

langage python

Table des matières

I - Présentation du langage Python.....	7
1. Présentation du langage python.....	7
1.1. Utilisation et popularité de Python.....	7
1.2. Principales caractéristiques du langage python.....	8
1.3. Historique et évolution.....	9
1.4. Distributions de "python".....	9
II - Prise en main de l'interpréteur Python.....	10
1. Installation et première utilisation de python.....	10
1.1. Quelques installations possibles de python.....	10
1.2. Prise en main de l'interpréteur python.....	11
1.3. Ecriture et lancement d'un programme python.....	11
1.4. Instructions élémentaires du langage python	12
1.5. Commentaires multi-lignes et séparateur d'instructions.....	13
2. IDE (éditeur de code) pour python.....	14

2.1. Pycharm.....	14
2.2. Visual Studio Code avec extension python.....	15
3. Bonnes pratiques "PEP8".....	15

III - Syntaxes élémentaires , types , boucles..... 16

1. Types et syntaxes élémentaires de Python.....	16
1.1. Types de données en python 3.....	16
1.2. Bloc d'instructions et indentation.....	19
1.3. Tests et opérateurs logiques (comparaisons, ...).....	19
1.4. 4 grands types de Collections en python.....	23
1.5. Listes et fonction len() pour connaître la taille/longueur.....	23
1.6. range() , séquence et généralités sur les boucles.....	23
1.7. Boucle for (pour chaque élément de ...).....	24
1.8. Boucle while (tant que ...).....	27

IV - Structures de données (liste,dictionnaire,...)..... 29

1. Manipulation de listes en python.....	29
2. Set (ensembles).....	31
3. Dictionnaires (associations).....	32
4. Manipulation de chaînes de caractères.....	35
4.1. String / str.....	35
4.2. f-string (depuis python 3.6).....	37
4.3. Premier aperçu sur dates et math.....	37

V - Fonctions, lambda , modules , exceptions..... 38

1. Fonctions (python).....	38
1.1. Fonctions élémentaires.....	38
1.2. Fonctions avec paramètres positionnels.....	38
1.3. Lambda (fonction anonyme).....	39
1.4. Paramètres nommés et facultatifs sur fonctions.....	40
1.5. Fonction avec nombre d'arguments variables.....	41
2. Modules et packages (python).....	42
2.1. importations de fonctions (et autres éléments).....	42
2.2. packages de fichiers réutilisables (modules/librairies).....	43
2.3. Principaux modules prédéfinis du langage python.....	43
3. Calculs mathématiques élémentaires.....	44
3.1. Principales fonctions du Module math.....	44
3.2. Exemple (résolution d'équation du second degré).....	45
3.3. Module "random" pour nombres aléatoires.....	45
3.4. Module datetime.....	46
3.5. Module os.....	46
3.6. Module sys.....	46

4. Fonctions natives et lambdas.....	47
4.1. tris en python 3.....	47
4.2. Filtrages et transformations.....	48
5. Gestion des exceptions (python).....	48
5.1. plantage du programme sans traitement d'exception.....	49
5.2. avec traitement des exceptions (try / except).....	49
5.3. Principaux types d'exceptions prédéfinies de python.....	49
5.4. Lever une exception personnalisée (raise Exception).....	50
5.5. Syntaxe facultativement complète (try / except / finally).....	50
5.6. Variantes sur except :.....	50
5.7. Groupe d'exceptions (levée multiple):.....	51
5.8. Notes d'exceptions (enrichissement avant re-propagation).....	52
5.9. Contexte auto fermant avec with.....	53

VI - Gestion des fichiers (depuis python).....55

1. Gestion des fichiers en python.....	55
1.1. ouverture , lecture et écritures.....	55
1.2. avec fermeture automatique (with).....	56
1.3. fichiers, répertoires , chemins via os et os.path.....	56
1.4. fichiers, répertoires , chemins via shutil.....	57
2. Formats de fichiers et gestion en python.....	59
2.1. Accès directs/aléatoires (souvent au format binaire).....	59
2.2. Gestion du format csv.....	60
2.3. gestion du format json.....	61
2.4. gestion du format xml.....	63
3. Compression au format gzip via zlib.....	64
3.1. Compression de fichier via zlib.....	64
3.2. Décompression de fichier via zlib.....	64

VII - Python orienté objet.....65

1. Programmation orientée objet en python.....	65
1.1. Classe et instances.....	65
1.2. Héritage simple/ordinaire.....	66
1.3. Héritage multiple (rare).....	68
1.4. polymorphisme.....	69
1.5. classe avec attribut/variable de classe.....	70
1.6. classe abstraite (pas directement instanciable).....	70
1.7. avec @staticmethod.....	72
1.8. avec @classmethod.....	73
1.9. avec @property et .setter (encapsulation).....	73

VIII - Gestion des modules/packages (pip).....75

1. Modules et packages (approfondissement).....	75
1.1. Modules.....	75
1.2. Chargement des modules (résolutions).....	75
1.3. Packages.....	76
2. PIP & PipEnv.....	78
2.1. pip (niveau global / installation de python).....	78
2.2. venv (de niveau projet/application).....	79
2.3. pipenv (venv + pip).....	80

IX - Décorateurs, surcharge opérateurs,82

1. Aspects divers et avancés de python.....	82
1.1. unpack collection as function args.....	82
1.2. zip/mix n collections in one.....	82
1.3. global et nonlocal.....	83
1.4. Liste en comprehension.....	85
1.5. Surcharge d'opérateurs.....	85
1.6. Décorateurs.....	87
1.7. itérateurs et générateurs.....	90
1.8. expressions régulières.....	92

X - Threads et coroutines asynchrones.....95

1. Threads et exécution asynchrone en python.....	95
1.1. Démarrage d'un nouveau thread.....	96
1.2. Verrou (lock) sur données partagées.....	97
1.3. ThreadPoolExecutor et Futures.....	98
1.4. Threads producteurs/consommateurs : Queue.....	99
1.5. Autres mécanismes de synchronisation de threads.....	100
1.6. coroutines asynchrones (async/await) , python ≥ 3.5.....	100

XI - Affichages graphiques en python..... 103

1. Affichage graphiques en python.....	103
1.1. Présentation de tKinter.....	103
1.2. Hello world avec tkinter.....	103
1.3. exemple basic (counter) :.....	103
1.4. Exemple simple (calculatrice).....	105
1.5. Vue d'ensemble sur positionnement des widgets.....	106
1.6. Boîtes de dialogues de tKinter.....	108
1.7. Widgets fondamentaux (Label, Entry, ...).....	109
1.8. Widgets secondaires de tKinter (Spinbox, Scale, ...).....	110
1.9. Canvas tKinter (drawing area).....	111
1.10. Affichage d'image tKinter.....	111
1.11. Menu tKinter.....	112

XII - Calculs scientifiques / python..... 113

- 1. Calculs scientifiques en python..... 113
 - 1.1. installation et utilisation de numpy dans env python virtuel..... 113
 - 1.2. Calcul matriciel (algèbre linéaire) avec numpy..... 113

XIII - Accès aux bases de données..... 117

- 1. Accès aux bases de données en python..... 117
 - 1.1. Principaux modules d'accès aux bases de données..... 117
 - 1.2. accès à MySQL en python via mysql.connector..... 117
 - 1.3. accès SQL via SqlAlchemy..... 120
 - 1.4. accès "ORM" via SqlAlchemy..... 124
 - 1.5. accès à MongoDB en python..... 126

XIV - Http , web , api REST..... 127

- 1. http , web , api rest en python..... 127
 - 1.1. Vue d'ensemble sur frameworks "web" et "rest" en python..... 127
 - 1.2. Django (bases élémentaires)..... 128
 - 1.3. Api rest avec flask (simple , classique)..... 130
 - 1.4. Api rest avec fastapi (spécialisé api-rest, performant)..... 134

XV - sys , os , 137

- 1. programmation système en python..... 137
 - 1.1. Arguments sur ligne de commande..... 137
 - 1.2. Accès aux variables d'environnement..... 137
 - 1.3. Ecriture d'erreurs sur stderr* 137

XVI - Annexe – Test..... 138

- 1. Test unitaire python..... 138
 - 1.1. Installation du module pytest..... 138
 - 1.2. Exemple de test élémentaire..... 138
 - 1.3. Lancement d'un test unitaire simple..... 138

XVII - Annexe – interfaçage python et c/c++..... 139

- 1. Liaison entre python et langage c/c++..... 139
 - 1.1. Vue d'ensemble sur les alternatives possibles..... 139
 - 1.2. Exemple de librairie partagée écrite en langage C : 139
 - 1.3. Appels via ctypes (low-level standard python library)..... 143
 - 1.4. variante ctypes avec numPy..... 147
 - 1.5. Appels via CFFI (C Foreign Function Interface)..... 148

XVIII - Annexe – Bibliographie, Liens WEB + TP.....149

1. Bibliographie et liens vers sites "internet".....	149
2. TD/TP python.....	149
2.1. Installer si besoin python et VisualStudioCode ou pyCharm.....	149
2.2. Utilisation directe de python en mode interactif.....	149
2.3. Syntaxes élémentaires (variables , boucles , ...)......	149
2.4. Structures de données (list, set , dictionnaires).....	151
2.5. Fonction et appels , lambdas.....	153
2.6. Exceptions , Fichiers et with et bases de prog système.....	155
2.7. Programmation orientée objet.....	158
2.8. pip, venv, et aspects avancés (décorateurs, ...)......	160
2.9. Librairies graphiques et scientifiques (tkinter, numpy , ...)......	164
2.10. Accès aux bases de données depuis python.....	166
2.11. Http/web/restApi en python.....	167
2.12. Aspects avancés de python (thread, ...)......	167

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 :

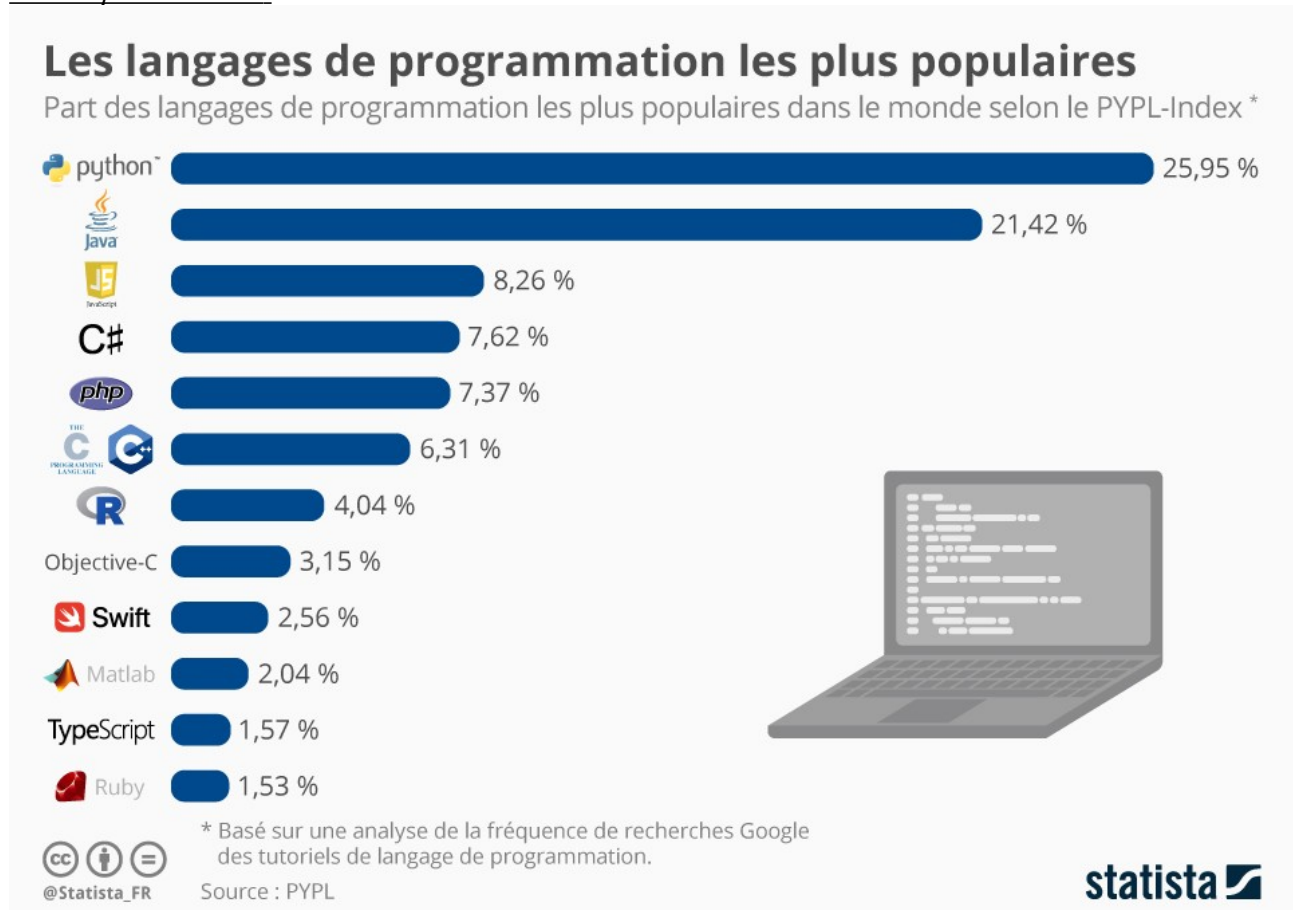
- **coder des petits scripts ou programmes simples**
- piloter/orchestrer des appels vers des fonctions prédéfinies efficaces (quelquefois codées en langage "C" de bas niveau et rapide)
- gérer des fichiers et/ou des grosses quantités de données (Big Data , Scrapping Web , ...)
- **piloter/orchestrer 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" , <i>avantages : souplesse et résultats immédiats</i> <i>inconvénients : exécution pas rapide (si 100% python)</i>
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 automatique 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 versions 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

