heures
sys	Accès à des variables du système (env , ...)
json	encodage/décodage au format json
...	

3. Calculs mathématiques élémentaires

Le module prédéfini ***math*** de python comporte quelques **fonctions mathématiques élémentaires** que l'on retrouve dans la plupart des autres langages de programmation (C, java, ...) .

3.1. Principales fonctions du Module math

fonction	fonctionnalité
math.**sin(x)**	sinus avec x en radians
math.**cos(x)**	cosinus avec x en radians
math.**tan(x)**	tangente avec x en radians
asin(x) , acos(x) , atan(x)	arc sinux , arc cosinus , arc tangante
math.**radians(x)**	convertit un angle de degrés en radians
math.**degrees(x)**	convertit un angle de radians en degrés
math.**sqrt(x)**	racine carrée
math.**pow(x,y)**	élève x à la puissance y
math.**exp**(x)	calcule **e** puissance x
math.**log**(x)	calcul ln(x)
math.**hypot**(x,y)	longueur de l'hypoténuse = sqrt(x*x+y*y)
math.**ceil**(x)	arrondi à l'entier au dessus : ceil(5.5) = 6
math.**floor**(x)	arrondi à l'entier en dessous : floor(5.5) = 5
math.**fabs**(x)	retourne la valeur absolue : fabs(-7) = 7
math.**gcd**(a,b)	retourne le pgcd : plus grand diviseur commun

Constantes : math.**pi** (3.141592....) et math.**e** (2.718281…)

Exemples :

```
from math import *
print("pi=",pi) # 3.141592653589793
print("e=",e) # 2.718281828459045
print("sin(pi/6)",sin(pi/6)) # 0.49999999999999
y=pow(2,3); print("2 puissance 3 = " , y); # 8
```

ou bien

```
import math
print("pi=",math.pi) # 3.141592653589793
print("e=",math.e) # 2.718281828459045
print("sin(pi/6)",math.sin(pi/6)) # 0.49999999999999
y=math.pow(2,3); print("2 puissance 3 = " , y); # 8
```

3.2. Exemple (résolution d'équation du second degré)

```
import math
#NB: math.sqrt() calcule la racine carrée (square root).

# resolution ax^2+bx+c=0
def resolEq2ndDegre(a,b,c):
        delta = b*b-4*a*c
        if delta==0 :
                x1=x2=-b/(2*a)
        if delta > 0 :
                x1=(-b-math.sqrt(delta))/(2*a)
                x2=(-b+math.sqrt(delta))/(2*a)
        if delta <0 :
                x1=(-b-1j*math.sqrt(-delta))/(2*a)
                x2=(-b+1j*math.sqrt(-delta))/(2*a)
        print("solutions pour equation ax^2+bx+c=0 avec a=",a, "b=",b , "c=" ,c );
        print("x1=",x1)
        print("x2=",x2)

resolEq2ndDegre(2,-9,-5); # x1=-0.5 et x2=5

resolEq2ndDegre(2,-1,-6); # x1=-1.5 et x2=2

resolEq2ndDegre(1,3,9/8); # x1=x2=4/4=0.75

resolEq2ndDegre(1,2,5); # x1=-1-2j et -1+2j avec j=i et j^2=i^2=-1
```

3.3. Module "random" pour nombres aléatoires

```
import random

x =random.random()       # Random float x, 0.0 <= x < 1.0
print(x) # x=0.3581652418510137 ou autre

x=random.uniform(1, 10)  # Random float x, 1.0 <= x < 10.0
print(x) # x=6.1800146073117523 ou autre

x=random.randint(1, 10)  # Integer from 1 to 10, endpoints included
print(x) # x=6 ou autre

import string
small_letters = string.ascii_lowercase # séquence des caractères de a à z
print(small_letters) # affiche abcdefghijklmnopqrstuvwxyz
c=random.choice( small_letters) # retourne un éléments de la séquence
                                # choisi aléatoirement : ici une lettre en a et z
print(c) # affiche c ou r ou autre

subList = random.sample([1, 2, 3, 4, 5],  3)  # Choose 3 elements
print(subList) # affiche [5, 2, 1] ou autre
```

3.4. Module datetime

Principales classes du module datetime :

datetime.**date**	Dates (year,month,day)
datetime.**time**	Heures (hour,minute,second,microsecond + tzinfo)
datetime.**datetime**	Date et heure
datetime.**timedelta**	Duration = différence de temps
datetime.**tzinfo** et .**timezone**	Gestion des décalage horaires

Exemple :

```
import datetime
date1 = datetime.datetime(2025,5,18)  # (year,month,day) 18 mai 2025
d=maintenant =  datetime.datetime.now()
print("date1=",date1) #2025-05-18 00:00:00
print("année=" , maintenant.year) # et .month , .day  , .hour, .minute , .second , ...
print("maintenant=",maintenant)  # 2025-05-21 09:47:53.340047 ou autre
```

Mise en forme de date via ***.strftime*** *:*

```
print("au format yyyy-mm-dd %Y-%m-%d : " , d.strftime("%Y-%m-%d")) # 2025-05-21
print("au format dd/mm/yyyy %d/%m/%Y : " , d.strftime("%d/%m/%Y")) # 21/05/2025

print("avec jour de la semaine : " , d.strftime("%A %Y-%m-%d")) # Wednesday 2025-05-21
print("avec %H:%M:%S : " , d.strftime("%Y-%m-%d %H:%M:%S")) # 2025-05-21 10:05:58
```

3.5. Module os

```
import os
cwd = os.getcwd()
print("Current working directory:", cwd) #  C:\tp\xyz

userHomeDirectory = os.path.expanduser("~")
print("userHomeDirectory:", userHomeDirectory) #  C:\Users\toto
```

```
try:
    os.mkdir("mon_sous_repertoire") # créer un sous répertoire (si droits suffisants)
    print(os.listdir()) # affiche la liste du contenu du répertoire courant
    os.rename("mon_sous_repertoire" , "my_sub_dir") # renommer un fichier ou répertoire (si droits suffisants)
    os.remove("my_sub_dir")# supprimer un fichier ou répertoire (si droits suffisants)
except Exception as e:
    print("une exception a eu lieu:",e)
```

Doc de référence : https://docs.python.org/fr/3.7/library/os.html

Attention : le module os fourni des opérations de très bas niveaux , il est souvent nécessaire de s'appuyer sur des modules/packages utilitaires de plus haut niveau pour bien manipuler le système de fichiers.

3.6. Module sys

```
import sys
print("version de python=",sys.version) # ex 3.13.3 (...)
print("system platform=",sys.platform) # ex: win32
print("path=",sys.path) # seulement partie accessible depuis python
sys.exit(0) #fin du process en retournant le code 0
```

4. Fonctions natives et lambdas

input() , print() , …

4.1. f-string (depuis python 3.6)

Au sein des ***formatted-string*** de ***python ≥ 3.6*** , la syntaxe spéciale **{variable}** est automatiquement remplacée par la valeur de la variable (même principe que les template-string de javascript)

```
# f-string depuis python 3.6
nom="toto"
age=30
taille=1.80
message = f'{nom} a {age} ans et mesure {taille} m'
print("message=",message) # toto a 30 ans et mesure 1.8 m

pers = { 'nom' : 'titi' , 'age' : 40 , 'taille': 1.66}
message = f'{pers["nom"]} a {pers["age"]} ans et mesure {pers["taille"]} m'
print("message=",message) # titi a 40 ans et mesure 1.66 m
```

4.2. tris en python 3

```
nombres = [34, 7, 12, 6, 89]
print("nombres=",nombres)
nombres.sort()
print("après tri, nombres=",nombres)  # [6, 7, 12, 34, 89]

nombres.sort(reverse=True)
print("après tri décroissant, nombres=",nombres)  # [89, 34, 12, 7, 6]

liste =  [34, 8, 15, 6, 67]
liste_triee = sorted(liste) #créer une nouvelle liste triée , option possible: reverse=True
print("liste=",liste)
print("liste_triee=",liste_triee) # [6, 8, 15, 34, 67]

liste = [ 'France' , 'Allemagne' , 'Suede' , 'Espagne' , 'Italie']
print("liste=",liste)
liste_triee = sorted(liste)
print("liste_triee=",liste_triee)
#liste_triee= ['Allemagne', 'Espagne', 'France', 'Italie', 'Suede']
```

```
listePers = [
    { 'nom' : 'Dupond' , 'age' : 23 },
    { 'nom' : 'Zorro' , 'age' : 44 },
    { 'nom' : 'Anatole' , 'age' : 66 },
    { 'nom' : 'Laurent' , 'age' : 25 }
]
print("listePers=",listePers)
listePers_triee_par_noms = sorted(listePers, key = lambda p : p['nom'])
print("listePers_triee_par_noms=",listePers_triee_par_noms)
listePers_triee_par_ages = sorted(listePers, key = lambda p : p['age'])
print("listePers_triee_par_ages=",listePers_triee_par_ages)
```

NB : l'ancienne version 2 de python effectuait des tris avec un ancien paramètre cmp correspondant à une fonction de comparaison (comme ce qui ce fait dans beaucoup d'autres langages tels que java, javascript, …)

Depuis la version 3 de python, la fonction sorted comporte un paramètre optionnel key permettant d'extraire si besoin la sous partie à triée et sur celle-ci , sont appelées automatiquement des méthodes de comparaisons de type _lt_ , _gt_ , _eq_ , _le_ , _ge_ , ne_ .

4.3. Filtrages et transformations

#Filtrages avec filter() et lambda :

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
nombres_pairs = list(filter(lambda x: x % 2 == 0, numbers))
# list() indispensable autour de filter() , sinon simple filter_object
print("nombres_pairs=",nombres_pairs) #  [2, 4, 6, 8]

nombres_plus_grands_que_4 = list(filter(lambda x: x>4 , numbers))
print("nombres_plus_grands_que_4=",nombres_plus_grands_que_4) # [5, 6, 7, 8]
```

#Transformations/mappings avec map() et lambda :

```
fruits = ['pomme', 'orange', 'cerise' , 'poire']
print("liste de fruits=",fruits)
liste_lengths = list(map(lambda x: len(x), fruits))
# list() ou autre indispensable autour de map() , sinon simple map_object
print("de longueurs=",liste_lengths) #  [5, 6, 6, 5]
```

 # reduce et lambda :

```
from functools import reduce
numbers = [5, 6, 2, 7]
print("numbers=",numbers)
total = reduce(lambda x, y: x + y, numbers)
print(f'The sum of the numbers is {total}.') #20
```

5. Gestion des exceptions (python)

NB : En python comme dans la plupart des autres langages de programmation, une division entre 2 nombres entiers provoque une erreur/exception dans le cas d'une ***division par zéro*** *.*

Attention : python soulève l'exception ***ZeroDivisionError: float division by zero*** *lors d'une division entre "float" (x / 0.0) , là où d'autres langages tels que java ou c++ retourne "NaN" (not a number)*

5.1. plantage du programme sans traitement d'exception

exceptions.py