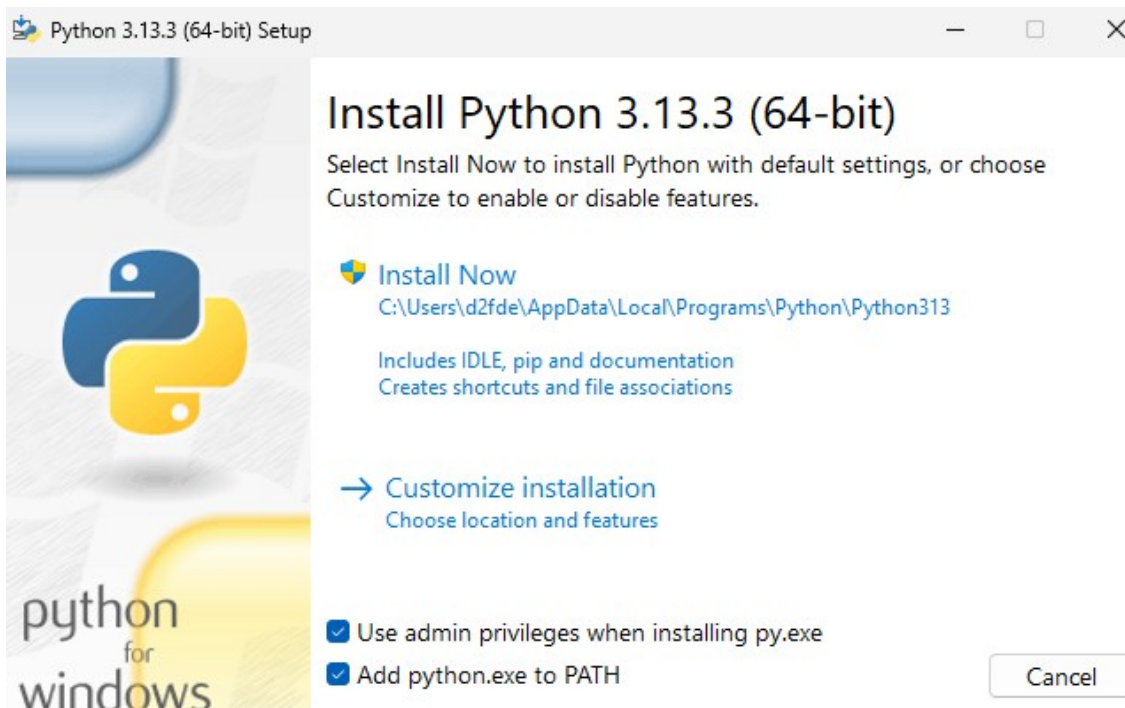
1. Installation et première utilisation de python

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 python 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'installateur 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 (notepad++, 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 chaîne 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

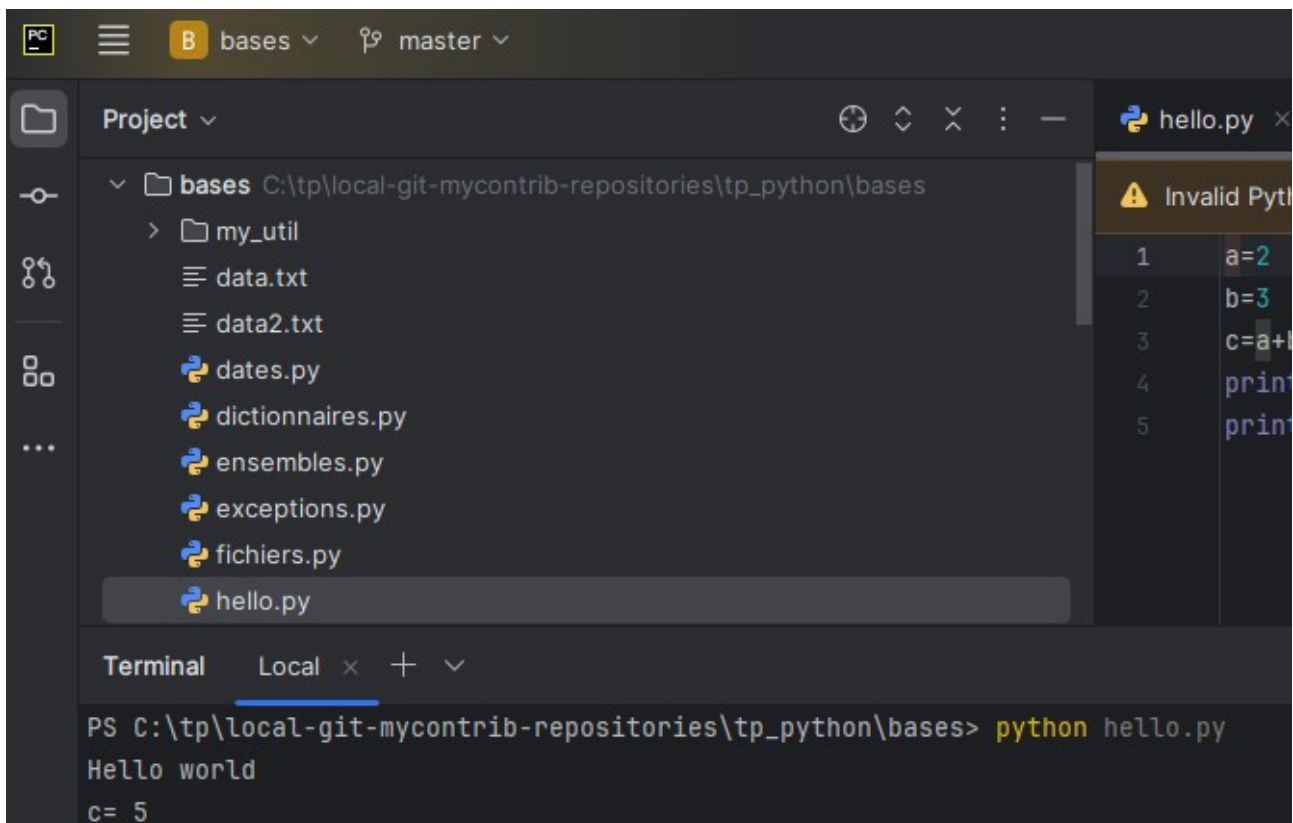
```
x=2 ; y=3 ; z=x*y ; print("z=x*y=",z) # ; est un séparateur d'instructions (sur même ligne)  
'''  
commentaire sur plusieurs lignes  
délimité par triple simple quote .  
'''  
print('suite')
```

2. IDE (éditeur de code) pour python

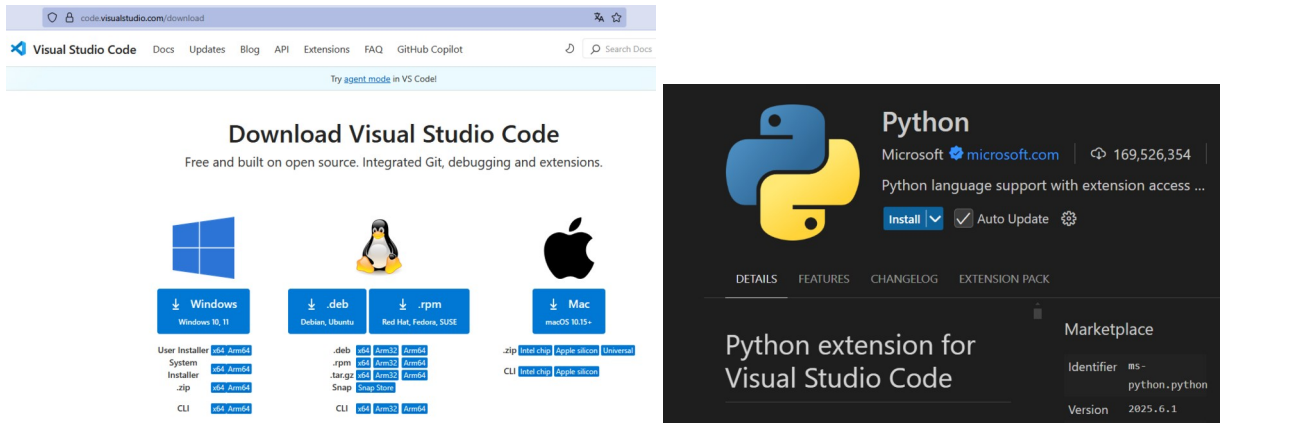
2.1. Pycharm



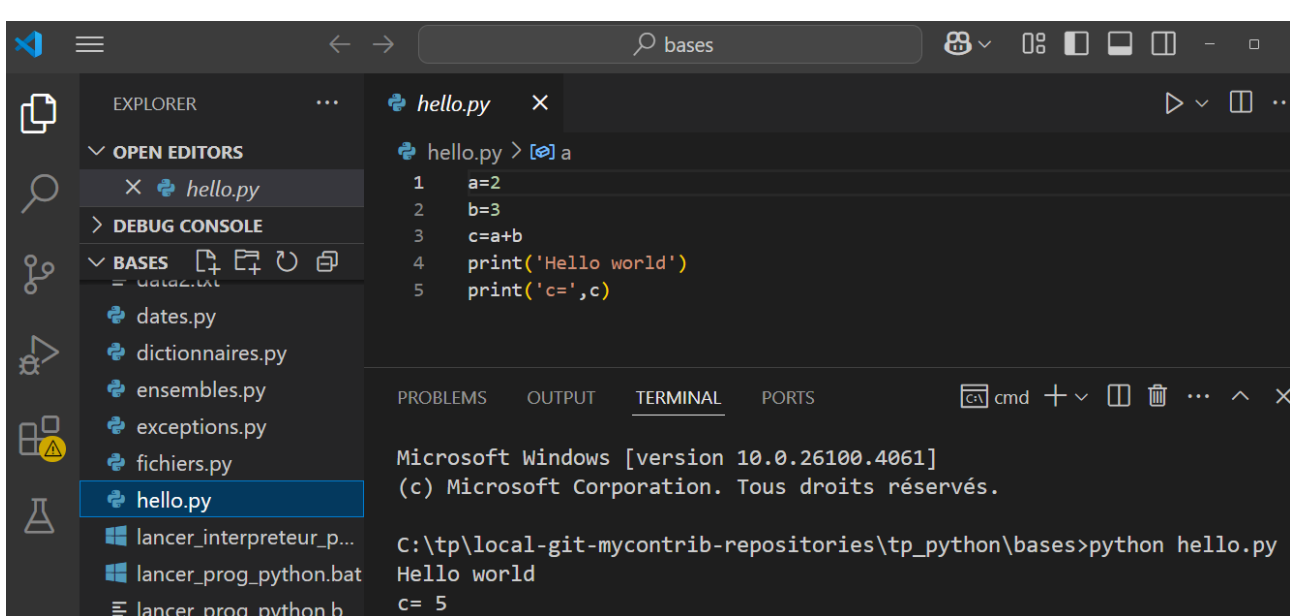
Bon Editeur python de l'entreprise JetBrains (comme IntelliJ, webstorm, ...)