```
a=5
b=0
c=a/b
print(c)
```

==>

```
Traceback (most recent call last):
  File "exceptions.py", line 3, in <module>
    c=a/b
ZeroDivisionError: division by zero
```

5.2. avec traitement des exceptions (try / except)

```
a=6
#b=2
b=0
try :
        c=a/b
        print("res division=" , c)
except :
        print("attention: une erreur s'est produite !!!")

print("suite du programme qui ne s'est pas planté")
```

avec b=2 :

```
res division= 3.0
suite du programme qui ne s'est pas planté
```

avec b=0 :

```
attention: une erreur s'est produite !!!
suite du programme qui ne s'est pas planté
```

5.3. Lever une exception personnalisée (raise Exception)

```
def myDivision(x,y):
        if y==0 :
                raise Exception("division par zéro invalide")
                    else :
                return x/y
```

```
a=6
#b=2
b=0
```

```
try :
        c=myDivision(a,b);
        print("res myDivision=" , c)
except Exception as e :
   print("une erreur a eu lieu :" , e)  # affiche une erreur a eu lieu : division par zéro invalide
```

5.4. Syntaxe facultativement complète (try / except / finally)

try :
        instructions susceptibles de lever une exception
except :
        instructions en cas d'exception
else :
        instructions après si pas d'exception
finally :
        instructions après dans tous les cas

VI - Gestion des fichiers (depuis python)

1. Gestion des fichiers en python

1.1. ouverture , lecture et écritures

```
'''
principaux modes d'ouverture:
        r :  read
        w : write / ré-écriture (écrasement)
        a : append (ajout à la fin)

le fichier est souvent créé en écriture s'il n'existe pas

modes secondaires d'ouverture :
        b : binaire (ex: images , videos, ...)
        t : texte
'''

f= open("data.txt","wt")
print("f=",f); # affiche le descripteur de fichier ouvert, par exemple :
#f= <_io.TextIOWrapper name='data.txt' mode='wt' encoding='cp1252'>
f.close() # fermeture du fichier
```

```
#ouvrir un nouveau fichier et écrire 2 lignes dedans:
f= open("data.txt","wt")
f.write("ligne1\n")
f.write("ligne2\n")
f.close();
```

```
#ré-ouvrir un fichier existant et ajouter 2 lignes dedans:
f= open("data.txt","at")
f.write("ligne3\n")
f.write("ligne4\n")
f.close();
```

```
#ré-ouvrir un fichier existant et charger son contenu d'un seul coup:
f= open("data.txt","rt")
toutLeContenu=f.read(); print(toutLeContenu);
f.close();
```

```
#ré-ouvrir un fichier existant et lire son contenu ligne par ligne via .readline() et while
f= open("data.txt","rt")
ligneLue="?"
while ligneLue :
        ligneLue=f.readline()
        if ligneLue.endswith("\n") :
```

```
                ligneLue=ligneLue[:-1]  # enlever le dernier caractère
        print(ligneLue);
f.close();
```

```
#ré-ouvrir un fichier existant et lire son contenu ligne par ligne via boucle for :
f= open("data.txt","rt")
for ligneLue in f :
        if ligneLue.endswith("\n") :
                ligneLue=ligneLue[:-1]
        print(ligneLue);
f.close();
```

1.2. avec fermeture automatique (with)

```
###### with keyword for automatic closing (as in try/except/FINALLY) ####

with open("data2.txt","wt") as f :
        f.write("ligne1\n")
        f.write("ligne2\n")
        # automatic f.close() even in case of exception
```

1.3. noms de fichiers, répertoires , chemins

...

2. Formats de fichiers et gestion en python

2.1. gestion du format json

JSON signifiant **JavaScript Object Notation** est un format de données très classique utilisé pour :
- **paramétrer des configurations**
- **structurer des données échangées** (documents , appels de WS REST, ...)

La structure et la syntaxe du format json est très proche de celle d'un dictionnaire python.

Principales équivalences :

Python	JSON
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	null

La bibliothèque prédéfinie **json** (à importer via ***import json***) comporte essentiellement les méthodes :

- **dumps**() permettant de transformer des données python en chaîne de caractères json .
- **loads**() permettant d'analyser une chaîne de caractères json et d'effectuer une conversion en python

Exemple d'écriture au format json dans un fichier :

```
import json

#personne1 en tant que dictionnaire python :
personne1={
  "nom" : "Bon" ,
  "age" : 45 ,
  "fou" : False,
  "adresse" : {
        "rue" : "12 rue elle",
                    "codePostal" : "75008",
                    "ville" : "Paris"
                    } ,
   "sports" : [ "velo" , "foot" ]
 }

#p1AsJsonString = json.dumps(personne1);
p1AsJsonString = json.dumps(personne1,indent=4);
print("p1AsJsonString=",p1AsJsonString)
with open("p1.json","wt") as f :
        f.write(p1AsJsonString)
```

contenu du fichier ***p1.json*** généré :

```
{
   "nom": "Bon",
   "age": 45,
   "fou": false,
   "adresse": {
      "rue": "12 rue elle",
      "codePostal": "75008",
      "ville": "Paris"
   },
   "sports": [
      "velo",
      "foot"
   ]
}
```

Exemple de lecture d'un fichier au format json:

```
#relecture du fichier p1.json et extraction du contenu en données python:
import json

with open("p1.json","rt") as f :
        fileContentAsJsonString=f.read()
        pers = json.loads(fileContentAsJsonString)
        print("pers=",pers);
        print("type(pers)=",type(pers));
```

-->

```
pers= {'nom': 'Bon', 'age': 45, 'fou': False, 'adresse': {'rue': '12 rue elle', 'codePostal': '75008', 'ville': 'Paris'}, 'sports': ['velo', 'foot']}
type(pers)= <class 'dict'>
```

2.2. gestion du format xml

...

VII - Python orienté objet

1. Programmation orientée objet en python

1.1. Classe et instances

En langage python, pas de mof clef **~~this~~** mais le mot **self** signifiant "objet courant de la classe" (qui sera spécifié depuis du code extérieur à la classe via un préfixe de type ***obj.***)

En langage python, la fonction constructeur (initialisant les valeurs interne de l'objet) a le nom spécial **__init__** .

Le mot clef ***del*** (signifiant delete) sert à déclencher facultativement une destruction d'objet .

Exemple :

objets.py

```
import math

############ code de la classe Cercle en python :

class Cercle() :

    #constructeur avec valeurs par défaut:
        def __init__(self,xc=0,yc=0,rayon=0) :
                self.xc=xc
                self.yc=yc
                self.rayon=rayon

        #méthode spéciale __str__ (équivalent à .toString() de java)
        #qui sera automatiquement appelée lors d'un print(cercle):
        def __str__(self) :
                return "Cercle(xc="+str(self.xc) +",yc="+str(self.yc)+",rayon="+str(self.rayon)+ ")"

        def perimetre(self) :
                return 2*math.pi*self.rayon

        def aire(self) :
                return math.pi*self.rayon*self.rayon

###### utilisation de la classe Cercle


c1=Cercle(); #instanciation (pas de mot clef new) mais nom de classe
             #vue comme fonction créant une nouvelle instance
c1.rayon=40;
print("rayon de c1=",c1.rayon) # rayon de c1= 40
print("perimetre de c1=",c1.perimetre()) # perimetre de c1= 251.32741228718345
print("surface de c1=",c1.aire()) # surface de c1= 5026.548245743669
```

```
c2=Cercle(40,60,20) # Cercle(xc,yc,rayon)
print("rayon de c2=",c2.rayon) # rayon de c2= 20

print("c2=" , c2) # équivalent à print("c2=" , str(c2))
# affiche c2= Cercle(xc=40,yc=60,rayon=20)
```

Récupération des valeurs de l'instance sous forme de dictionnaire python :

```
#suite de l'exemple précédent (où c2 est une instance de la classe Cercle)
print("type(c2)=",type(c2)) # <class '__main__.Cercle'>
c2AsDict = vars(c2) # converti un objet en un dictionnaire  (autre solution = c2.__dict__ )
print("c2AsDict=",c2AsDict) # {'xc': 40, 'yc': 60, 'rayon': 20}
print("type(c2AsDict)=",type(c2AsDict)) #  <class 'dict'>
```

NB:

- pas de mot clef ~~public, private , protected~~ en python .
- En python, tout est par défaut public
- On peut tout de même utiliser la convention de nommage **_protectedAttribute** et **__privateAttribute** (avec un double underscore en préfixe) . Ce n'est qu'une convention. Seul l'ajout d'éventuels décorateurs peuvent assurer une parfaite restriction/encapsulation .

1.2. Héritage simple/ordinaire

code de la classe Figure parente en python :

```
class Figure:

    #constructeur avec valeurs par défaut:
    def __init__(self,x=0,y=0,color="black"):
        self.x=x
        self.y=y
        self.color=color

            #méthode spéciale __str__ (équivalent à .toString() de java)
            #qui sera automatiquement appelée lors d'un print(cercle):
    def __str__(self):
        return f"Figure(x={self.x} ,y={self.y} ,color={self.color})"

    def perimetre(self):
        return 0

    def aire(self):
        return 0

    def deplacer(self,dx,dy):
        self.x=self.x+dx
        self.y=self.y+dy

    def afficher(self):
```

```
    print(f"Figure(x={self.x} ,y={self.y} ,color={self.color})")
```

Code de la classe Cercle héritant de Figure :

```
import math
class Cercle(Figure):

    #constructeur avec valeurs par défaut:
    def __init__(self,xc=0,yc=0,rayon=0,color="black"):
        super().__init__(xc,yc,color) #ok python 3
        self.rayon=rayon

    def __str__(self):
        return f"Cercle xc={self.x} ,yc={self.y} ,rayon={self.rayon} ,color={self.color})"

    def perimetre(self):
        return 2*math.pi*self.rayon

    def aire(self):
        return math.pi*self.rayon*self.rayon

    def afficher(self):
        print(f"Cercle(rayon={self.rayon})", end =" heritant de ")
        super().afficher()
```

Code de la classe Rectangle héritant de Figure :

```
class Rectangle(Figure):

    #constructeur avec valeurs par défaut:
    def __init__(self,x=0,y=0,largeur=0,hauteur=0,color="black"):
        #self.x=x; self.y=y; self.color=color #not advised
        #Figure.__init__(self,x,y,color) # python 2 ou 3
        super().__init__(x,y,color) # python 3
        self.largeur=largeur
        self.hauteur=hauteur

    def __str__(self):
        return f"Rectangle (x={self.x} ,y={self.y} ,largeur={self.largeur} ,hauteur={self.hauteur} ,color={self.color})"

    def perimetre(self):
        return 2*(self.largeur + self.hauteur)

    def aire(self):
        return self.largeur*self.hauteur

    def afficher(self):
        print(f"Rectangle(largeur={self.largeur} ,hauteur={self.hauteur})" , end =" heritant de ")
        super().afficher()
```