2.2. Visual Studio Code avec extension python



The top part of the image shows the Visual Studio Code download page. It features the title "Download Visual Studio Code" and a list of download links for Windows, Linux (deb, rpm), and Mac. The bottom part shows the Python extension for Visual Studio Code in the marketplace, with details like the identifier "ms-python.python" and version "2025.6.1".



The bottom part of the image shows the Visual Studio Code interface. The Explorer sidebar on the left lists files like "hello.py", "dates.py", "dictionnaires.py", "ensembles.py", "exceptions.py", "fichiers.py", "lancer_interpreteur_p...", "lancer_prog_python.bat", and "lancer_prog_python.b...". The main editor shows a Python script "hello.py" with the following code:

```
1 a=2
2 b=3
3 c=a+b
4 print('Hello world')
5 print('c=',c)
```

The TERMINAL panel at the bottom shows the command prompt output:

```
Microsoft Windows [version 10.0.26100.4061]
(c) Microsoft Corporation. Tous droits réservés.

C:\tp\local-git-mycontrib-repositories\tp_python\bases>python hello.py
Hello world
c= 5
```

3. Bonnes pratiques "PEP8"

PEP signifie "Python Extension Proposal".

PEP8 correspond essentiellement à un ensemble de **bonne pratiques** dont les plus importantes sont les suivantes :

Encodage caractères du code source	UTF-8
Indentation	4 caractères espaces consécutifs
Longueur maxi idéale ligne de code	79 caractères par ligne, pas plus.
Import	Les import sont à déclarer au début du script . Prévoyez une ligne par import
Espaces	Pas d'espaces inutiles , juste pour gagner en lisibilité
Commentaires	Idéalement en anglais
Nom des variables et fonctions	snake_case : jour_du_mois plutôt que jourDuMois

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

<i>type</i>	<i>caractéristiques</i>	<i>exemples</i>
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'
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
liste_heterogene = [ 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'instructions 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
<pre>int factorielle(int n) { if (n < 2) { return 1; } else { return n * factorielle(n - 1); } }</pre>	<pre>def factorielle(n): if n < 2: return 1 else: return n * factorielle(n - 1)</pre>

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 âgée')
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 \geq 3.10
<pre>couleur = "vert" if couleur == "rouge" : print("red") elif couleur=="vert" : print("green") elif couleur=="bleu" : print("blue") else: print("other color")</pre>	<pre>couleur = "vert" match couleur: case "rouge" : print("red") case "vert" : print("green") case "bleu" : print("blue") case _ : print("other color / <i>default</i>")</pre>

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 indexé	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 **liste_xy[positionN]** permet d'accéder directement à un élément de la liste .

Exemples :

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

Syntaxes spéciales pour plage d'index :

```
print('plage 1 inclus à 3 exclus :' , liste_couleurs[1:3]) # ['vert', 'bleu']
print('plage début à 3 exclus :',liste_couleurs[:3]) # ['rouge', 'vert', 'bleu']
print('plage 2 inclus à fin :' ,liste_couleurs[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 élément de ...)

```
#indices_de0a3= 0,1,2,3
indices_de0a3= range(4) # range(4) construit 0, 1, 2, 3
for i in indices_de0a3 :
    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
```

```
jours_semaine = ['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 jours_semaine :
    print(jour,'est un element de jours_semaine')
print('suite apres la boucle')
```

==>

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

```
#nb.JoursDansUneSemaine=7
nb_jours_dans_semaine=len(jours_semaine) # la fonction len() retourne la taille/longueur
                                           #(length en anglais)
for i in range(nb_jours_dans_semaine) :
    print("i=",i,"jour=",jours_semaine[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]
premier_positif=None
for v in valeurs :
    if v >=0 :
        premier_positif=v
        break #fin anticipée de boucle (premier_positif déjà trouvé)
print("premier_positif=",premier_positif) # 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 déclenchée 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 .

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

```
liste_valeurs = []
valeur_ou_fin=input("val=")
while (valeur_ou_fin != 'fin' ) and (valeur_ou_fin != " ") :
    nouvelle_valeur = float(valeur_ou_fin)
    liste_valeurs.append(nouvelle_valeur)
    valeur_ou_fin=input("val=")
print('liste_valeurs=',liste_valeurs)
```

==>

```
val=4
val=6
val=
liste_valeurs= [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" ]
saisons_maj = []
for i in range(len(saisons)) :
    saisons_maj.append(saisons[i].upper())

print("saisons_maj=",saisons_maj)
```

saisons_maj= ['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.

```
liste_vide=[]

liste_initiale=[1,2,3]

liste=liste_initiale
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 ]
```

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

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

```
indice_b_dans_liste5= liste5.index('b')
print("indice_b_dans_liste5",indice_b_dans_liste5) # 1 (premier trouvé)

indice_c_dans_liste5= liste5.index('c')
print("indice_c_dans_liste5",indice_c_dans_liste5) # 4
```



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

```
autre_liste = [ 'a' , 'b' , 'c1' , 'd' , 'e' ]
autre_liste.insert(3,'c2') #insertion avant l'élément actuel d'indice 3
                        #insertion du nouvel élément en pos 3 et en décalant la fin
print('autre_liste apres insertion:', autre_liste)
#autre_liste 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 ]
ref_liste = liste6 # ref_liste référence la même liste que celle référencée par liste6
ref_liste[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 ]
```

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

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

mes_couleurs=";" .join(liste_couleurs) # transforme liste en chaîne de caractères
#attention : très différent de java/javascript (listeJs.join(";"))
print("mes_couleurs=",mes_couleurs) # 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)

```
ensemble_vide={}

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

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

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

```
for f in ensemble_fruits:
    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()*

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

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

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

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

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

```
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", }
liste_from_set1 = list(set1)
print("liste_from_set1=",liste_from_set1,type(liste_from_set1))
#liste_from_set1= ['b' 'c', 'a'] <class 'list'> # NB: ordre aléatoire/variable
```

```
tuple2 = ("a", "b", "c" )
liste_from_tuple2 = list(tuple2)
print("liste_from_tuple2",liste_from_tuple2,type(liste_from_tuple2))
# liste_from_tuple2 ['a', 'b', 'c'] <class 'list'>
```

```
l1 = ["a", "doublon", "b", "c", "doublon"]
set_from_l1 = set(l1)
print("set_from_l1",set_from_l1,type(set_from_l1))
# set_from_l1 {'b', 'c', 'doublon', 'a'} <class 'set'>
```

```
tuple1 = ("a", "doublon", "b", "c", "doublon")
set_from_tuple1 = set(tuple1)
print("set_from_tuple1 =", set_from_tuple1 ,type(set_from_tuple1))
# set_from_tuple1 = {'b', 'c', 'a', 'doublon'} <class 'set'>
```

3. Dictionnaires (associations)

```
dictionnaire_vide={ }
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") .

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

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

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

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

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

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

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

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

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

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

ou bien

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

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

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

```
dico_pers= { "nom" : "Bon" , "age" : 45 }  
print("nb associations=" , len(dico_pers)) # 2  
  
dico_duplique=dico_pers.copy(); #ou bien dico_duplique=dict(dico_pers);  
dico_duplique["age"]=30  
print(dico_pers) # { "nom" : "Bon" , "age" : 45 }  
print(dico_duplique) # { "nom" : "Bon" , "age" : 30 }  
  
dico_duplique.clear() # vide le contenu du dictionnaire
```

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

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

4. Manipulation de chaînes de caractères

4.1. String / str

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

dernier_caractere=s5[-1]
print(dernier_caractere) # 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

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

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

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

```
s6=" Ile De France "
s6_bis=s6.strip() #like trim of other language --> supprime espaces inutiles
                #au début ou à la fin
print(s6_bis) # "Ile De France"
```

```
s7="Mont Saint Michel"
s7_maj = s7.upper(); print(s7_maj) # MONT SAINT MICHEL
s7_min = s7.lower(); print(s7_min) # mont saint michel
```

```
s7="Mont Saint Michel"
s7_bis=s7.replace(' ','-') # replace substring with another string
print(s7_bis) # Mont-Saint-Michel
```

```
s8="partie1;partie2;partie3"
liste_parties=s8.split(';')
print(liste_parties) # ['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";
s12_bis=s12.capitalize() # transforme première lettre en Majuscule
print(s12_bis); # Dupond
```

```
file_name="p2.py"
dot_index = file_name.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 . =", dot_index) # 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 ")
```

4.2. 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
```

4.3. Premier aperçu sur dates et math

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 generer_message_salutation():
    message="bonjour"
    return message
```

#fonction/procédure basique ne retournant aucune valeur

#mais exécutant une action:

```
def saluer():
    salutation=generer_message_salutation() #appel de sous-fonction
    print(salutation)
```

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

1.2. Fonctions avec paramètres positionnels

#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
```

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

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

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

```
# le paramètre f_calcul est une référence sur une fonction de calcul à déclencher
# le paramètre f_prefixage est une référence sur une fonction de préfixage à déclencher
def enchaîner_calcul_et_affichage_avec_prefixage(x, f_calcul, f_prefixage):
    res = f_calcul(x)
    print(f_prefixage(res))

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

enchainer_calcul_et_affichage_avec_prefixage(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 a un nom , une valeur (renseignée ou bien par défaut) .

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

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

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

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

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

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

Vocabulaire important (au sein de python) :

- Au sein d'un appel de fonction python, les paramètres qui sont **passés (sans nom) et dans un ordre bien déterminé** sont appelés **paramètres positionnels** / "*positional argument*" (c'est le cas des paramètres qui n'ont *pas de valeur par défaut* et qui doivent absolument être renseigné lors d'un appel).
- Au sein d'un appel de fonction python, les paramètres qui sont **passés avec leurs noms (et dans un ordre quelconque)** sont appelés "**keyword arguments**" / "***kwargs*"
- Si un appel mixe des arguments positionnels et nommés, les paramètres positionnels doivent alors être passés en première(s) position(s) .

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 **kvars for keyValue args (c'est un double splat)

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

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

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

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

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

display_args(2,6,8) # a= 2, a= 6 , a= 8
print("somme_args=",somme_args(2,6,8)) # 16

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

print("somme_any_args=",somme_any_args(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 double_de(x):
    return 2*x

def moitie_de(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(double_de(x)) # affiche 24
print(moitie_de(x)) # affiche 6
```

ou bien (avec nom du module en préfixe)

my_app.py

```
import my_fct

# --> appels de my_fct.double_de(x) et my_fct.moitie_de(x)
x=12;
print(my_fct.double_de(x)) # affiche 24
print(my_fct.moitie_de(x)) # affiche 6
```

ou bien (avec préfixe renommé)

my_app.py

```
import my_fct as m

# --> appels de m.double_de(x) et m.moitie_de(x)
x=12;
print(m.double_de(x)) # affiche 24
print(m.moitie_de(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 triple_de(x):
    return 3*x

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

my_util/op4.py

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

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

my_app.py

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

x=12 ;
print(triple_de(x)) # affiche 36
print(quart_de(x)) # affiche 3.0
```

2.3. Principaux modules prédéfinis du langage python

<i>modules</i>	<i>fonctionnalités</i>
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
<code>math.sin(x)</code>	sinus avec x en radians
<code>math.cos(x)</code>	cosinus avec x en radians
<code>math.tan(x)</code>	tangente avec x en radians
<code>asin(x)</code> , <code>acos(x)</code> , <code>atan(x)</code>	arc sinux , arc cosinus , arc tangente
<code>math.radians(x)</code>	convertit un angle de degrés en radians
<code>math.degrees(x)</code>	convertit un angle de radians en degrés
<code>math.sqrt(x)</code>	racine carrée
<code>math.pow(x,y)</code>	élève x à la puissance y
<code>math.exp(x)</code>	calcule e puissance x
<code>math.log(x)</code>	calcul ln(x)
<code>math.hypot(x,y)</code>	longueur de l'hypoténuse = $\sqrt{x^2+y^2}$
<code>math.ceil(x)</code>	arrondi à l'entier au dessus : <code>ceil(5.5) = 6</code>
<code>math.floor(x)</code>	arrondi à l'entier en dessous : <code>floor(5.5) = 5</code>
<code>math.fabs(x)</code>	retourne la valeur absolue : <code>fabs(-7) = 7</code>
<code>math.gcd(a,b)</code>	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)) # 0.4999999999999999
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)) # 0.4999999999999999
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 resol_eq_2nd_degre(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)

resol_eq_2nd_degre(2,-9,-5); # x1=-0.5 et x2=5

resol_eq_2nd_degre(2,-1,-6); # x1=-1.5 et x2=2

resol_eq_2nd_degre(1,3,9/8); # x1=x2=4/4=0.75

resol_eq_2nd_degre(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

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

3.4. Module datetime

Principales classes du module datetime :

<code>datetime.date</code>	Dates (year,month,day)
<code>datetime.time</code>	Heures (hour,minute,second,microsecond + tzinfo)
<code>datetime.datetime</code>	Date et heure
<code>datetime.timedelta</code>	Duration = différence de temps
<code>datetime.tzinfo</code> et <code>.timezone</code>	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

Ce module sert essentiellement à manipuler des chemins vers des fichiers et répertoires
→ L'essentiel est décrit dans le chapitre "gestion des fichiers en python" .

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. 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']
```

```
liste_pers = [
    { 'nom' : 'Dupond' , 'age' : 23 },
    { 'nom' : 'Zorro' , 'age' : 44 },
    { 'nom' : 'Anatole' , 'age' : 66 },
    { 'nom' : 'Laurent' , 'age' : 25 }
]

print("liste_pers=",liste_pers)
liste_pers_triee_par_noms = sorted(listePers, key = lambda p : p['nom'])
print("liste_pers_triee_par_noms=",liste_pers_triee_par_noms)
liste_pers_triee_par_ages = sorted(listePers, key = lambda p : p['age'])
print("listePers_triee_par_ages=",liste_pers_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 se 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 trier et sur celle-ci, sont appelées automatiquement des méthodes de comparaisons de type `_lt_`, `_gt_`, `_eq_`, `_le_`, `_ge_`, `_ne_`.

4.2. 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. Principaux types d'exceptions prédéfinies de python

TypeError	Argument de type incorrect (int à la place de str, ...)
ValueError	Valeur incorrecte (ex : valeur négative pour calcul de racine carrée , "abc" pour conversion en float() , ...)
...	

5.4. Lever une exception personnalisée (raise Exception)

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

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

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

5.5. 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
```

5.6. Variantes sur except :

Possibilité de traiter de la même manière plusieurs types d'exception au sein d'un tuple :

```
... except (RuntimeError, TypeError, NameError):
...     pass
```

Plusieurs blocs "except TypeExceptionN as err :" en terminant par le plus vague Exception :

```
import sys

try:
    f = open('myfile.txt')
```

```

s = f.readline()
i = int(s.strip())
except OSError as err:
    print("OS error:", err)
except ValueError:
    print("Could not convert data to an integer.")
except Exception as err:
    print(f"Unexpected {err=}, {type(err)=}")
    raise # sans argument et dans un bloc except , raise sert à repropager l'exception

```

NB : il existe aussi

```

raise RuntimeError('erreur suite a ...') from err
#pour indiquer qu'une exception est la conséquence d'une autre (de plus bas niveau)

```

NB:

- Les classes d'exception de python sont liées entre elles par des relations d'héritage.
- **BaseException** (sommet hiérarchie) correspond à des **exceptions rattrapables ou pas**.
- **Exception** hérite de BaseException et correspond à **toutes les exceptions rattrapables**.

5.7. Groupe d'exceptions (levée multiple):

```

def f1():
    excs = [ OSError('erreur1'), SystemError('erreur2') ]
    raise ExceptionGroup('plusieurs problemes', excs)

```

```

#f1()
"""
ExceptionGroup: plusieurs problemes (2 sub-exceptions)
+-+----- 1 -----
| OSError: erreur1
+----- 2 -----
| SystemError: erreur2
+-----
"""

```

```

try:
    f1()
except Exception as e:
    print(f'exception rattrapée : {e} de type {type(e)}')

```

⇒ exception rattrapée : plusieurs problemes (2 sub-exceptions) de type <class 'ExceptionGroup'>

```

try:
    f1()
except* OSError as e:

```

```
print(f"au moins une OSError a eu lieu dans ce sous groupe filtré {e}")
except* SystemError as e:
    print(f"au moins une SystemError a eu lieu dans ce sous groupe filtré {e}")
```

5.8. Notes d'exceptions (enrichissement avant re-propagation)

```
def f2():
    raise TypeError('bad type')
```

#Enrichissement d'exception à retransmettre avec des notes
#ajoutées au niveau d'une fonction intermédiaire
#qui re-propage d'exception :

```
def f3() :
    try:
        f2()
    except Exception as e :
        e.add_note("f3 appelant f2 qui a planté")
        raise
```

```
#f3()
"""
TypeError: bad type
f3 appelant f2 qui a planté
"""
```

NB : sans try/except au niveau externe/appelant, la/les note(s) s'affichent automatiquement en cas d'exception.

Au sein d'un try/except , les notes ne s'affichent que via un code de ce genre :

```
print("(exception =" , err, err.__notes__ )
```

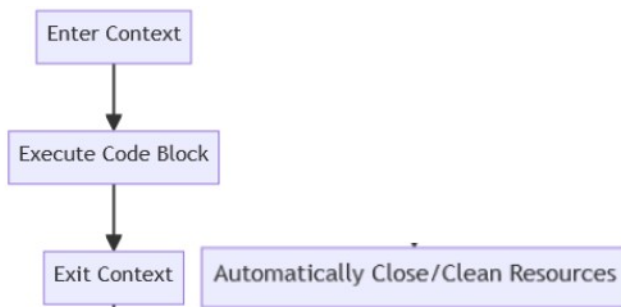

5.9. Contexte auto fermant avec with

```
with ContextA() as ca , ContextB() as cb , open(...) as f3 :
    ...
    ...
# automatic calls of __exit__  #(indirect close() or ...)
```

NB: ressemblant un peu au "try with resource" du langage java , l'instruction **with ... as ...** :

de python opère de manière fiable sur des éléments appelés "**contexte(s)**" comme des fichiers , des connexions à des bases de données ou bien n'importe quel autre objet comportant les méthodes

`__enter__` et `__exit__`



Ainsi sur un objet de type fichier, la méthode prédéfinie `__exit__` va automatiquement déclencher l'instruction `.close()` .

Exemple *custom_with.py* (pour la compréhension des mécanismes) :

```
import time

class MyTraceDump:
    def __init__(self, filename):
        self.filename = filename

    def __enter__(self):
        self.f = open(self.filename, "wt")
        return self

    def dump(self, trace):
        self.f.write(f'{trace}\n')

    def __exit__(self, exc_type, exc_value, traceback):
        self.f.close()

class ChronoContext:
    def __enter__(self):
        self.debut = time.time()
        return self
```

```
def __exit__(self, exc_type, exc_value, traceback):
    self.fin = time.time()
    self.duree = self.fin - self.debut
    print(f'elapsed time : {self.duree:.2f} s')
```

```
with ChronoContext() as chrono , MyTraceDump('my_traces.txt') as f_dump :
    # Bloc de code dont on souhaite mesurer le temps d'exécution via ChronoContext
    # et avec f_dump automatiquement ouvert et fermé:
    for i in range(10):
        time.sleep(0.1) # 0.1s de pause
        f_dump.dump(f'i={i}')
```

Résultat à la console :

elapsed time : 1.00 s # (10 fois 0.1 s)

Résultats dans mytraces.txt :

i=0

i=1

...

i=8

i=9

NB: Cet exemple sera encore plus compréhensible lorsque la programmation orientée objet aura été abordée au sein d'un chapitre ultérieur.

NB: Il est également possible de coder un contexte auto ouvrant/auto-fermant (s'utilisant dans un bloc with ... as) via le décorateur **@contextmanager** .

Les décorateurs seront ultérieurement vus dans le chapitre "syntaxes avancées"

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 :    # In Python, empty strings are considered as falsy
    ligneLue=f.readline() # returning empty string if EOF is reached
    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. fichiers, répertoires , chemins via os et os.path

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 (ex : shutil) pour bien manipuler le système de fichiers.

```
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>

Autres exemples classiques :

```
import os
repertoire = input("nom ou chemin de répertoire à analyser : ")
liste_contenu = os.listdir(repertoire)
for c in liste_contenu:
    chemin_complet=os.path.join(repertoire,relative_file_name)
```

```
if os.path.isfile(c): # or os.path.isdir(c)
    taille = os.path.getsize(c)
```

1.4. fichiers, répertoires , chemins via shutil

shutil est une librairie de haut niveau (mieux que os et os.path)

documentation officielle : <https://docs.python.org/3/library/shutil.html>

import **shutil**

shutil.copyfile('file1','file2') ou bien shutil.copy('file1','file2')	Copie de fichier (chemins absolus ou relatifs) mais sans recopier métadonnées (permissions, date, ...)
fh1 = open('myFile') fh2 = open('myFile2', 'w') shutil.copyfileobj(fh1, fh2) fh1.close(); fh2.close()	Copie de fichiers avec "file_handle"
shutil.copy('myFile', 'myDir')	Recopier un fichier dans un répertoire
shutil.copymode('file1','file2')	Copie des permissions (ex : rwx...)sans changer contenu
shutil.copystat('file1','file2')	Copie des permissions et dates/heures sans changer contenu
shutil.move('myDir', 'myDir2')	Renomme ou bien déplace un répertoire (ou fichier)
shutil.rmtree('myDir')	Détruit un répertoire et tout son contenu/arborescence
shutil.copytree('myDir', '/myDir1/myDir2')	Copie de branche entière avec création du sous répertoire myDir2 dans myDir1 si besoin
Option <i>symlinks</i> = <i>True</i> de copytree	Avec recopie des liens symboliques
Option <i>ignore</i> = <i>shutil.ignore_patterns('*.txt', '*.csv')</i> de copytree	En ignorant certains fichiers

shutil.make_archive('myDir', 'zip', '.', 'myDir')

permet la création d'une archive myDir.zip à partir des fichiers du répertoire courant "myDir" .

NB :

- le premier argument est le nom de l'archive, sans l'extension.
- le 2ème argument est le type d'archive.
- le 3ème argument est le chemin relatif de la (sous-)partie à archiver
- le 4ème argument est le répertoire à archiver.