```
#utilisation des classes Figure , Cercle et Rectangle

f1=Figure()
f2=Figure(50,50,"red"); f2.deplacer(10,10)
print("f1=",f1) # f1= Figure(x=0 ,y=0 ,color=black)
print("f2=",f2) # f2= Figure(x=60 ,y=60 ,color=red)
f2.afficher()      # Figure(x=60 ,y=60 ,color=red)

c1=Cercle() ;c1.rayon=40 ; print("c1=",c1)
print("rayon de c1=",c1.rayon) # rayon de c1= 40
print("perimetre de c1=",c1.perimetre()) # perimetre de c1= 251.32741228718345
print("surface de c1=",c1.aire()) # surface de c1= 5026.548245743669

c2=Cercle(40,60,20) # Cercle(xc,yc,rayon)
print("rayon de c2=",c2.rayon) # rayon de c2= 20
c2.afficher() # Cercle(rayon=20) heritant de Figure(x=40 ,y=60 ,color=black)

c2.deplacer(10,30) ;print("apres c2.deplacer(10,30) c2=" , c2)
# c2= Cercle xc=50 ,yc=90 ,rayon=20 ,color=black)
r1=Rectangle(15,15,200,150)
print("largeur de r1=",r1.largeur)
print("r1=",r1)
r1.deplacer(10,30) ;print("apres r1.deplacer(10,30) r1=" , r1)
#r1= Rectangle (x=25 ,y=45 ,largeur=200 ,hauteur=150 ,color=black )
r1.afficher() #Rectangle(largeur=200 ,hauteur=150) heritant de Figure(x=25 ,y=45 ,color=black)
print("perimetre de r1=",r1.perimetre()) ; # 700
print("surface de r1=",r1.aire()) #30000
print("issubclass(Cercle,Figure):" , issubclass(Cercle,Figure)) # True
print("issubclass(Cercle,Rectangle):" , issubclass(Cercle,Rectangle)) # False
```

1.3. Héritage multiple (rare)

```
class Avion:
   def __init__(self,altitude=0):
```

```
        self.altitude=altitude

    def voler(self):
        print("volant à l'altitude="+str(self.altitude))
```

```
class Flottant:
    def __init__(self,nbFlotteurs=2):
        self.nbFlotteurs=nbFlotteurs

    def pouvantFlotter(self) :
        print("pour flotter avec nbFlotteurs="+str(self.nbFlotteurs))
```

```
class Hydravion(Avion,Flottant):
    def __init__(self,nbFlotteurs=2):
        Avion.__init__(self,0); #altitude initiale=0
        Flottant.__init__(self,nbFlotteurs)

h1 = Hydravion()
h1.altitude=1250
h1.voler() #  volant à l'altitude=1250
h1.pouvantFlotter() # pouvant flotter avec nbFlotteurs=2
```

1.4. polymorphisme

Polymorphisme en boucle :

```
listeFigures = []
#listeFigures.append(c1)
listeFigures.append(Cercle(100,150,50,"red"))
#listeFigures.append(r1)
listeFigures.append(Rectangle(200,200,60,70,"orange"))
for f in listeFigures:
    f.deplacer(3,2)
    print("\nPour f de type=", type(f))
    f.afficher()
    print("isinstance(f,Cercle):" ,  isinstance(f,Cercle))
    print("isinstance(f,Rectangle):" ,  isinstance(f,Rectangle))
    print("isinstance(f,Figure):" ,  isinstance(f,Figure)) # True
    print("------------------------------")
```

```
Pour f de type= <class '__main__.Cercle'>
Cercle(rayon=50) heritant de Figure(x=103 ,y=152 ,color=red)
isinstance(f,Cercle): True
isinstance(f,Rectangle): False
isinstance(f,Figure): True
------------------------------

Pour f de type= <class '__main__.Rectangle'>
Rectangle(largeur=60 ,hauteur=70) heritant de Figure(x=203 ,y=202 ,color=orange)
isinstance(f,Cercle): False
```

```
isinstance(f,Rectangle): True
isinstance(f,Figure): True
------------------------------
```

1.5. classe avec attribut/variable de classe

avec attribut/variable de classe (proche du mot clef static de c++/java)

```
class CompteEpargne :
    tauxInteret = 1.25 # valeur par défaut liée à l'ensemble de la classe
                       # variable de classe (pas liée à self)
    def __init__(self,num,label,solde):
        self.num=num
        self.label=label
        self.solde=solde

    def __str__(self):
        return f'CompteEpagne num={self.num} label={self.label} solde={self.solde} avec tauxInteret = {CompteEpargne.tauxInteret}'
```

La variable de classe **CompteEpargne.tauxInteret** est **partagée** entre toutes les instances de la classe CompteEpargne . Un changement impacte toutes les instances .

```
ce1 = CompteEpargne(1,"compteEpargne1",50.0)
ce2 = CompteEpargne(2,"compteEpargne2",100.0)
print("ce1,",ce1); print("ce2,",ce2)
CompteEpargne.tauxInteret=2.1
print("ce1,",ce1); print("ce2,",ce2)
```

⇒

ce1, CompteEpagne num=1 label=compteEpargne1 solde=50.0 avec tauxInteret = **1.25**
ce2, CompteEpagne num=2 label=compteEpargne2 solde=100.0 avec tauxInteret = 1.25
ce1, CompteEpagne num=1 label=compteEpargne1 solde=50.0 avec tauxInteret = **2.1**
ce2, CompteEpagne num=2 label=compteEpargne2 solde=100.0 avec tauxInteret = **2.1**

1.6. classe abstraite (pas directement instanciable)

Code de la classe abstraite AnimalDomestique en python :

```
from abc import ABC, abstractmethod

#NB: ABC signifie AbstractBaseClass
class AnimalDomestique(ABC):

    #constructeur avec valeurs par défaut:
    def __init__(self,nom=""):
        self.nom=nom

    def __str__(self):
        return f"AnimalDomestique(nom={self.nom})"

    def decrire(self):
        print("AnimalDomestique de nom=",self.nom)
```

```
    @abstractmethod
    def parler(self):
        pass
```

```
#classe Chat héritant de AnimalDomestique:

class Chat(AnimalDomestique):

    #constructeur avec valeurs par défaut:
    def __init__(self,nom="ChatChat",nbHeuresSommeil=14):
        super().__init__(nom)
        self.nbHeuresSommeil=nbHeuresSommeil

    def __str__(self):
        return f"Chat(nom={self.nom} ,nbHeuresSommeil={self.nbHeuresSommeil} )"

    def decrire(self):
        print("Je suis un chat qui dort ",self.nbHeuresSommeil, " h")
        super().decrire()

    def parler(self):
        print("miaou miaou")

    def ronronner(self):
        print("ronron ...")
```

```
#classe Chien héritant de AnimalDomestique:

class Chien(AnimalDomestique):

    #constructeur avec valeurs par défaut:
    def __init__(self,nom="ChienChien",fonction="?"):
        super().__init__(nom)
        self.fonction=fonction

    def __str__(self):
        return f"chien(nom={self.nom} ,fonction={self.fonction} )"

    def decrire(self):
        print("Je suis un chien , fonction= ",self.fonction)
        super().decrire()

    def parler(self):
        print("whaouf whaouf")

    def monterLaGarde(self):
        print("je monte la garde ...")
```

```
#utilisation de la classe AnimalDomestique et de ses sous classes

'''
```

```
#partie impossible si la classe AnimalDomestique est abstraite
a=AnimalDomestique() #impossible d'instancier une classe abstraite
a.nom="animal_domestique_inconnu"
a.decrire()
a.parler()
'''

chat1 = Chat( "malo" , 15)
print(chat1) # Chat(nom=malo ,nbHeuresSommeil=15 )
chat1.decrire() # Je suis un chat qui dort  15  h   AnimalDomestique de nom= malo
chat1.parler() # miaou miaou
chat1.ronronner() # ronron ...
print("chat1_as_dict:" , vars(chat1))

chien1 = Chien( "medor" , "gardien de troupeau" )
print(chien1) # chien(nom=medor ,fonction=gardien de troupeau )
chien1.decrire() # Je suis un chien , fonction=  gardien de troupeau   AnimalDomestique de nom= medor
chien1.parler() # whaouf whaouf
chien1.monterLaGarde() # je monte la garde ...
print("chien1_as_dict:" , vars(chien1))

#polymorphisme en boucle :
listeAnimaux = []
listeAnimaux.append(chat1)
listeAnimaux.append(chien1)
for a in listeAnimaux:
    print("\n pour a de type=", type(a))
    a.decrire()
    a.parler()
    '''
    #mais surtout pas :
    if isinstance(a,Chat):
        a.decrire_chat();
        a.miauler()
    else:
        a.decrire_chien();
        a.aboyer()
    '''

print("\n")
print("isinstance(chat1,Chat):" ,  isinstance(chat1,Chat)) #True
print("issubclass(Chat,AnimalDomestique):" ,  issubclass(Chat,AnimalDomestique)) #True
```

1.7. avec @staticmethod

```
class MyBasicUtils:
    @staticmethod
    def addition(a, b):
        return a + b
```

```
# Appel direct sans créer d'instance (et/mais sans aucun accès possible à la classe ni à l'instance)
print("MyBasicUtils.addition(3, 5)=", MyBasicUtils.addition(3, 5))  #  8
```

1.8. avec @classmethod

```
class CompteAvecTauxInteret:
    tauxInteret = 1.25 #partagé au niveau classe , avec valeur par défaut

    def __init__(self,name):
        self.name=name

    def __str__(self):
        return f'CompteAvecTauxInteret name={self.name} avec tauxInteret={CompteAvecTauxInteret.tauxInteret}'

    @classmethod # pour utilisation de cls. à la place de .self
    def augmenterTauxInteret(cls,augmentation):
        # cls. permet d'accéder à la classe courante (iciCompteAvecTauxInteret)
        cls.tauxInteret += augmentation
```

```
# Utilisation de la méthode de classe
c1 = CompteAvecTauxInteret("c1")
c2 = CompteAvecTauxInteret("c2")
print(c1);print(c2)
CompteAvecTauxInteret.augmenterTauxInteret(0.5) # + 0.5%
print(c1);print(c2)
```

⇒

*CompteAvecTauxInteret name=c1 avec tauxInteret=**1.25***

CompteAvecTauxInteret name=c2 avec tauxInteret=1.25

*CompteAvecTauxInteret name=c1 avec tauxInteret=**1.75***

CompteAvecTauxInteret name=c2 avec tauxInteret=1.75

1.9. avec @property et .setter (encapsulation)

```
class Personne :

    def __init__(self,nom,taille):
        self.nom=nom
        self.__taille=taille if taille >=0 else 0

    def __str__(self):
        return f'Personne nom={self.nom} taille={self.__taille}'
```

```
@property
def taille(self):
    return self.__taille

@taille.setter
def taille(self, nouvelle_taille):
    if nouvelle_taille < 0 :
        raise ValueError(f"la taille ne peut pas etre négative; nouvelle_taille={nouvelle_taille} invalide")
    else :
        self.__taille = nouvelle_taille
```

NB:

- Grâce aux décorateurs @property et à @taille.setter la variable d'instance interne self.__taille (considérée comme privée) se voit indirectement récupérée et mise à jour via les deux versions (getter et setter) de la méthode taille .
- Le code interne du setter de taille n'accepte que les nouvelles tailles positives. En d'erreur de validation on soulève une exception de type ValueError .
- De l'extérieur , la taille est virtuellement vue comme si on avait affaire à un attribut public (même comportement qu'au niveau des langages c# et typescript) .