Formats d'archives possibles :

'zip' , 'tar' , 'gztar' : pour les tar.gz et 'bztar' pour les tar.bz2

Exemple élémentaire :

```
import shutil
source_path = "example.xml"
destination_path = "example2.xml"

dest = shutil.copy(source_path, destination_path)
#shutil.copymode(source_path, destination_path)
```

```
# Print path of newly created file :  
print("Destination path:", dest) # example2.xml
```

2. Formats de fichiers et gestion en python

2.1. Accès directs/aléatoires (souvent au format binaire)

```
import struct

listOfFloat=[]
for i in range(10):
    listOfFloat.append(float(i)+float(i)/100)

print("listOfFloat",listOfFloat)
#[0.0, 1.01, 2.02, 3.03, 4.04, 5.05, 6.06, 7.07, 8.08, 9.09]

structBinaryFormat="!d" # ! for network (big indian) , d for double/float

#écriture de 10 "float/double" au format binaire dans le fichier data.bin
f=open("data.bin","wb")
for i in range(10):
    binary_float_value=struct.pack(structBinaryFormat,listOfFloat[i])
    #taille=len(binary_float_value) #size=8 for float as double
    #print(f"writing binary_float_value={binary_float_value} of size={taille}")
    f.write(binary_float_value)
f.close()

#####

#réouverture du fichier binaire data.bin et relecture d'une valeur sur 2
sublistOfFloat=[]
f2=open("data.bin","rb")
taille=8 # taille d'une valeur (ou enregistrement) binaire à lire
for i in range(10):
    if i%2==0:
        f2.seek(taille*i)
        binary_float_value = f2.read(taille)
        #print(f"read binary_float_value={binary_float_value}")
        float_valueTuple=struct.unpack(structBinaryFormat,binary_float_value)
        sublistOfFloat.append(float_valueTuple[0])
f2.close()
print("sublistOfFloat",sublistOfFloat) #[0.0, 2.02, 4.04, 6.06, 8.08]
```

Points clefs:

- **f.seek(position)** permet de se positionner à une position bien précise au sein d'un fichier en accès direct/aléatoire de manière à préparer la prochaine lecture ou écriture .
- **struct.pack(format,valeur(s))** permet de convertir des valeurs python (str,int,float,...) en valeurs binaires (bytes). On peut convertir d'un seul coup un tableau entier de valeurs.
- **struct.unpack(format,valeur(s))** effectue la conversion inverse et renvoie un tuple de une ou plusieurs valeurs binaires converties en python.

2.2. Gestion du format csv

CSV (Comma Separated Values) est un format très classique de fichier.

Le module python csv permet de lire ou écrire simplement des fichiers au format "csv" .

Exemple : write_read_csv.py

```
import csv

file_name="produits.csv"
field_names = ["ref","label","prix"]

data_dict_list = [
    { 'ref':1 , 'label' : "cahier" , 'prix' : 4.2},
    { 'ref':2 , 'label' : "stylo" , 'prix' : 2.2},
    { 'ref':3 , 'label' : "classeur" , 'prix' : 5.2},
    { 'ref':4 , 'label' : "gomme" , 'prix' : 3.2}
]

with open(file_name, 'w', newline="") as csvfile:
    # Creating a CSV DictWriter object
    writer = csv.DictWriter(csvfile, fieldnames=field_names , delimiter=";")

    writer.writeheader()# Writing headers (field names)
    writer.writerows(data_dict_list)# Writing data rows
```

⇒produits.csv

```
ref;label;prix
1;cahier;4.2
2;stylo;2.2
3;classeur;5.2
4;gomme;3.2
```

Relecture :

```
# Opening the CSV file
with open(file_name, mode='r') as file_to_read:
    # Reading the CSV file
    csv_reader = csv.reader(file_to_read, delimiter=";")

    # Skipping the header (uncomment or comment if needed)
    next(csv_reader)

    # Displaying the contents of the CSV file
    for line in csv_reader:
        print(line)
```

⇒

```
['1', 'cahier', '4.2']
['2', 'stylo', '2.2']
['3', 'classeur', '5.2']
['4', 'gomme', '3.2']
```


2.3. 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

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

#p1AsJsonString = json.dumps(personnel);
p1AsJsonString = json.dumps(personnel,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'>
```

2.4. gestion du format xml

Il existe plusieurs librairies python capables de gérer des fichiers au format xml :

xml.etree.ElementTree	Simple et efficace , module standard (pas besoin de pip install)
lxml	Version encore plus performante (mais à installer en plus)
xml.dom.minidom	Respect de l'api DOM
xml.sax	Pour lecture au fil de l'eau de très gros fichiers
BeautifulSoup	Gère le format XML (bien ou mal formé) et aussi HTML, pas rapide
untangle	Conversion entre Xml et objet python , pas rapide

example.xml

```
<pays name="France">
  <ville name="Paris" population="2133111" />  <ville name="Lyon" population="522250" />
  <ville name="Lille" population="236710" />  <ville name="Bordeaux" population="261804" />
  <ville name="Toulouse" population="504078" />  <ville name="Marseille" population="873076" />
</pays>
```

Exemple simple avec api ElementTree (as ET) :

read_xml.py

```
import xml.etree.ElementTree as ET

# Parse the XML file
tree = ET.parse('example.xml')

# Get the root element
root = tree.getroot()

# Iterate through each child of the root element
for child in root:
    # Print the tag and attributes of each child element
    print(child.tag, child.attrib)
```

⇒

```
ville {'name': 'Paris', 'population': '2133111'}      ville {'name': 'Lyon', 'population': '522250'}
ville {'name': 'Lille', 'population': '236710'}      ville {'name': 'Bordeaux', 'population': '261804'}
ville {'name': 'Toulouse', 'population': '504078'}   ville {'name': 'Marseille', 'population': '873076'}
```







En génération/écriture :

```
root=ET.Element("xys") # <xys> ....</xys>
tree=ET.ElementTree(root)

for xy in xy_list :
    element=ET.Element("xy")
    acteurElement.set("a",xy['a'])
    acteurElement.set("b", xy['b'])
    root.append(element) # <xy a="valeur_a" b="valeur_b" />

with open("my_output_file.xml","wb") as f:
    tree.write(f, encoding='utf-8')
```

3. Compression au format gzip via zlib

 compress_file.py	06/06/2025 12:29	Python File	1 Ko
 image1.tiff	06/06/2025 12:28	Fichier TIFF	1 106 Ko
 image1.tiff.gz	06/06/2025 12:37	Dossier d'archive ...	562 Ko
 image1copy1.tiff	06/06/2025 12:37	Fichier TIFF	1 106 Ko
 image1copy1.tiff.gz	06/06/2025 12:29	Dossier d'archive ...	562 Ko
 uncompress_file.py	06/06/2025 12:36	Python File	1 Ko

Exemple: une image .tiff (pas compressée) peut être compressée en un fichier .tiff.gz deux fois plus petit .

3.1. Compression de fichier via zlib

compress_file.py

```
import zlib

input_file_name='image1.tiff'
output_file_name=input_file_name+".gz"

original_data = open(input_file_name, 'rb').read()
compressed_data = zlib.compress(original_data, zlib.Z_BEST_COMPRESSION)

compress_ratio = (float(len(original_data)) - float(len(compressed_data))) /
float(len(original_data))

print('Compressed: %d%%' % (100.0 * compress_ratio))

f = open(output_file_name, 'wb')
f.write(compressed_data)
f.close()
```

⇒ Compressed: 49%

3.2. Décompression de fichier via zlib

uncompress_file.py

```
import zlib

output_file_name='image1copy1.tiff'
input_file_name=output_file_name+".gz"

compressed_data = open(input_file_name, 'rb').read()
decompressed_data = zlib.decompress(compressed_data)

f = open(output_file_name, 'wb')
f.write(decompressed_data)
f.close()
```

VII - Python orienté objet

1. Programmation orientée objet en python

1.1. Classe et instances

En langage python, pas de mot 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'CompteEpargne 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, CompteEpargne num=1 label=compteEpargne1 solde=50.0 avec tauxInteret = 1.25
ce2, CompteEpargne num=2 label=compteEpargne2 solde=100.0 avec tauxInteret = 1.25
ce1, CompteEpargne num=1 label=compteEpargne1 solde=50.0 avec tauxInteret = 2.1
ce2, CompteEpargne 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. Modules et packages (approfondissement)

1.1. Modules

Python est avant tout un langage interprété et dans les cas les plus simples, un seul fichier est interprété/exécuté, c'est un scriptlet.

Dès qu'un programme est plus complexe, le script de démarrage s'appuie alors sur d'autres fichiers complémentaires appelés modules. → En python, **un fichier = un module**.

Le fichier/script de démarrage correspond au **module principal** dont le nom (valeur de `__name__`) vaut `__main__`.

1.2. Chargement des modules (résolutions)

Lorsque python interprète une ligne de type "import modxxx" il va rechercher le module/fichier xxx de la façon suivante :

1. dans la liste des modules prédéfinis ([sys.builtin_module_names](#))
2. dans la liste des répertoires à scruter définie par la variable `sys.path`.

La variable `sys.path` comporte :

- le **répertoire courant** (là où est lancée l'application)
- tous les répertoires de la variable d'environnement `PYTHONPATH` (même syntaxe que `PATH`)
- la valeur par défaut liée à l'installation de python (dossier "site-packages" du module site)
ex : `C:\Users\d2fde\AppData\Local\Programs\Python\Python313\Lib\site-packages` ou autres

Chargement automatiquement optimisé :

Pour accélérer le chargement des modules, Python garde en cache une version compilée de chaque module dans un fichier nommé `module(version).pyc` (ou `version` représente le format du fichier compilé, typiquement une version de Python) dans le dossier `__pycache__`.

Par exemple, avec CPython 3.13, la version compilée de `xyz.py` serait `__pycache__/xyz.cpython-33.pyc`

Ce cache est automatiquement invalidé/remplacé si la date de la version en cache est antérieure à la version source.

NB : `dir()` ou `dir(moduleXyz)` affiche la liste des noms des éléments d'un module.

1.3. Packages

Un **paquet (package)** est un **ensemble de modules** (ayant souvent une **structure arborescente**)
(exemple : `p1.m1.aa` , `p1.m1.bb` , `p1.m2.xx` , `p1.m2.yy`)

Structure arborescente d'un package python :

```
p1/
  __init__.py
  m1/
    __init__.py
    aa.py
    bb.py
  m2/
    __init__.py
    xx.py
    yy.py
```

NB: le fichier `__init__.py` (pouvant éventuellement resté vide dans les cas les plus simples) permet à l'interpréteur de bien considérer votre package comme un package (sans cela on risquerait une collision de nom entre `nomPackage` et autre nom).

Variantes d'importation :

```
import p1.m1.aa
p1.m1.aa.aa1()
```

```
import p1.m1.aa as pmla
pmla.aa1()
```

```
from p1.m1.aa import aa1 # or from p1.m1.aa import *
aa1()
```

```
from p1.m1 import aa
aa.aa1()
```

`__all__` pour configurer le comportement de `import *` sur sous-package :

par défaut , l'écriture suivante ne fonctionne pas :

```
from p1.m1 import *
aa.aa1()
```

il faut placer

```
__all__ = [ "aa" , "bb" ]
```

dans `p1.m1.__init__.py`
pour que cela puisse fonctionner .

Egalement possible : si le fichier p1.m2. `__init__.py` comporte

```
from .xx import *
```

alors on peut écrire

```
from p1.m2 import xx1  
xx1()
```

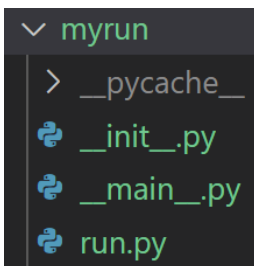
comme simplification de

```
from p1.m2.xx import xx1  
xx1()
```

NB: si un fichier `__init__.py` comporte **certaines instructions d'initialisation** alors **elles ne seront lancées qu'une seule fois (lors du premier chargement du module en mémoire)**.

Le fichier spécial et facultatif `__main__.py` peut éventuellement être placé dans un package/module exécutable via l'option `-m` de la **ligne de commande** python .

Exemple :



run.py

```
def print_maj(message):  
    print(message.upper())
```

myrun/ `__main__.py`

```
from .run import print_maj  
import sys  
print_maj(f'my_run with sys.argv[1]={sys.argv[1]}')
```

```
python -m myrun aa
```

⇒

```
MY_RUN WITH SYS.ARGV[1]=AA
```

NB: Le fichier `__main__.py` est par exemple utilisé dans le code tournant autour de **python -m zipapp** permettant de packager/livrer/executer du code python sous forme d'archive .zip (un peu comme les ".jar" du langage java).

2. PIP & PipEnv

2.1. pip (niveau global / installation de python)

pip = package installer for python

NB: l'ancien outil "easy_install" (datant de 2004) est maintenant obsolète .

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 3.13.3

pip --version

pip 25.0.1 from C:\Users\d2fde\AppData\Local\Programs\Python\Python313\Lib\site-packages\pip (python 3.13)

Utilisation de pip :

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

pip uninstall nomPackage

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) .

Eventuel enregistrement de packages nécessaires à installer :

pip freeze > requirements.txt

à peut être lancer dans un contexte pipenv shell ou autre ...

requirements.txt

mysql-connector-python==9.3.0

...

pytest==8.3.5

...

virtualenv==20.31.2

pip install -r requirements.txt

2.2. venv (de niveau projet/application)

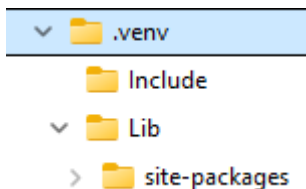
Les différentes applications "python" n'ont pas toutes les mêmes besoins (en termes de librairies et en termes de versions) .

venv = virtual **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* .

Concrètement, un environnement python virtuel correspond souvent à un sous répertoire **.venv** comportant essentiellement le sous répertoire **Lib/site-packages** comportant lui même la liste des packages ultérieurement téléchargés.



Exemple d'utilisation directe (de bas niveau) de venv (sans pipenv) sous linux :

```

#create virtualenv:
python -m venv .venv

#activate virtualenv:
. .venv/bin/activate

#ou bien source .venv/bin/activate
#ou bien (sous windows) .venv\Scripts\activate
# => début du prompt (.venv)

#use pip in venv :
python -m pip install -r requirements.txt
#sudo python -m pip install matplotlib

#run app
python ex_matplotlib.py

#sortir de venv :
# => fin du prompt (.venv)
deactivate
  
```

Via pipenv (combinant venv et pip , présenté ci-après) les choses seront plus simples ...

2.3. pipenv (venv + pip)

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

```
python -m pipenv --version
```

pipenv, version 2025.0.3

ou bien *pipenv --version* (en ajoutant C:\Users\...\AppData\Roaming\Python\Python313\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***

NB: par défaut, le répertoire de l'environnement virtuel est un sous répertoire de \$HOME (exemple : C:\Users\d2fde\virtualenvs\tkinter_app-4w_795AK)

Eventuelle possibilité de lancer un sous shell au sein de l'environnement virtuel créé via pipenv :

```
python -m pipenv shell
```

*#ou bien **pipenv shell***

Launching subshell in virtual environment...

Microsoft Windows [version 10.0.19041.804]

(c) 2020 Microsoft Corporation. Tous droits réservés.

(tkinter_app-4w_795AK) D:\tp\tp_python\tkinter_app>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
(tkinter_app-4w_795AK) D:\tp\tp_python\tkinter_app> python ex_matplotlib.py
(tkinter_app-4w_795AK) D:\tp\tp_python\tkinter_app> exit
```

Lancement d'une seule commande au sein de l'environnement virtuel :

```
python -m pipenv run ex_matplotlib.py
```

ou bien

```
pipenv run ex_matplotlib.py
```

IX - Décorateurs, surcharge opérateurs, ...

1. Aspects divers et avancés de python

1.1. unpack collection as function args

```
def myFunction(username="",password="",role=""):
    print(f'myFunction: username={username} password={password} role={role}')

myFunction() #with default values
myFunction("titi", "pwd1", "user") #with explicit positionnal args
myFunction(username="titi", password="pwd1", role="user") #with explicit keyword args
```

⇒

```
myFunction: username= password= role=
myFunction: username=titi password=pwd1 role=user
myFunction: username=titi password=pwd1 role=user
```

```
myOrderedParamList=[ 'tata', 'pwd2', 'admin']

#unpack myOrderedParamList into ordered positional args of myFunction :
myFunction(*myOrderedParamList)
```

⇒

```
myFunction: username=tata password=pwd2 role=admin
```

```
myParamDict={
    'username': 'toto',
    'password': 'pwd3',
    'role': 'admin'
}

#unpack myParamDict into kw-args of myFunction :
myFunction(**myParamDict)
```

⇒

```
myFunction: username=toto password=pwd3 role=admin
```

1.2. zip/mix n collections in one

```
keys=["red","green","blue"]
values=["rouge","vert","bleu","black"]
col3=['un','deux','trois','quatre']

#zipObject_as_iteratorOnTuple = zip(keys,values)
#colorDict = dict(zipObject_as_iteratorOnTuple)

colorDict = dict(zip(keys,values))
print("colorDict=" + str(colorDict))
```

⇒

```
colorDict={'red': 'rouge', 'green': 'vert', 'blue': 'bleu'}
```

```
experimentalTupleList = list(zip(keys,values,col3))
print("experimentalTupleList=" + str(experimentalTupleList))
```

⇒

```
experimentalTupleList=[('red', 'rouge', 'un'), ('green', 'vert', 'deux'), ('blue', 'bleu', 'trois')]
```

```
#colorDict2 = {keys[i] : values[i] for i, v in enumerate(keys)}
colorDict2 = {keys[i] : values[i] for i, _ in enumerate(keys)} # _ is for unused special variable name
print("colorDict2=" + str(colorDict2))
```

⇒

```
colorDict2={'red': 'rouge', 'green': 'vert', 'blue': 'bleu'}
```

1.3. global et nonlocal

```
g1=4
var_ambigue=2

def ma_fonction():
    l1=3
    print(f'dans ma_fonction, l1={l1} et g1={g1}')
    # g1=g1+1 ne fonctionne pas ici car python considère ce g1
    # comme un autre g1 (de niveau local) qui a par hasard le même nom que le g1 global
    """
    Message d'erreur exact:
    UnboundLocalError: cannot access local variable 'g1'
    where it is not associated with a value
    """
    # attention: le comportement de python n'est ici pas le même que celui de javascript
    var_ambigue=12 # cette variable locale là a le même nom que la variable globale
    #du haut --> 2 cases mémoires distinctes avec même nom, c'est ambigu (pas bien)!!!
    print(f'dans ma_fonction, l1={l1} et g1={g1} et var_ambigue={var_ambigue}')

def ma_fonction2():
    global g1 # le mot clef global permet de déclarer que g1 est ici le nom d'une variable globale
    l2=3
    print(f'dans ma_fonction2, l2={l2} et g1={g1}')
    g1=g1+1 # g1 est considéré ici (et jusqu'à la fin de cette fonction) comme une variable globale
    # le mot clef global est donc LE MOYEN de modifier si besoin une variable globale primitive
    print(f'dans ma_fonction2, après incrementation de g1, l2={l2} et g1={g1}')

def ma_fonction3():
    global g2
    g2=3

ma_fonction()
print(f'au niveau principal, après appel à ma_fonction, g1={g1} et var_ambigue={var_ambigue}')
ma_fonction2()
print(f'au niveau principal, après appel à ma_fonction2, g1={g1}')
```

```
ma_fonction3()
print(f'au niveau principal,après appel à ma_fonction3, g2={g2}')
```

⇒

dans ma_fonction, l1=3 et g1=4
dans ma_fonction, l1=3 et g1=4 et var_ambigue=12
au niveau principal,après appel à ma_fonction, g1=4 et var_ambigue=2
dans ma_fonction2, l2=3 et g1=4
dans ma_fonction2,après incrementation de g1, l2=3 et g1=5
au niveau principal,après appel à ma_fonction2, g1=5
au niveau principal,après appel à ma_fonction3, g2=3

```
class MySimpleCounter() :
    def __init__(self,val=0) :
        self.val=val

    def __str__(self) :
        return "MySimpleCounter(val="+str(self.val)+ ")"

    def increment(self) :
        self.val+=1

    def decrement(self) :
        self.val-=1
#####
globalCounter1 = MySimpleCounter(1)

def ma_fonction4():
    globalCounter1.increment()
    globalCounter1.increment()
    # ce n'est pas utile d'utiliser le mot clef global car globalCounter1 est une instance d'une classe
    # et on ne fait que modifier une des valeurs internes de l'objet
    print(f'dans ma_fonction4,après deux appels à increment globalCounter1={globalCounter1}')

print(f'au niveau principal,avant appel à ma_fonction4, globalCounter1={globalCounter1}')
ma_fonction4()
print(f'au niveau principal,après appel à ma_fonction4, globalCounter1={globalCounter1}')
```

⇒

au niveau principal,avant appel à ma_fonction4, globalCounter1=MySimpleCounter(val=1)
dans ma_fonction4,après deux appels à increment globalCounter1=MySimpleCounter(val=3)
au niveau principal,après appel à ma_fonction4, globalCounter1=MySimpleCounter(val=3)

pour cas très pointu seulement, nonlocal permet de déclarer dans une sous fonction imbriquée
qu'une variable n'est pas locale mais qu'elle est liée à une fonction englobante

```
def ma_fonction5():
    x=1
    y=2
    print(f'au debut ma_fonction5, x={x} y={y}')
    def inner_function():
        x=3
        nonlocal y
        y=3
        print(f'dans ma_fonction5.inner_function, x={x} y={y}')
    inner_function() # appel direct de inner_function
    print(f'a la fin de ma_fonction5, x={x} y={y}')
```

```
ma_fonction5()
```


⇒ au début de `ma_fonction5`, `x=1` `y=2`
 dans `ma_fonction5.inner_function`, `x=3` `y=3`
 à la fin de `ma_fonction5`, `x=1` `y=3`

1.4. 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)]
```

1.5. 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
-	<code>__neg__(self)</code>
+	<code>__pos__(self)</code>
~	<code>__invert__(self)</code>

1.6. 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'exécution de la fonction originale
Hello !
> après l'exécution de la fonction originale

Décorateur pour fonction avec arguments :

```
def basic_decorateur2(func):  
    def wrapperFunction(*args,**kwargs):  
        print("> avant l'exécution de la fonction avec args")  
        res=func(*args,**kwargs)  
        print("> après l'exécution 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'exécution de la fonction avec args
Hello *Laurence* !
> après l'exécution 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'exécution: {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'exécution: 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'exécution: 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 .