```
p = Personne("toto",170)
print(p, "p.taille=", p.taille)
try:
    p.taille = p.taille - 220
except ValueError as e:
    print(e)
print(p)
p.taille = 180
print(p)
```

⇒

Personne nom=toto taille=170 p.taille= 170

la taille ne peut pas etre négative; nouvelle_taille=-50 invalide

Personne nom=toto taille=170

Personne nom=toto taille=180

VIII - Gestion des modules/packages (pip)

1. PIP & PipEnv

pip = **p**ackage **i**nstaller for **p**ython

Le site web ***https://pypi.org/*** correspond à l'***index des packages "python"*** téléchargeables .

- Par défaut , **pip** fonctionne à un niveau global (les paquets téléchargés via pip sont utilisables par tous les utilisateurs d'un ordinateur et dans tous les projets python .)
- L'option **--user** de pip permet de déclencher un téléchargement/installation qui sera circonscrit à l'utilisateur courant .

pip est normalement déjà installé par défaut :

python --version
Python 3.8.2

pip --version
pip 19.2.3 from c:\users\didier\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)

Utilisation de pip :

pip install nomPackage *ou bien* **python -m pip install** nomPackage

pip **uninstall** nomPckage
pip **list**
pip **show** nomPackage

NB : en lançant **python -m pip** plutôt que **pip** on peut **plus choisir l'interpréteur python** (version a ou b) .

Les différentes applications "python" n'ont pas toutes les mêmes besoins (en termes de librairies et en termes de versions) .

venv = **v**irtual **env** .

Un *environnement virtuel* est un environnement Python semi-isolé qui autorise les paquets à être installés pour une application particulière, plutôt que d'être installés au niveau (global) du système .

- L'utilisation de **venv** est fortement recommandée depuis la version 3.5 de python.
- **venv** remplace maintenant les anciens *virtualenv* et *pyenv* devenus *obsolètes* .

---

pipenv (installable via pip) permet de combiner les fonctionnalités de **pip** et **venv** , avec une logique "par projet" , un peu comme *npm* en *javascript* ou *maven* en *java* .

installation de pipenv :

python -m pip install --user pipenv
...downloading...install...

python -m pipenv --version

pipenv, version 2020.11.15

ou bien *pipvenv --version* (en ajoutant C:\Users\didier\AppData\Roaming\Python\Python38\Scripts dans le PATH) .

Utilisation de **pipenv** (exemples):

load_packages.bat

```
cd /d %~dp0
python -m pipenv install numpy
python -m pipenv install matplotlib
pause
```

...téléchargement & installation... , Mise à jour des fichiers ***Pipfile*** et ***Pipfile.lock***

python -m pipenv shell

Launching subshell in virtual environment...

Microsoft Windows [version 10.0.19041.804]

(c) 2020 Microsoft Corporation. Tous droits réservés.

*(with_numpy-GWv6188t) D:\tp\tp_python\with_numpy>**pip list***

```
Package         Version
--------------- -------
cycler          0.10.0
kiwisolver      1.3.1
matplotlib      3.3.4
numpy           1.20.1
Pillow          8.1.2
pip             21.0.1
pyparsing       2.4.7
python-dateutil 2.8.1
setuptools      52.0.0
six             1.15.0
wheel           0.36.2
```

*(with_numpy-GWv6188t) D:\tp\tp_python\with_numpy>**exit***

python -m pipenv run ...

IX - Décorateurs, surcharge opérateurs, ...

1. Aspects divers et avancés de python

1.1. Surcharge d'opérateurs

Opérateurs + et * déjà surchargés sur nombre et chaîne de caractères :

```
print(1 + 2) # addition → 3
print("debut_"+"suite")  # concatenate two strings →  debut_suite

print(3 * 4) # Product two numbers → 12
print("Abc"*4) # Repeat the  string → "AbcAbcAbcAbc"
```

```
class MyBasicArray:
    #constructor:
    def __init__(self,*args):
        self.values = list(args)

    def __str__(self):
        sA="["
        for a in self.values:
            sA += ( str(a)+ ",")
        return sA[:-1] + "]"

    # adding two arrays (self and other)
    def __add__(self, other):
        resArray= MyBasicArray()
        for i in range(len(self.values)):
            v1=self.values[i]
            v2=other.values[i]
            resArray.values.append(v1+v2)
        return resArray

a1 = MyBasicArray(2,4,8); print("a1",a1) # [2,4,8]
a2 = MyBasicArray(6,1,7); print("a2",a2) # [6,1,7]
a3=a1+a2 # operator + between two instances of MyBasicArray ( __add__(self,other) )
print("a3=a1+a2=",a3) #[8,5,15]
```

Binary Operators :

Operator	Magic Method
+	**__add__(self, other)**
-	**__sub__(self, other)**
*	**__mul__(self, other)**
/	**__truediv__(self, other)**
//	**__floordiv__(self, other)**
%	**__mod__(self, other)**
**	**__pow__(self, other)**
>>	**__rshift__(self, other)**
<<	**__lshift__(self, other)**
&	**__and__(self, other)**
|	**__or__(self, other)**
^	**__xor__(self, other)**

Comparison Operators:

Operator	Magic Method
<	**__lt__(self, other)**
>	**__gt__(self, other)**
<=	**__le__(self, other)**
>=	**__ge__(self, other)**
==	**__eq__(self, other)**
!=	**__ne__(self, other)**

Assignment Operators:

Operator	Magic Method
-=	__isub__(self, other)
+=	__iadd__(self, other)
*=	__imul__(self, other)
/=	__idiv__(self, other)
//=	__ifloordiv__(self, other)
%=	__imod__(self, other)
**=	__ipow__(self, other)
>>=	__irshift__(self, other)
<<=	__ilshift__(self, other)
&=	__iand__(self, other)
|=	__ior__(self, other)
^=	__ixor__(self, other)

Unary Operators:

Operator	Magic Method
-	__neg__(self)
+	__pos__(self)
~	__invert__(self)

1.2. Liste en comprehension

Syntaxes compactes/astucieuses :

```
new_list = [ functionCall(item) for item in list ]
new_list = [ item for item in list if someCondition(item)  ]
new_list = [ functionCall(item) for item in list if someCondition(item) ]
```

Exemples :

```
numbersAsStrings = [ "2" , "-1" , "45" , "6"]
numbers = [ int(sn) for sn in numbersAsStrings ]
print("numbers=",numbers) # [2, -1, 45, 6]
```

```
values = [ -1 , 3 , -6 , 7 ,-8 , 5 , -12 , 13]
positivesValues=[ v for v in values if v >=0 ]
print("positivesValues=",positivesValues) # [3, 7, 5, 13]
```

```
import math
values = [ -1 , 3 , -6 , 4 ,-8 , 5 , -12 , 2]
positivesValuesWithPowers=[ (v,v*v,math.pow(v,3)) for v in values if v >=0 ]
print("positivesValuesWithSquare=",positivesValuesWithPowers)
# [(3, 9, 27.0), (4, 16, 64.0), (5, 25, 125.0), (2, 4, 8.0)]
```

1.3. décorateurs

Principe de fonctionnement d'un décorateur :

Un décorateur peut être vu comme une fonction technique qui admet en entrée une référence de fonction et qui renvoie en retour une fonction améliorée/modifiée/enrichie .

Exemple basique :

```
def basic_decorateur(func):
    def wrapperFunction():
        print("> avant l'execution de la fonction originale")
        func()
        print("> après l'execution de la fonction originale")
    return wrapperFunction
```

```
def dire_bonjour():
    print("Bonjour !")

# Appliquer manuellement/explicitement le décorateur:
decoratedFunction = basic_decorateur(dire_bonjour)
decoratedFunction()
```

⇒

> avant l'execution de la fonction originale
Bonjour !

> après l'execution de la fonction originale

```
# Appliquer automatiquement le décorateur:

@basic_decorateur
def say_hello():
    print("Hello !")

say_hello()
```

⇒

> avant l'execution de la fonction originale
Hello !
> après l'execution de la fonction originale

Décorateur pour fonction avec arguments :

```
def basic_decorateur2(func):
    def wrapperFunction(*args,**kwargs):
        print("> avant l'execution de la fonction avec args")
        res=func(*args,**kwargs)
        print("> après l'execution de la fonction avec args")
        return res
    return wrapperFunction

@basic_decorateur2
def say_hello_with_name(name):
    print(f"Hello {name} !")

say_hello_with_name("Laurence")
```

⇒

> avant l'execution de la fonction avec args

Hello Laurence !

> après l'execution de la fonction avec args

Décorateur mesurant la temps d'exécution :

```
import time

def logExecutionTimeDeco(func):
    def wrapperFunction(*args,**kwargs):
        startTime=time.time()
        res=func(*args,**kwargs)
        endTime=time.time()
        print(f"Durée d'execution: {endTime - startTime:.4f} secondes")
        return res
    return wrapperFunction

@logExecutionTimeDeco
def calcul_lent(x):
```

```
    time.sleep(1)  # pause 1s pour simuler un calcul
    print(f"calcul_lent({x}) returning {x*x}")
    return x*x

y=calcul_lent(5)
print("y=",y)
```

⇒

calcul_lent(5) returning 25

Durée d'execution: 1.0007 secondes

y= 25

Empilement/enchaînement de plusieurs décorateurs :

```
def logDecorator(func):
    def wrapperFunction(*args,**kwargs):
        print(f"*** Appel fonction {func.__name__} avec args={args} et kwargs={kwargs}")
        res=func(*args,**kwargs)
        print(f"*** valeur de retour = {res}")
        return res
    return wrapperFunction


#will be applied: logDecorator(logExecutionTimeDeco(calcul_lent2(x)))
@ logDecorator #outer decorator
@logExecutionTimeDeco #inner decorator
def calcul_lent2(x):
    time.sleep(1)  # pause 1s pour simuler un calcul
    print(f"calcul_lent2({x}) returning {x*x}")
    return x*x


y=calcul_lent2(5)
print("y=",y)
```

⇒

*** Appel fonction wrapperFunction avec args=(5,) et kwargs={}

calcul_lent2(5) returning 25

Durée d'execution: 1.0007 secondes

*** valeur de retour = 25

y= 25

NB: petite imperfection (peut être due à la version utilisée de python ou bien à un manque de paramétrage) : {func.__name__} affiche wrapperFunction plutôt que calcul_lent2 .

Décorateurs prédéfinis pour la programmation orientée objet :

@staticmethod	Fonction indépendante placée dans une classe qui n'a pas accès à l'instance ni à la classe elle même
@classmethod	Transforme une méthode pour qu'elle reçoive la classe comme premier argument au lieu de l'instance (`cls` au lieu de `self`).
@property et @xyz.setter	Pour configurer des propriétés avec getter/setter en complément d'éléments internes considérés privés et avec un meilleur contrôle sur les changement de valeur
...	

NB: L'application détaillée de ces décorateurs se trouve en fin de chapitre sur la programmation orientée objet .

1.4. itérateurs et générateurs