1.7. 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 retourne 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)
```

1.8. 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)

Répétitions 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 moins 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 ...

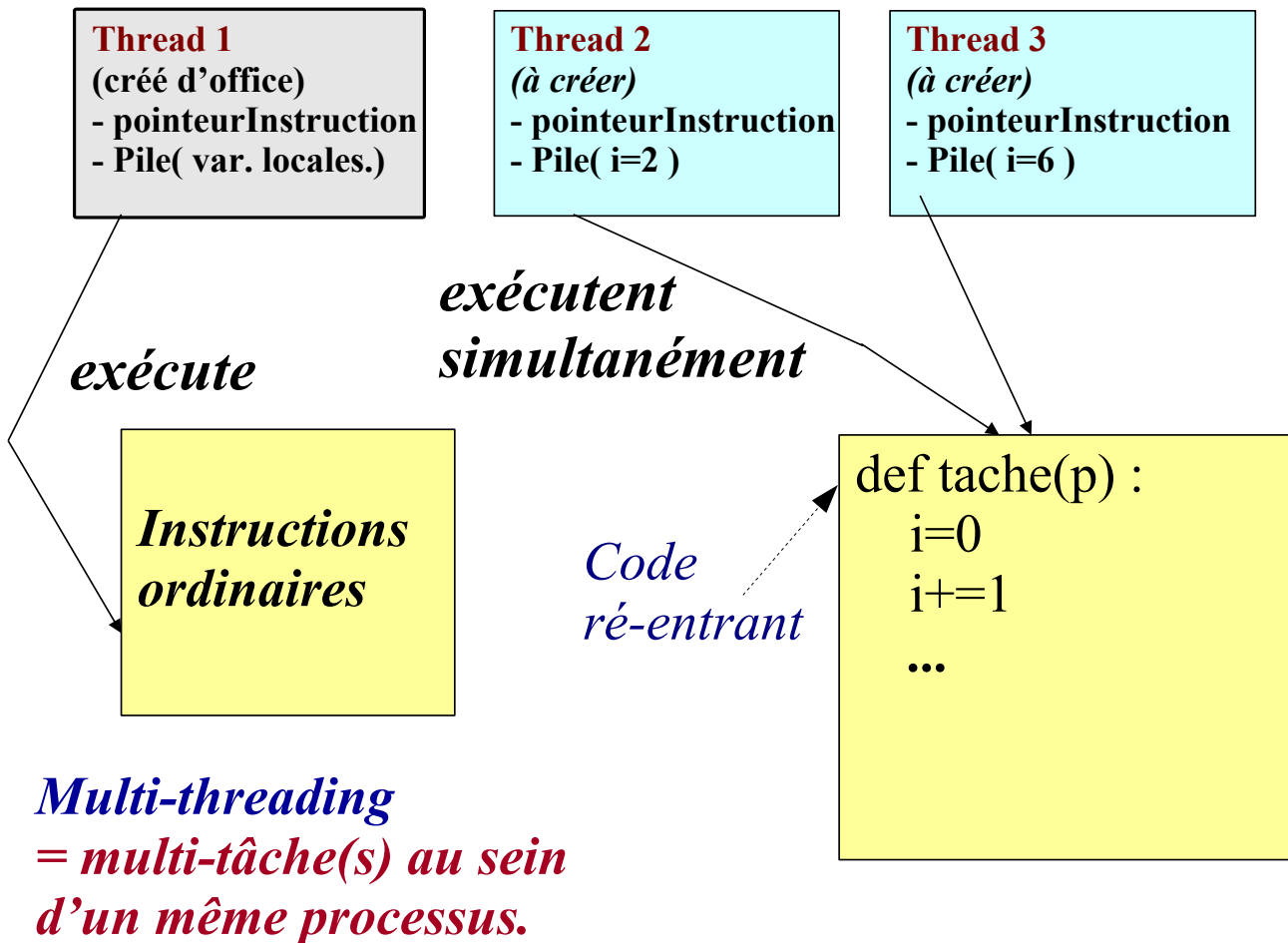
ANNEXES

X - Threads et coroutines asynchrones

1. Threads et exécution asynchrone en python

Un **Thread** (ou **fil d'exécution**) est une **entité qui exécute du code**. Un thread python délègue des exécutions d'instructions à un thread du système d'exploitation (ex : linux ou windows).

En démarrant plusieurs thread on peut effectuer des traitements en parallèle et quelquefois mieux exploiter la puissance d'un micro-processeurs ayant plusieurs coeurs .



1.1. Démarrage d'un nouveau thread

```
import threading
import time;

def tache1():
    print(">>> tache1: préparer du café")
    time.sleep(4) #4s de pause
    print(">>> tache1: le café est prêt")

# Création d'un nouveau thread secondaire
mon_thread = threading.Thread(target=tache1)
mon_thread.start() # démarrage du thread secondaire

#actions effectuées en // par le thread principal:
print("action1 effectuée par le thread principal")
time.sleep(1) #1s de pause
print("action2 effectuée par le thread principal")

# Attente de la fin de l'execution du thread secondaire
mon_thread.join()

print("fin de l'ensemble de l'application python")
```

⇒

```
>>> tache1: préparer du café
action1 effectuée par le thread principal
action2 effectuée par le thread principal
>>> tache1: le café est prêt
fin de l'ensemble de l'application python
```

1.2. Verrou (lock) sur données partagées

NB:

- Les **différents threads gérés par python** sont **supervisés** par le **GIL** (Global Interpreter Lock). Ce système gère la concurrence d'accès et la synchronisation des threads.
- Lorsque différents threads exécutent simultanément de code de certaines fonctions, les variables locales, les paramètres d'entrées et les valeurs de retour sont dans des zones mémoires différentes (pile de chacun des Threads).
- Certaines variables globales (ou autres) sont par contre en accès partagées entre tous les threads et il faut alors prendre des précautions pour que les modifications et lectures de valeurs ne soient pas effectuées anarchiquement par plusieurs threads en même temps.

t2 lock.py

```
import threading
import time;

compteur_partage=0
verrou_global = threading.Lock()

def incrementer(num_thread):
    compteur_local=0
    global compteur_partage
    for _ in range(10):
        with verrou_global :
            compteur_local +=1
            compteur_partage +=1
    print(f'compteur_partage={compteur_partage} venant d'etre incrementé par t{num_thread} , compteur_local={compteur_local}')

# Création de plusieurs threads
mes_threads = [threading.Thread(target= incrementer, args=[num_t+1]) for num_t in range(5)]

for t in mes_threads :
    t.start() # démarrage des threads secondaires
for t in mes_threads :
    t.join() # Attente de la fin de l'execution d'un thread secondaire

print("fin de l'ensemble de l'application python, compteur_partage=",compteur_partage) # 50
=>
```

```
compteur_partage=1 venant d'etre incrementé par t1 , compteur_local=1 ...
compteur_partage=9 venant d'etre incrementé par t1 , compteur_local=9 ...
compteur_partage=14 venant d'etre incrementé par t2 , compteur_local=5
compteur_partage=15 venant d'etre incrementé par t3 , compteur_local=1
compteur_partage=16 venant d'etre incrementé par t3 , compteur_local=2
compteur_partage=17 venant d'etre incrementé par t4 , compteur_local=1
compteur_partage=18 venant d'etre incrementé par t4 , compteur_local=2
compteur_partage=19 venant d'etre incrementé par t4 , compteur_local=3
compteur_partage=20 venant d'etre incrementé par t5 , compteur_local=1 ...
compteur_partage=50 venant d'etre incrementé par t3 , compteur_local=10
fin de l'ensemble de l'application python, compteur_partage= 50
```

1.3. ThreadPoolExecutor et Futures

La classe **ThreadPoolExecutor** du module **concurrent.futures** simplifie la gestion des threads en déclenchant automatiquement des exécutions de manière asynchrone en s'appuyant sur un pool de threads réutilisables et en s'occupant de la répartition des tâches .

t3_executor_submit.py

```
from concurrent.futures import ThreadPoolExecutor
import time
import threading

def tache_longue(duree):
    curr_thread = f'thread_{threading.current_thread().ident}" #python 3
    print(f"Début tâche longue exécutée par {curr_thread} pendant {duree} secondes ")
    time.sleep(duree)
    return f"Fin tâche longue exécutée par {curr_thread}"

# Création d'un ThreadPoolExecutor
with ThreadPoolExecutor(max_workers=3) as executor:
    durees_args = [3, 3, 1, 5, 2]
    # Soumission des tâches au pool
    futures = [executor.submit(tache_longue, duree_arg) for duree_arg in durees_args]

    # Récupération des résultats :
    for future in futures:
        print(future.result())
```

⇒

Début tâche longue exécutée par thread_13916 pendant 3 secondes
 Début tâche longue exécutée par thread_14940 pendant 3 secondes
 Début tâche longue exécutée par thread_16272 pendant 1 secondes
 Début tâche longue exécutée par thread_16272 pendant 5 secondes
 Début tâche longue exécutée par thread_13916 pendant 2 secondes
 Fin tâche longue exécutée par thread_13916
 Fin tâche longue exécutée par thread_14940
 Fin tâche longue exécutée par thread_16272
 Fin tâche longue exécutée par thread_16272
 Fin tâche longue exécutée par thread_13916

La fonction **executor.submit()** renvoie un objet technique de type **Future** (comme en langage java et proche de Promise de javascript). Cet objet permettra d'attendre et de récupérer ultérieurement (dans le futur) le résultat du traitement long effectué en tâche de fond .

NB :

- L'utilisation de ThreadPoolExecutor est idéale pour des opérations I/O-intensives, comme les requêtes réseau, les lectures/écritures de fichiers, où la mise en attente est fréquente.
- Pour des tâches CPU-intensives, il peut être plus efficace d'utiliser ProcessPoolExecutor qui utilise plusieurs processus au lieu de threads, contournant ainsi le Global Interpreter Lock (GIL) de Python.

1.4. Threads producteurs/consommateurs : Queue

t4 queue.py

```
import threading
import queue
import time

# Création d'une file de données produites/consommées
# qui sera vue par les différents threads avec verrouillage/syncro automatique
myQueue = queue.Queue()

def producteur(iMinInc=0,iMaxExcl=5):
    for i in range(iMinInc,iMaxExcl):
        curr_thread = f'thread_{threading.current_thread().ident}'
        print(f'Sending/Producing {i} in myQueue by {curr_thread}')
        myQueue.put(i)
        time.sleep(1)

def consommateur():
    while True:
        item = myQueue.get()
        curr_thread = f'thread_{threading.current_thread().ident}'
        square = item * item
        print(f'Receiving/Consuming {item} in myQueue by {curr_thread} , square={square}')
        myQueue.task_done()

# Création des threads
thread_producteur1 = threading.Thread(target=producteur,args=[1,4])
thread_producteur2 = threading.Thread(target=producteur,args=[6,9])

thread_consommateur = threading.Thread(target=consommateur, daemon=True)
#daemon=True pour accepter fin du thread consommateur même le(s) dernier(s) élément(s)
# extrait(s) de la file n'a/ont pas été entièrement traité(s)

# Démarrage des threads
thread_producteur1.start(); thread_producteur2.start()
thread_consommateur.start()

# Attente de la fin des producteurs
thread_producteur1.join();thread_producteur2.join()

myQueue.join() # Attente jusqu'à ce que toutes les tâches soient consommées
```

```
⇒
Sending/Producing 1 in myQueue by thread_14916
Sending/Producing 6 in myQueue by thread_15596
Receiving/Consuming 1 in myQueue by thread_3112 , square=1
Receiving/Consuming 6 in myQueue by thread_3112 , square=36
...
Sending/Producing 3 in myQueue by thread_14916
Receiving/Consuming 3 in myQueue by thread_3112 , square=9
Sending/Producing 8 in myQueue by thread_15596
Receiving/Consuming 8 in myQueue by thread_3112 , square=64
```

1.5. Autres mécanismes de synchronisation de threads

Lock	Verrou ordinaire	
RLock	ReEntrant Lock , pour fonction récursive	rlock = threading.Rlock() et with rlock : fonction_recursive()
Event	Signalisation et attente d'un événement	evenement = threading.Event() et evenement.wait() et evenement.set()
Condition	Attente d'une condition vérifiée/remplie	condition = threading.Condition() et with condition: condition.wait() et ...

1.6. coroutines asynchrones (async/await) , python ≥ 3.5

```
async def simple_print(msg):
    print(msg)
```

Via le mot clef `async`, la fonction `simple_print` renvoie une nouvelle coroutine (ici du code qui affiche un message en paramètre) à chaque futur appel .

Le code d'une coroutine ne s'exécute pas directement , il faut la lancer dans un environnement asynchrone (géré par le module `asyncio`) :

- soit à l'intérieur d'une autre co-routine via le mot clef **await**
- soit via **asyncio.run()** ou autre équivalent

Voici un équivalent simplifié de **asyncio.run()** :