```
### itération basique avec fonction iter() et next() prédéfinies du langage python:
it = iter(range(3))
try:
    while True:
        print(next(it)) # affiche 0 puis 1 puis 2 puis provoque une exception StopIteration
except StopIteration:
        print("---fin iteration---")
```

⇒
0
1
2
---fin iteration---

```
### itérateur basique générant/renvoyant une par une
# les valeurs de startInclusive à stopExclusive:
class FromStartIncToStopExcIt:

    #constructor
    def __init__(self,startInc=0,stopExc=10):
        self.stopExc=stopExc # stop exclusive
        self.current = startInc - 1 # -1 : before first incrementation

    # __iter(obj)__ return a iterator from a iterable object
    def __iter__(self):
        return self

    # __next()__ return next value until raise StopIteration
    def __next__(self):
        self.current += 1
        if self.current >= self.stopExc:
            raise StopIteration
```

```
        return self.current

for i in FromStartIncToStopExcIt(1,10):
    print(i)
```

⇒
1
2
...
9

```
### objet itérable renvoyant une par une
# les valeurs positives d'une collection après filtrage automatique:
class PositiveValuesFilterIterable :

    #constructor
    def __init__(self,col=[]):
        self.filteredCol=list(filter(lambda x : x>=0,col))

    # __iter(obj)__ return a iterator from this/self iterable object
    def __iter__(self):
        return iter(self.filteredCol)
        # pas besoin de __next__() ici car on retoure iter(...) comportant déjà __next()__


for v in PositiveValuesFilterIterable([1,-2,5,-6,8]):
    print("positive v=",v)
```

⇒
positive v= 1
positive v= 5
positive v= 8

générateurs (avec yield) :

```
def my_basic_generator():
    yield "un"
    yield "deux"
    yield "trois"

for v in  my_basic_generator():
    print("generated value=",v)
```

⇒
generated value= un
generated value= deux
generated value= trois

```
def fromStartIncToStopExcSquareTupleGenerator(startInc=0,stopExc=10):
    current = startInc
    while current < stopExc:
        yield (current,current * current)
        current +=1

for x_xx_tuple in fromStartIncToStopExcSquareTupleGenerator(0,10):
    print(x_xx_tuple)
```

⇒

(0, 0)

(1, 1)

(2, 4)

…

(8, 64)

(9, 81)

1.5. expressions régulières

Besoin d'importer le module "**re**" pour ensuite effectuer des filtrages/comparaisons/remplacements en fonction de parties qui matchent ou pas au sein d'une chaîne de caractères.

Symboles ayant une signification particulière : . ^ $ * + ? { } [ ] \ | ( )

. Le point correspond à n'importe quel caractère.
^ Indique un commencement de segment mais signifie aussi "contraire de"
$ Fin de segment
[xy] Une liste de segment possible. Exemple [abc] équivaut à : a, b ou c
(x|y) Indique un choix multiple type (red|rouge) équivaut à "red" OU "rouge"
\d le segment est composé uniquement de chiffre, ce qui équivaut à [0-9].
\D le segment n'est pas composé de chiffre, ce qui équivaut à [^0-9].
\s Un espace, ce qui équivaut à [\t\n\r\f\v].
\S Pas d'espace, ce qui équivaut à [^ \t\n\r\f\v].
\w Présence alphanumérique, ce qui équivaut à [a-zA-Z0-9_].
\W Pas de présence alphanumérique [^a-zA-Z0-9_].
\ Est un caractère d'échappement (pour interpréter normalement de caractère qui suit)

Répétions éventuelles :

A**{2}** : on attend à ce que la lettre A (en majuscule) se répète 2 fois consécutivement.
SEG**{1,9}** : on attend à ce que le segment SEG se répète de 1 à 9 fois.
SEG**{,10}** : on attend à ce que le segment SEG ne soit pas présent du tout ou présent jusqu'à 10 fois.
SEG**{1,}** : on attend à ce que le segment SEG soit présent au mois une fois.
x**?** : 0 ou 1 fois
x+ :1 fois ou plus
x* : 0, 1 ou plus

Exemples :

NB : le préfixe r"..." signifie "raw string" et permet d'éviter des warnings au sein des versions très récentes de python.

Tests de correspondances :

```
import re

tab = [ "Durand" , "2a" , "toto" , "R2d2", "Dupond"]
for s in tab:
    matchTuple = (s , re.match(r"^[A-Z]\D" , s))
    print(matchTuple)
    if matchTuple[1] :
        print (matchTuple[0] + " est un nom correct")
    else:
        print (matchTuple[0] + " n'est pas un nom correct")
```

⇒

('Durand', <re.Match object; span=(0, 2), match='Du'>) Durand est un nom correct

('2a', None) 2a n'est pas un nom correct

('toto', None) toto n'est pas un nom correct

('R2d2', None) R2d2 n'est pas un nom correct

('Dupond', <re.Match object; span=(0, 2), match='Du'>) Dupond est un nom correct

Recherche de sous parties selon correspondance avec une expression régulière :

```
import re
allNumbers = re.findall(r"([0-9]+)", "Entre 001 et 999")
print("allNumbers=",allNumbers) # ['001', '999']
```

Remplacement (substitution) au sein d'une chaîne de caractères via re.sub():

```
import re
s1 = "12 8 16.8 9 -1"
print("s1=",s1)
s2 = re.sub(r"\s", r"," , s1) #remplace les espaces par des virgules
print("s2=",s2) # "12,8,16.8,9,-1"
```

et plein d'autres possibilités (groupes , …)

si exercices , faire ça de manière très progressive ...

ANNEXES

X - Affichages graphiques en python

1. Affichage graphiques en python

1.1. ...

XI - Calculs scientifiques / python

1. Calculs scientifiques en python

1.1. installation et utilisation de numpy dans env python virtuel

Préalablement, si nécessaire :

```
python -m pip install --user pipenv
python -m pipenv --version
```

Au sein d'un projet python :

installation de numpy via pipenv :

```
python -m pipenv install numpy
```

utilisation de numpy au sein de pipenv :

```
python -m pipenv run python inverse-matrix.py
```

1.2. Calcul matriciel (algèbre linéaire) avec numpy

inverse-matrix.py

```
import numpy as np

#NB: np.matrix() is deprecated , use np.array() instead
mA =  np.array([
                        [ 5.0 , 2.0 , 2.0 ] ,
                        [ -5.0 , 6.0 , 4.0 ] ,
                        [ 8.0 , 3.0 , 7.0 ]
                ])
print("matrice mA=\n",mA)
print("nombre d'élements de la matrice mA=", np.size(mA))
print("taille (nbLignes,nbColonnes) de la matrice mA=", np.shape(mA))
```

matrice mA=
[[5. 2. 2.]
[-5. 6. 4.]
[8. 3. 7.]]
nombre d'élements de la matrice mA= **9**
taille (nbLignes,nbColonnes) de la matrice mA= **(3, 3)**

```
detA = np.linalg.det(mA)
print("determinant detA=",detA)
```

```
#trA = np.transpose(mA)
trA=mA.T
print("transposee trA=\n",trA)

invA = np.linalg.inv(mA)
print("matrice inverse invA=\n",invA)
```

```
determinant detA= 158.00000000000003
transposee trA=
 [[ 5. -5.  8.]
 [ 2.  6.  3.]
 [ 2.  4.  7.]]
matrice inverse invA=
 [[ 0.18987342 -0.05063291 -0.02531646]
 [ 0.42405063  0.12025316 -0.18987342]
 [-0.39873418  0.00632911  0.25316456]]
```

```
#prodAinvA = mA * invA # ok si mA et invA ont meme taille
prodAinvA = np.dot(mA ,invA) #produit matriciel
print("verification, mA * invA = matrice identite=\n",prodAinvA)
prodAinvA = np.round(prodAinvA ,4)
print("verification (après arrondi à 0.0001) , mA * invA = \n",prodAinvA)
```

```
verification, mA * invA = matrice identite=
 [[ 1.00000000e+00 -1.21430643e-17  0.00000000e+00]
 [-4.44089210e-16  1.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.38777878e-17  1.00000000e+00]]
verification (après arrondi à 0.0001) , mA * invA =
 [[ 1. -0.  0.]
 [-0.  1.  0.]
 [ 0. -0.  1.]]
```

```
#résolution d'un système d'équations linéaire
print("Résolution de 3x+y=9 et x+2y=8");
A= np.array([[3,1], [1,2]])
print("A=\n" , A)
B = np.array([9,8])
print("B=\n" , B)
X = np.linalg.solve(A, B)
print("La solution de A*X=B est X=\n" , X);
aFoisX = np.dot(A ,X)
print("verification, A * X = \n",aFoisX)
```

```
Résolution de 3x+y=9 et x+2y=8
A=
 [[3 1]
 [1 2]]
B=
 [9 8]
La solution de A*X=B est X=
 [2. 3.]
verification, A * X =
 [9. 8.]
```

```
#initialisation de matrices:
V0 = np.zeros(3); #1 dimension , 0 partout
print("V0=\n" , V0)

V1 = np.ones(3);#1 dimension , 1 partout
print("V1=\n" , V1)

M0 = np.zeros((3,2));#2 dimensions 3lignes,2cols , 0 partout
print("M0=\n" , M0)
M1 = np.ones((3,2));#2 dimensions 3lignes,2cols , 1 partout
print("M1=\n" , M1)

#matrice identité (ici à 2 dimensions 3lignes,3cols)
# ( 1 sur diagonale , 0 ailleurs)
MI = np.eye(3,3); #ou bien MI= np.identity(3)
print("MI=\n" , MI)
```

```
V0=
 [0. 0. 0.]
V1=
 [1. 1. 1.]
M0=
 [[0. 0.]
 [0. 0.]
 [0. 0.]]
M1=
 [[1. 1.]
 [1. 1.]
 [1. 1.]]
MI=
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
#matrice de nombres complexes:
mc = np.array([[  2j, 4+3j],
               [2+5j, 5   ],
               [   3,    6+2j]])
print("mc=\n" , mc)

#NB: la matrice conjuguée est une matrice ou chaque élément
#est le nombre complexe conjugé (meme partie réélle , partie imaginaire opposée)
mcConj = np.conj(mc)
print("mcConj=\n" , mcConj)
```

```
mc=
 [[0.+2.j 4.+3.j]
 [2.+5.j 5.+0.j]
```

```
 [3.+0.j 6.+2.j]]

mcConj=
 [[0.-2.j 4.-3.j]
 [2.-5.j 5.-0.j]
 [3.-0.j 6.-2.j]]
```

XII - Accès aux bases de données

1. Accès aux bases de données en python

1.1. accès à MySQL en python via mysql.connector

Installation du connecteur :

```
python -m pip install mysql-connector-python
```

Exemple d'utilisation :

```
import json
import mysql.connector
# python -m pip install mysql-connector-python

connection_params = {
   'host': "localhost",
   'port' : 3306,
   'user': "root",
   'password': "root",
   'database': "geoDB"
}

#se connecter à la base de donnees et utiliser la connexion
#** operator to unpack dictionaries directly into keyword arguments of a function
with mysql.connector.connect(**connection_params) as db  :

   #insertion de donnees:
   insertRequest = """insert into departement
         (numero, nom, population, superficie, prefecture)
         values (%s, %s, %s, %s , %s)"""  # all parameter marker must be %s (as ?), not %d
   params = ("57", "Moselle", 0, 0 , "Metz")
   with db.cursor() as c :
      c.execute(insertRequest, params)
      db.commit()
      print("Nombre de lignes insérées :", c.rowcount)

   #mise à jour de donnees:
   updateRequest = """update departement
         set population=%s, superficie=%s
         where numero=%s"""
   params = (1049942, 6216 ,"57")
   with db.cursor() as c :
      c.execute(updateRequest, params)
      db.commit()
      print("Nombre de lignes modifiees :", c.rowcount)

   # requete sql select ...
```

```
selectRequest = "select numero, nom, population, superficie, prefecture from departement"
columnNamesTuple = ("numero", "nom", "population", "superficie", "prefecture")

with db.cursor() as c :
    c.execute(selectRequest)
    resultats = c.fetchall()
    print("Nombre de lignes sélectionnées/récupérées :", c.rowcount)
    for departement in resultats:
        #print(departement) # department is a tuple (immutable list of field values)
        depAsDict = {columnNamesTuple[i] : departement[i]
                                for i, _ in enumerate(columnNamesTuple)}
        #print(depAsDict)
        depAsJsonString = json.dumps(depAsDict);
        print (depAsJsonString)

#suppression de donnees:
deleteRequest = """delete from departement
      where numero=57"""
with db.cursor() as c :
    c.execute(deleteRequest)
    db.commit()
    print("Nombre de lignes supprimées :", c.rowcount)

#NB : fermeture automatique de la connexion en fin de bloc with db:
```

⇒

Nombre de lignes insérées : 1

Nombre de lignes modifiees : 1

Nombre de lignes sélectionnées/récupérées : 19

{"numero": "02", "nom": "Aisne", "population": 539783, "superficie": 7369, "prefecture": "Laon"}

...

{"numero": "95", "nom": "Val-d-Oise", "population": 1205539, "superficie": 1246, "prefecture": "Cercy-Pontoise"}

Nombre de lignes supprimées : 1

Structure de la base geoDB compatible avec le code python précédent :

geoDB.mysql_script.sql

```
CREATE DATABASE IF NOT EXISTS geoDB ;

USE geoDB;

# DROP TABLE Dans ordre inverse des CREATE TABLE et/ou INSERT INTO
# si FOREIGN KEY CONSTRAINTS
DROP TABLE IF EXISTS Departement;
DROP TABLE IF EXISTS Region;


CREATE TABLE Region(
   num VARCHAR(8) PRIMARY KEY,
        nom VARCHAR(64) ,
        chef_lieu VARCHAR(32)
```

```
) ENGINE=InnoDB;
# via des quotes inversees , un nom de tables ou de colonne pourrait comporter
# des espaces ex `chef lieu`


CREATE TABLE Departement(
   numero VARCHAR(6) PRIMARY KEY,
          nom VARCHAR(48) ,
          population INTEGER,
          superficie INTEGER,
          prefecture VARCHAR(48),
          refRegion VARCHAR(8)
)ENGINE=InnoDB;

ALTER TABLE Departement
 ADD CONSTRAINT FK_RegionValidePourDepartement
  FOREIGN KEY (refRegion) REFERENCES Region(num) ;

# show tables est une instruction MySql qui affiche la liste des tables
show tables;

INSERT INTO Region(num, nom , chef_lieu )
VALUES ("FR-IDF", "Ile-de-France" , "Paris"),
  ("FR-HDF", "Hauts-de-France" , "Lille" ),
  ("FR-NOR", "Normandie","Rouen"),
  ("FR-BRE", "Bretagne" , "Rennes"),
  ("FR-NAQ", "Nouvelle-Aquitaine" , "Bordeaux" ),
  ("FR-OCC", "Occitanie","Toulouse"),
  ("FR-PDL", "Pays-de-la-Loire" , "Nantes" ),
  ("FR-PAC", "Provence-Alpes-Cote-Azur","Marseille"),
  ("FR-CVL", "Centre-Val-de-Loire" , "Orleans"),
  ("FR-GES", "Grand-Est" , "Strasbourg" ),
  ("FR-ARA", "Auvergne-Rhone-Alpes" , "Lyon"),
  ("FR-BFC", "Bourgogne-Franche-Comte" , "Dijon" ),
  ("FR-COR", "Corse","Ajaccio");

INSERT INTO Departement(numero, nom , prefecture , population , superficie,  refRegion )
     VALUES ("75", "Paris" , "Paris" ,2220445 ,105, "FR-IDF"),
                ("92", "Hauts-de-Seine" , "Nanterre",1597770, 176 , "FR-IDF"),
                                ("78", "Yvelines" , "Versailles" ,1421670 ,2284, "FR-IDF"),
                ("93", "Seine-Saint-Denis" , "Bobigny", 1571028, 236 , "FR-IDF"),
                                ("95", "Val-d-Oise" , "Cercy-Pontoise" ,1205539 ,1246, "FR-IDF"),
                ("77", "Seine-et-Marne" , "Melun",1377846, 5915 , "FR-IDF"),
                                ("91", "Essonne" , "Evry" ,1268228 ,1804, "FR-IDF"),
                ("94", "Val-de-Marne" , "Creteil",1365039, 245 , "FR-IDF"),
                ("59", "Nord" , "Lille" ,2603472, 5743 , "FR-HDF"),
          ("62", "Pas de calais" , "Arras", 1472589,6671 , "FR-HDF"),
                                ("60", "Oise" , "Beauvais" ,818380,5860 , "FR-HDF"),
                                ("80", "Somme" , "Amiens" ,571632 ,6170, "FR-HDF"),
                                ("02", "Aisne" , "Laon" ,539783 ,7369, "FR-HDF"),
                                ("27", "Eure" , "Evreux" ,598347, 6040 , "FR-NOR"),
          ("76", "Seine-Maritime" , "Rouen", 1257920,6278 , "FR-NOR"),
                                ("14", "Calvados" , "Caen" ,691670,5548 , "FR-NOR"),
                                ("61", "Orne" , "Alençon" ,287750 ,6103, "FR-NOR"),
                                ("50", "Manche" , "Saint-Lo" ,499958 ,5938, "FR-NOR");

# affichage de la structure pour verifier:
#describe Region;
#describe Departement;

# affichage des donnees pour verifier:
SELECT * FROM Region;
```

```
SELECT * FROM Departement;
```

1.2. accès à MongoDB en python

... à compléter ...

XIII - Http , web , api REST

1. http , web , api rest en python

1.1. Vue d'ensemble sur frameworks "web" et "rest" en python

Framework	*Fonctionnalités*	*Caractéristiques*
django	pour application WEB générant pages HTML en python et éventuellement pour api REST	très utilisé, très populaire framework très complet (un peu surdimensionné pour simple api REST)
flask	pour api REST ou bien appli web simple	Bon compromis fonctionnalités/complexités
falcon	pour api REST seulement	assez peu utilisé mais efficace / léger
... autres ...	...	....

1.2. api rest avec Flask

XIV - Annexe – Test

1. Test unitaire python

Texte ....

1.1. Titre Paragraphe

Texte ....

XV - Annexe – interfaçage python et c/c++

1. Liaison entre python et langage c/c++

1.1. Vue d'ensemble sur les alternatives possibles

alternatives	*principes*	*caractéristiques (avantages, inconvénients)*
ctypes	librairies de bas niveau pour appeler des fonctions "C" en précisant tous les détails d'interfaçage en python	- solutions de bas niveau - pour cas simples et de petites tailles (peu de fonctions à interfacer)
CFFI	pour relier python à du code écrit en c. Scripts à prévoir pour générations	- solution plus élaborée / automatisée - solution plus complexe appropriée lorsqu'il y a un grand nombre de fonction à interfacer
PyBind11	pour relier python à du code écrit en c++ v11. Paramétrages en c++ , Scripts à prévoir pour générations	- solution dédiée à C++ V11 (orienté objet)
Cython	génère et compile / utilise du code c depuis une description python	- plutôt à voir comme une optimisation d'un code python que comme un interface avec une librairie C existante .
...	d'autres librairies existent	solutions moins connues , moins utilisées

1.2. Exemple de librairie partagée écrite en langage C :

clib.h

```
#ifndef CLIB_H_INCLUDED
#define CLIB_H_INCLUDED
#include <stdbool.h>

double calculerTva(double ht, double tauxPct);
double xPuissanceN(double x, int n);
char* sayHello(char* s);
bool isEquals(char* s1, char* s2);
double moyenne(double* tab, int n);
void transposeMatriceCarreeV1(int n,double* matrice,double* mt);
void transposeMatriceCarreeV2(int n,double matrice[n][n],double mt[n][n]);

#endif // CLIB_H_INCLUDED
```

clib.c

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "clib.h"

double calculerTva(double ht, double tauxPct){
    return ht*tauxPct/100;
}


double xPuissanceN(double x, int n){

  double res =  pow(x, n);
   printf("xPuissanceN(x,n) called with x=%f and n=%d , res=%f \n",x , n , res );
  return res;
}

char* sayHello(char* s){
   char* sHello="Hello ";
   size_t lsh=strlen(sHello);
   size_t ls = strlen(s);
   char* sRes = (char*) malloc( (lsh + ls + 1) * sizeof(char) );
   strcpy(sRes,sHello);
   strcat(sRes,s);
   printf("sayHello(s) called with s=%s and res=%s \n",s , sRes );
   return sRes;
   //attention , bloc mémoire à libérer via free() par fonction appelante
}

bool isEquals(char* s1, char* s2){
   bool res;
  if(strcmp(s1,s2)==0)
   res=true;
  else
   res = false;
   printf("isEquals(s1,s2) called with s1=%s and s2=%s , res=%d \n",s1, s2 , res );
   return res;
}

double moyenne(double* tab, int n){
  double somme=0;
  int i;
  for(i=0;i<n;i++){
   somme += tab[i];
  }
  double moy = somme / n;
  printf("moyenne(tab,n) called with n=%d and res=%f \n",n , moy );
  return moy;
}

void transposeMatriceCarreeV1(int n,double* matrice,double* mt){
 int i,j;
  for(i=0;i<n;i++){
```

```
    for(j=0;j<n;j++){
      mt[i*n+j]=matrice[j*n+i];
    }
  }
}


void transposeMatriceCarreeV2(int n,double matrice[n][n],double mt[n][n]){
  int i,j;

  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      mt[i][j]=matrice[j][i];
    }
  }
}
```

main.c (test)

```
#include <stdio.h>
#include <stdlib.h>
#include "clib.h"

int main()
{
   printf("test myclib\n");

   double montantHt=200;
   double tauxTva=20;
   double tva = calculerTva(montantHt,tauxTva);
   printf("tva=%f\n",tva);

   double x=5;
   int n=3;
   double y = xPuissanceN(x,n);
   printf("y=%f\n",y);


   char* sRes = sayHello("guy");
   printf("sRes=%s\n",sRes);
   free(sRes);

   char* sA = "aaa";
   bool bRes = isEquals(sA, "aaa");
   printf("bRes=%d\n",bRes);
   if(bRes == true)
      printf("sA vaut aaa\n");
   else
      printf("sA est different de aaa\n");

   double tabX[5] = { 5.0 , 3.5 , 6.5 , 2.0 , 8.0};
   double m = moyenne(tabX, 5);
   printf("moyenne=%f\n",m);
```

```
    int i;

    double matriceA[3][3] = {
                    { 5.0 , 2.0  , 2.0 } ,
                    { -5.0 , 6.0 , 4.0 } ,
                    { 8.0 , 3.0 , 7.0 }
                  };
    printf("matriceA:\n");
    for(i=0;i<3;i++){
        printf("%f %f %f\n",matriceA[i][0] , matriceA[i][1] , matriceA[i][2]);
    }

    double trA[3][3];
    //transposeMatriceCarreeV1(3, matriceA,trA);
    transposeMatriceCarreeV2(3, matriceA,trA);
    printf("matrice transposee trA:\n");
    for(i=0;i<3;i++){
        printf("%f %f %f\n",trA[i][0] , trA[i][1] , trA[i][2]);
    }
    return 0;
}
```

script de construction de la librairie partagée :

build_myclib_via_gcc.bat

```
cd /d %~dp0
set MINGW=C:\Program Files (x86)\CodeBlocks\MinGW
set PATH=%MINGW%\bin;%PATH%

REM myclib.exe "all in .exe , no shared librairie (.so or .dll)"
gcc clib.c main.c -o myclib.exe

REM clib.so not found on windows (.so on linux)

REM building "shared library" :
REM gcc -shared -o myclib.so -fPIC clib.c
gcc -shared -o myclib.dll -fPIC clib.c

REM building myclibbis.exe (using shared libray):
REM -L. option is need to tell in wich directory search myclib.so (or .dll)
gcc  main.c -L. -lmyclib -o  myclibbis.exe

pause
```

==> ça génère (entre autre) **myclib.dll** et **myclibbis.exe** utilisant cette dll

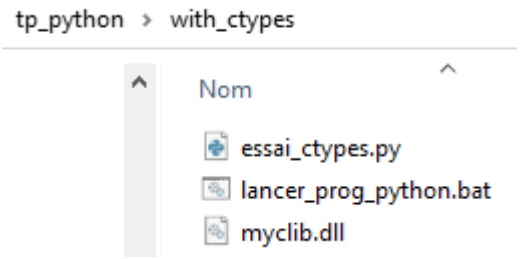
lancer_test_myclib.bat

```
REM .\myclib
.\myclibbis
pause
```

Affichages via le programme de test (en langage C) :

test myclib
tva=40.000000
xPuissanceN(x,n) called with x=5.000000 and n=3 , res=125.000000
y=125.000000
sayHello(s) called with s=guy and res=Hello guy
sRes=Hello guy
isEquals(s1,s2) called with s1=aaa and s2=aaa , res=1
bRes=1
sA vaut aaa
moyenne(tab,n) called with n=5 and res=5.000000
moyenne=5.000000
matriceA:
5.000000 2.000000 2.000000
-5.000000 6.000000 4.000000
8.000000 3.000000 7.000000
matrice transposee trA:
5.000000 -5.000000 8.000000
2.000000 6.000000 3.000000
2.000000 4.000000 7.000000

1.3. Appels via ctypes (low-level standard python library)



myclib.dll (ou .so sous linux) est ici recopié près du code suivant écrit en python

essai_ctypes.py

```
import ctypes
import pathlib
import platform

#platform.system() return "Linux" or "Windows" or "..."
currentPlatfom=platform.system()

#sharedLibraryExt ".dll" on windows or ".so" on linux
sharedLibraryExt=".dll" if currentPlatfom=="Windows" else ".so"


#myclib.so OR myclib.dll
libfullpath = pathlib.Path().absolute() / ("myclib"+ sharedLibraryExt)

#exemple: D:\tp\tp_python\with_ctypes\myclib.dll
#as WindowsPath
libfullpath=str(libfullpath).replace('\\','/')
# Exemple: D:/tp/tp_python/with_ctypes/myclib.dll
```

```
print(libfullpath)

#cdll.LoadLibrary(libfullpath) equivalent a CDLL(libfullpath)
myclib = ctypes.cdll.LoadLibrary(libfullpath)
#myclib = ctypes.CDLL(libfullpath)

#print(myclib)
#dispay <CDLL 'D:\tp\tp_python\with_ctypes\myclib.dll', handle 6d080000 at 0x1767148>
#print(myclib.xPuissanceN)
#display <_FuncPtr object at 0x034BD338>

ht=200.00
tauxTvaPct=20 #au sens 20%

#appel de double calculerTva(double ht, double tauxPct);
myclib.calculerTva.restype = ctypes.c_double
myclib.calculerTva.argtypes = ctypes.c_double , ctypes.c_double
resTva = myclib.calculerTva(ht, tauxTvaPct)

print("calculerTva res=",resTva)

print("----------------------------------------")

x=5.0
n=3

#appel de double xPuissanceN(double x, int n);
myclib.xPuissanceN.restype = ctypes.c_double
myclib.xPuissanceN.argtypes = ctypes.c_double, ctypes.c_int
res = myclib.xPuissanceN(x, n)

print("xPuissanceN res=",res)

print("----------------------------------------")

s1="toto";
print("type de s1=", type(s1));
#appel de char* sayHello(char* s);
myclib.sayHello.restype = ctypes.c_char_p
myclib.sayHello.argtype = ctypes.c_char_p
#NB: s1 est de type str python (unicode)
#tandis que s1AsByteArray est de type tableau de bytes
#(comptatible char* du langage c avec caractères ASCII sur 1 seul octet proche utf-8)
s1AsByteArray = s1.encode('utf-8')
print("type et valeur de s1AsByteArray = s1.encode('utf-8'): ", type(s1AsByteArray) ,
s1AsByteArray);

resSayHelloAsByteArray = myclib.sayHello(s1AsByteArray)
print("resSayHelloAsByteArray=",resSayHelloAsByteArray)
resSayHello = resSayHelloAsByteArray.decode('utf-8')

print("resSayHello=",resSayHello)
```

```
print("----------------------------------------")

sA = "aaa"

#appel de bool isEquals(char* s1, char* s2);
myclib.isEquals.restype = ctypes.c_bool
myclib.isEquals.argtypes = ctypes.c_char_p , ctypes.c_char_p
#NB: b"aaa" est une valeur de type byteArray
resComparaison = myclib.isEquals(sA.encode('utf-8'),b"aaa")
print("resComparaison=",resComparaison)
if resComparaison :
        print("sA vaut aaa")
else :
        print("sA different de aaa")

print("----------------------------------------")

tabX = [ 5.0 , 3.5 , 6.5 , 2.0 , 8.0 ];
tailleTabX=len(tabX)
print("tailleTabX=" , tailleTabX ," tabX=",tabX)

#appel de double moyenne(double* tab, int n);
myclib.moyenne.restype = ctypes.c_double
myclib.moyenne.argtypes = ctypes.POINTER(ctypes.c_double),ctypes.c_int
resMoy = myclib.moyenne((ctypes.c_double * tailleTabX)(*tabX), tailleTabX)

print("resMoy=",resMoy)

print("----------------------------------------")

def matrixFromListOfListToBigList(mat,nbLines, nbCols) :
    matBigList = []
    for i in range(nbLines):
        for j in range(nbCols):
            matBigList.append(mat[i][j])
    return matBigList

def matrixFromBigListToListOfList(matBigList,nbLines, nbCols) :
    mat = []
    for i in range(nbLines):
        colList = []
        for j in range(nbCols):
            colList.append(matBigList[i*n+j])
        mat.append(colList)
    return mat

matA =  [
         [ 5.0 , 2.0  , 2.0 ] ,
         [ -5.0 , 6.0 , 4.0 ] ,
         [ 8.0 , 3.0 , 7.0 ]
       ]
print("matA=",matA)
matABigList = matrixFromListOfListToBigList(matA,3,3)
```

```
print("matABigList=",matABigList)

trABigList = [ 0.0 , 0.0  , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 , 0.0 ]
n=3

#appel de void transposeMatriceCarreeV1(int n,double* matrice,double* mt)
#   ou de void transposeMatriceCarreeV2(int n,double matrice[n][n],double mt[n][n]);

myclib.transposeMatriceCarreeV2.argtypes =
ctypes.c_int , ctypes.POINTER(ctypes.c_double), ctypes.POINTER(ctypes.c_double)

pointerOnTrBigList=(ctypes.c_double * (n  * n))(*trABigList)

myclib.transposeMatriceCarreeV2(n,
                                (ctypes.c_double * (n * n))(*matABigList),
                                 pointerOnTrBigList)
print("pointerOnTrBigList=",pointerOnTrBigList)
trABigList=pointerOnTrBigList[:]
print("trABigList=",trABigList)
trA = matrixFromBigListToListOfList(trABigList,3,3);
print("trA=",trA)
```

python essai_ctypes.py

```
==>
D:\tp\tp_python\with_ctypes>python essai_ctypes.py
D:/tp/tp_python/with_ctypes/myclib.dll
calculerTva res= 40.0
----------------------------------------
xPuissanceN(x,n) called with x=5.000000 and n=3 , res=125.000000
xPuissanceN res= 125.0
----------------------------------------
type de s1= <class 'str'>
type et valeur de s1AsByteArray = s1.encode('utf-8'):  <class 'bytes'> b'toto'
sayHello(s) called with s=toto and res=Hello toto
resSayHelloAsByteArray= b'Hello toto'
resSayHello= Hello toto
----------------------------------------
isEquals(s1,s2) called with s1=aaa and s2=aaa , res=1
resComparaison= True
sA vaut aaa
----------------------------------------
tailleTabX= 5  tabX= [5.0, 3.5, 6.5, 2.0, 8.0]
moyenne(tab,n) called with n=5 and res=5.000000
resMoy= 5.0
----------------------------------------
matA= [[5.0, 2.0, 2.0], [-5.0, 6.0, 4.0], [8.0, 3.0, 7.0]]
matABigList= [5.0, 2.0, 2.0, -5.0, 6.0, 4.0, 8.0, 3.0, 7.0]
pointerOnTrBigList= <__main__.c_double_Array_9 object at 0x014B98E0>
trABigList= [5.0, -5.0, 8.0, 2.0, 6.0, 3.0, 2.0, 4.0, 7.0]
trA= [[5.0, -5.0, 8.0], [2.0, 6.0, 3.0], [2.0, 4.0, 7.0]]
```

1.4. variante ctypes avec numPy

essai_ctypes_numpy.py

```
import ctypes
import pathlib
import platform
import numpy as np

#platform.system() return "Linux" or "Windows" or "..."
currentPlatfom=platform.system()

#sharedLibraryExt ".dll" on windows or ".so" on linux
sharedLibraryExt=".dll" if currentPlatfom=="Windows" else ".so"


#myclib.so OR myclib.dll
libfullpath = pathlib.Path().absolute() / ("myclib"+ sharedLibraryExt)

#exemple: D:\tp\tp_python\with_ctypes\myclib.dll
#as WindowsPath
libfullpath=str(libfullpath).replace('\\','/')
# Exemple: D:/tp/tp_python/with_ctypes/myclib.dll
print(libfullpath)

#cdll.LoadLibrary(libfullpath) equivalent a CDLL(libfullpath)
myclib = ctypes.cdll.LoadLibrary(libfullpath)
#myclib = ctypes.CDLL(libfullpath)

#print(myclib)
#dispay <CDLL 'D:\tp\tp_python\with_ctypes\myclib.dll', handle 6d080000 at 0x1767148>

tabXnp = np.array([5.0 , 3.5 , 6.5 , 2.0 , 8.0])
tailleTabXnp=len(tabXnp)
print("tailleTabXnp=" , tailleTabXnp ," tabXnp=",tabXnp)

#appel de double moyenne(double* tab, int n);
myclib.moyenne.restype = ctypes.c_double
myclib.moyenne.argtypes=
np.ctypeslib.ndpointer(dtype=np.float64, ndim=1, flags='C_CONTIGUOUS'), ctypes.c_int

resMoy = myclib.moyenne(tabXnp, tailleTabXnp)

print("resMoy=",resMoy)

print("----------------------------------------")

matAnp =  np.matrix([
                        [ 5.0 , 2.0 , 2.0 ] ,
                        [ -5.0 , 6.0 , 4.0 ] ,
                        [ 8.0 , 3.0 , 7.0 ]
                    ])
print("matAnp=",matAnp)
```

```
trAnp =  np.matrix([
                        [ 1.0 , 1.0 , 1.0 ] ,
                        [ 1.0 , 1.0 , 1.0 ] ,
                        [ 1.0 , 1.0 , 1.0 ]
                    ])
n=3

#appel de void transposeMatriceCarreeV2(int n,double matrice[n][n],double mt[n][n]);
myclib.transposeMatriceCarreeV2.argtypes = [
        ctypes.c_int ,
        np.ctypeslib.ndpointer(dtype=np.float64, ndim=2, flags='C_CONTIGUOUS'),
        np.ctypeslib.ndpointer(dtype=np.float64, ndim=2, flags='C_CONTIGUOUS')
        ]

myclib.transposeMatriceCarreeV2(n,matAnp,trAnp);

print("trAnp=",trAnp)
```

==> résultats :

```
tailleTabXnp= 5  tabXnp= [5.  3.5 6.5 2.  8. ]
moyenne(tab,n) called with n=5 and res=5.000000
resMoy= 5.0
----------------------------------------
matAnp= [[ 5.  2.  2.]
 [-5.  6.  4.]
 [ 8.  3.  7.]]
trAnp= [[ 5. -5.  8.]
 [ 2.  6.  3.]
 [ 2.  4.  7.]]
```

en lançant
python -m pipenv run python essai_ctypes_numpy.py

dans un contexte de projet préalablement initialisé par
python -m pipenv install numpy

1.5. Appels via CFFI (C Foreign Function Interface)

Solution complexe

https://cffi.readthedocs.io/en/latest/goals.html

https://sametmax.com/introduction-aux-extensions-python-avec-cffi/

XVI - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TD/TP python

2.1. Installer si besoin python et VisualStudioCode ou pyCharm

https://www.python.org

https://code.visualstudio.com/ puis ajouter ensuite l'extension python .
et/ou
https://www.jetbrains.com/fr-fr/pycharm/download (la version "community" est déjà très bien)

Pour tester l'installation :

python --version

2.2. Utilisation directe de python en mode interactif

python

```
>>> x=5
>>> y=6
>>> x+y
11
>>>exit
```

2.3. Syntaxes élémentaires (variables , boucles , ...)

Exo A1	Ecrire un script a1.py qui affiche "Hello world"
Exo A2	Ecrire un script a2.py qui : - initialise 3 variables (ex : jour=5 , mois = "avril" , annee = 2025) - construit par concaténation un message de type "aujourdhui=5 avril 2025" - affiche ce message

Exo A3	Ecrire un script a3.py qui : - initialise une variable largeur à la valeur 5 et une variable longeur à 10 - qui calcul et affiche le périmètre et la surface d'un rectangle
Exo A4	Ecrire un script a4.py qui : - demande à saisir un jour (ex : lundi ou ...) - affiche "bonne journée" si on est en semaine ou bien "bon weekend" si on est samedi ou dimanche
Exo A4bis	Ecrire un script a4.py qui : - demande à saisir un age (ex : 15 ou 28) - affiche "voici un verre de vin" si l'age est entre 18 et 70 ou bien "voilà un jus de fruit" sinon
	../..
Exo A5	Ecrire un script a5.py qui : - initialise un tableau de valeurs avec [10, 5, 2, 8 , 4 ,7 , 3 , 6 , 9] - affiche la longueur de ce tableau - affiche en boucle les valeurs au carré - affiche en boucle les valeurs paires (avec v % 2 == 0) - affiche une par une les valeurs par ordre inverse du tableau initial - trouve et affiche la première impaire
Exo A6	Ecrire un script a6.py qui : boucle (via while nombre!= 33) sur : - saisir un nombre - afficher "trop grand" si ce nombre est supérieur à 33 - afficher "trop petit" si ce nombre est inférieur à 33 - afficher "vous avez trouvé 33 en n=3 essais"
	Ecrire un script a7.py qui : génère par concaténation du caractère '*' et affiche en boucle 8 lignes d'étoiles formant globalement un triangle : * ** *** **** ***** ****** ******* ********

2.4. Structures de données (list, set , dictionnaires)

Exo B1 (list)	Ecrire un script b1.py qui : - initialise une liste l1 avec les valeurs [-1, 2 , -8 , 7 , 4 , 3 , -9 , 9 , 12 , -5 , 5] - construit et affiche la liste positifs des valeurs positives - construit et affiche la liste impairs des valeurs impaires - initialise une liste noms avec les valeurs ["jean" , "Luc" , "eric" , "Julie"] - retire toutes les valeurs ne commençant pas par une majuscule dans la liste des noms - affiche la liste des noms restants
ExoB2 (list)	Ecrire un script b2.py qui : - initialise une chaîne de caractères sSuite = "2;7;8;9;26;5" - transforme sSuite en un tableau de valeur numériques (avec " ;" comme séparateur") - affiche une par une les valeurs du tableau - ajoute la valeur 4 en début de liste/tableau - ajoute la valeur 30 en fin de tableau/liste - ré-affiche globalement les nouvelles valeurs de la liste - ré-affiche globalement (dans l'ordre inverse) les nouvelles valeurs de la liste
ExoB3 (set)	Ecrire un script b3.py qui va: - partir de liste = ["rouge" , "vert" , "bleu" , "rouge" , "vert"] - générer l'ensemble colorSet des valeurs sans doublon - afficher les valeurs et le type de colorSet - initialise colorSet2 avec colorSet2 = { "jaune" , "vert" , "orange"} - afficher l'union des ensemble colorSet et colorSet2 - initialiser jourSet avec{ "erreur", "lundi" , "mardi" , "mercredi" , "jeudi"} - supprimer l'élément "erreur" de jourSet - ajouter l'élément "vendredi" dans jourSet - ré-afficher jourSet - ajouter d'un coup les l'éléments {"samedi" , "dimanche"} dans jourSet - ré-afficher jourSet - vider tous les éléments de jourSet - ré-afficher jourSet
ExoB4 (dict)	Ecrire un script b4.py qui va : - partir de values = [-2 , -7 , 2 , 8 , -9 , 10 , 1 , 3 , -3 , 5]

	dicoStats = { 'nb_positif' : 0 , 'nb_negatif' : 0 , 'nb_pair' : 0 , 'nb_impair' : 0 , 'taille' : 0 , 'somme' : 0 - calculer et afficher les éléments du dictionnaire des statistiques en fonction de l'analyse du tableau values .

ExoB5
(dict)
	Ecrire un script b5.py qui va :
	- partir de
	dicoProduits={
	'fruit' : ["banane" , "orange" , "pomme"] ,
	'legume' : ["carotte" , "choux" , "tomate"],
	'divers' : ['stylo' , 'cahier']
	}
	- supprimer la categorie de produits qui ne se mange pas
	- afficher le dictionnaire complet:
	- afficher la liste des clefs (catégories):
	- afficher la liste des fruits
	- transformer les noms des produits en majuscule au sein du dictionnaire
	- ré-afficher le dictionnaire

ExoB6 (list/dict)	Ecrire un script b6.py qui va : - partie de listePays=[{ 'nom':'France' , 'capitale' : 'Paris' , 'population': 66352469 }, { 'nom':'Allemagne' , 'capitale' : 'Berlin' , 'population': 81174000 }, { 'nom':'Espagne' , 'capitale' : 'Madrid' , 'population': 46439864 }, { 'nom':'Italie' , 'capitale' : 'Rome' , 'population': 60795612 }, { 'nom':'Royaume-Uni', 'capitale' :'Londres', 'population': 64767115 },] - calculer et afficher la population totale (319529060) .
ExoB7 (str , palindrome)	Ecrire un script b7.py qui va vérifier (et afficher) si oui ou non une chaine de caractères saisie est un palindrome (se lit dans les 2 sens à l'identique tel que kayak ou bob , radar, elle) . pour tester des phrases entières , on supprimera d'abord tous les espaces exemple : esope reste ici et se repose
ExoB8 (str)	nomFichiers=["fichierA.txt" , "configB.json" , "b8.py"] # boucler sur chaque nom complet de fichier # et afficher séparément nom et extension
ExoB9 (.format)	listeDicoDates =[{ 'jour' : 4 , 'mois' : 'avril' , 'annee' : 2024} ,

```
    { 'jour' : 12 , 'mois' : 'juin' , 'annee' : 2023} ,
    { 'jour' : 22 , 'mois' : 'octobre' , 'annee' : 2025} ,
]

# construire et afficher des dates sous forme de chaines (via .format() )
# récupérer et afficher la date d'aujourd'hui ( et l'heure aussi)
```

2.5. Fonction et appels , lambdas

Exo C1	Ecrire un script qui : - comporte une fonction volumeSphere() permettant de calculer le volume d'une sphere selon son rayon et la formule 4/3 * PI * rayon * rayon * rayon - appelle en boucle cette fonction a partir de rayons= [5.0 , 10, 100] - affiche les rayons et les volumes calculés

2.6. Exceptions , Fichiers et with

Exo D1	

2.7. Programmation orientée objet

Exo E1	Classe Personne avec (prenom,nom,anneeNaissance) et incrementerAge()
Exo E2	Classe Voiture avec (marque,modele,couleur, vitesse) et accelerer() , ralentir()
Exo E3	Classe Compte avec (numero,label,solde) et debiter() , crediter()
Exo E4	Sous classe Employe héritant de Compte et ajoutant (salaire , fonction)
...	

2.8. ….

Exo F1	

2.9. ….

Exo G1	