```
def my_async_run(coroutine):
    #sort of new REPL (Read-Eval-Print-Loop)
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(coroutine)
    #... plus ... finalisation...
```

Techniquement parlant, une coroutine est un enrobage orienté objet autour d'un itérateur/générateur renvoyant une certaine valeur après interprétation . Un objet coroutine comporte une méthode interne `__await__` permettant d'accéder au code du générateur interne .

Et finalement , lancer via **asyncio.run()** une nouvelle **co-routine** consiste à démarrer une nouvelle petite boucle d'interprétation . Cette boucle événementielle est propre au moteur asynchrone utilisé, et permet une utilisation concurrente des tâches (quelquefois sans utiliser de nouveaux threads).

NB : Comme en javascript, le **mot clef await** (de python) **ne peut être utilisé qu'à l'intérieur d'une fonction préfixée par async**

Le mot clef **await** permet d'attendre une chose attendable (**awaitable** in english) :

- soit une **autre (sous-) coroutine**
- soit un objet **Future** de bas niveau (ressemblant au résultat de `executor.submit()`)
- soit un objet **Task** (à créer par exemple via `task1 = asyncio.create_task(nested_coroutine1())`) et servant potentiellement à exécuter plusieurs traitements en parallèle (de manière concurrente, pas en mode séquentiel).

Attention : **`await asyncio.sleep(delay)`** plutôt que ~~`time.sleep(delay)`~~

Exemple simple coroutines.py:

```
import asyncio
import time
```

```
async def upper_after_delay(s, delay=1):
    await asyncio.sleep(delay)
    return s.upper()
```

```
async def in_specific_order():
    print(f'in_specific_order() started at {time.strftime('%X')}")
    s1 = await upper_after_delay("ile-de-", 2)
    print(f's1={s1}')
    s2 = await upper_after_delay("france", 2)
    s3 = s1+s2 ; print(f's2={s2} s3={s3}')
    print(f'in_specific_order() finished at {time.strftime('%X')}")
```

```
async def in_same_time():
    print(f'in_same_time() started at {time.strftime('%X')}")
    task1 = asyncio.create_task(upper_after_delay("ile-de-", 2))
    task2 = asyncio.create_task(upper_after_delay("france", 2))
    s1 = await task1
    print(f's1={s1}')
    s2 = await task2
    #NB: TaskGroup pour attendre en meme temps fin de task1 et task2 : à partir seulement de python 3.11
    s3 = s1+s2 ; print(f's2={s2} s3={s3}')
    print(f'in_same_time() finished at {time.strftime('%X')}")
```

```
asyncio.run(in_specific_order())
asyncio.run(in_same_time())
```

==>

```
in_specific_order() started at 11:41:20
s1=ILE-DE-
s2=FRANCE s3=ILE-DE-FRANCE
in_specific_order() finished at 11:41:24
in_same_time() started at 11:41:24
s1=ILE-DE-
s2=FRANCE s3=ILE-DE-FRANCE
in_same_time() finished at 11:41:26
```

Pour approfondir asyncio:

<https://docs.python.org/fr/3.9/library/asyncio-task.html>

XI - Affichages graphiques en python

1. Affichage graphiques en python

1.1. Présentation de tKinter

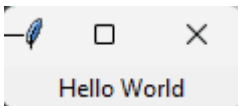
Tkinter (de l'anglais **Tool Kit interface**) est la **bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques**. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl (ancien langage de script "tool command langage").

NB: La librairie tKinter est normalement déjà installée dans le coeur de python (sous linux ou bien sous windows)
 ---> pas besoin de pip ni pipenv , sauf si besoin d'ajouter des extensions (ex: matplotlib)

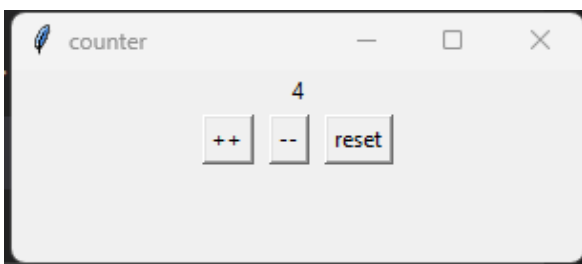
1.2. Hello world avec tkinter

```
from tkinter import *

#Hello world:
fenetre = Tk()
label = Label(fenetre, text="Hello World")
label.pack()
fenetre.mainloop()
```



1.3. exemple basic (counter) :



```
counter.py
import tkinter as tk

#create (default) top level window first:
mainWindow = tk.Tk()
mainWindow.title("counter")
mainWindow.geometry("300x100")# width x height + x + y
```

```

counterTkStringVar = tk.StringVar() #for binding content of labelCounter
counterTkStringVar.set("0") #default value

def onOpButtonPress(event):
    button = event.widget
    buttonText= button['text']
    v=int(counterTkStringVar.get())
    if buttonText=="++" :
        v+=1
    else:
        v-=1
    counterTkStringVar.set(str(v))

def onResetCommand():
    counterTkStringVar.set("0")

labelCounter = tk.Label(mainWindow, textvariable=counterTkStringVar)
labelCounter.pack()

btnFrame = tk.Frame(mainWindow)

buttonIncrement = tk.Button(btnFrame,text="++")
#buttonIncrement.pack() #without grid
buttonIncrement.grid(row=1,column=1,padx=4,pady=2)
buttonIncrement.bind("<ButtonPress>",onOpButtonPress)

buttonDecrement = tk.Button(btnFrame,text="--")
buttonDecrement.bind("<ButtonPress>",onOpButtonPress)
#buttonDecrement.pack() #without grid
buttonDecrement.grid(row=1,column=2,padx=4,pady=2)

buttonReset = tk.Button(btnFrame,text="reset",command=onResetCommand)
#buttonReset.pack() #without grid
buttonReset.grid(row=1,column=3,padx=4,pady=2)

btnFrame.pack()

mainWindow.mainloop()

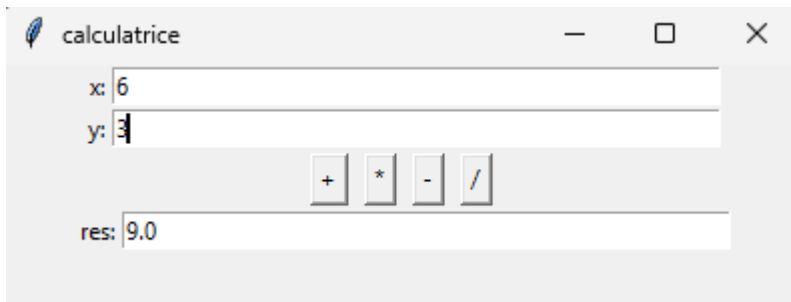
```

NB:

- en ajustant la valeur de `counterTkStringVar` via `.set()` on déclenche l'actualisation de la zone Label `labelCounter` associée via le paramétrage `textvariable="counterTkStringVar"`
- Une fonction événementielle telle que `onOpButtonPress(...)` et enregistrée via `.bind("<eventType>",...)` doit avoir un paramètre `event` .
- Une callback de type "`command`" telle que `onResetCommand()` et enregistrée via `command=onResetCommand` n'a normalement **aucun paramètre** .
- Au sein de la sous fenêtre `btnFrame` de type `Frame` les boutons sont disposés selon une grille (ici à une ligne et 3 colonnes) et les paramètres `padx` et `pady` servent à obtenir un certain

espacement entre les boutons .

1.4. Exemple simple (calculatrice)



calculatrice.py

```
import tkinter as tk

#create (default) top level window first:
mainWindow = tk.Tk()
mainWindow.title("calculatrice")
mainWindow.geometry("400x200")

resTkStringVar = tk.StringVar() #for binding content of labelResor entryRes
resTkStringVar.set("0") #default value

def computeRes(op):
    x=entryX.get()
    y=entryY.get()
    try:
        match op:
            case "+":
                z=float(x)+float(y)
            case "-":
                z=float(x)-float(y)
            case "*":
                z=float(x)*float(y)
            case "/":
                z=float(x)/float(y)
            case _:
                z=0
    except Exception:
        z=0
    resTkStringVar.set(str(z))

def onAdd():
    computeRes("+")

def onMult():
    computeRes("*")

def onMinus():
    computeRes("-")
```

```
def onDiv():
    computeRes("/")

panelX = tk.Frame(mainWindow)
labelX = tk.Label(panelX, text="x:");    labelX.pack(side="left")
entryX = tk.Entry(panelX,width=50); entryX.pack(side="right")
panelX.pack()
# idem pour panelY , labelY , entryY

panelBtn = tk.Frame(mainWindow)
buttonAdd = tk.Button(panelBtn,text="+", command=onAdd)
buttonAdd.grid(row=1,column=1,padx=4,pady=2)
# idem pour buttonMult , buttonMinus , buttonDiv
panelBtn.pack()

panelRes = tk.Frame(mainWindow)
labelOfLabelRes = tk.Label(panelRes, text="res:")
labelOfLabelRes.pack(side="left")
entryRes = tk.Entry(panelRes, textvariable=resTkStringVar,width=50)
entryRes.pack(side="right")
panelRes.pack()

mainWindow.mainloop()
```

NB :

- la méthode `.pack()` admet un paramètre facultatif `side=` dont la valeur peut être `"left"` , `"right"` , `"top"` ou `"bottom"` .
- Ordre d'affichage = ordre des appels à `subcomponentxy.pack()`

1.5. Vue d'ensemble sur positionnement des widgets

Placement absolu	<code>.pack()</code> mais <code>.place(x=... , y=...)</code>
Placement sur un des bords	<code>.pack()</code> avec <code>side= "left" , "right" , "top" ou "bottom"</code> et espacement avec <code>padx=... et pady=...</code>
Placement par grille	<code>.pack()</code> mais <code>.grid(row=..., column=...)</code>

Fonctions utilitaires (pour tkinter)

myTkinterUtil.py

```
import tkinter as tk

def withinFrameWithLabel(frame,labelText,tkControl,*othersControls):
    labelTk = tk.Label(frame, text=labelText,width=12)
    labelTk.pack(side="left")
    tkControl.pack(side="right")
    for otherControl in othersControls:
```

```
otherControl.pack(side="right")
frame.pack()
```

```
def entryStringVarWithLabelInFrame(master,labelText,textvariable=None):
```

```
    panelRow = tk.Frame(master)
```

```
    if not textvariable :
```

```
        textvariable = tk.StringVar()
```

```
    entryTk = tk.Entry(panelRow,width=48,textvariable=textvariable)
```

```
    withinFrameWithLabel(panelRow,labelText,entryTk)
```

```
    return textvariable
```

```
def buttonsInRowFrame(master,buttonLabelCommandTupleList):
```

```
    panelRow = tk.Frame(master)
```

```
    c=1
```

```
    for (btnLabel,btnCommand) in buttonLabelCommandTupleList:
```

```
        btn = tk.Button(panelRow,text=btnLabel , command=btnCommand)
```

```
        btn.grid(row=1,column=c,padx=2,pady=2)
```

```
        c+=1
```

```
    panelRow.pack()
```

1.6. Boîtes de dialogues de tkinter

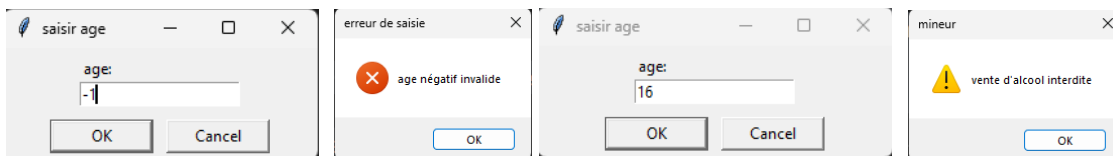
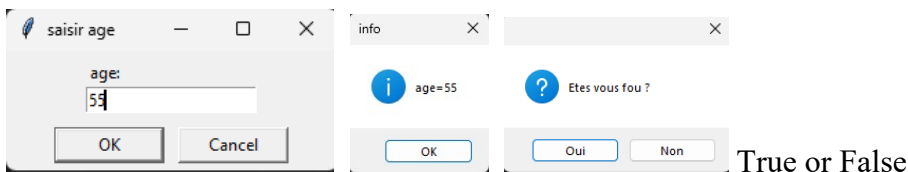
tk dialog.py

```
import tkinter.messagebox as tkmb;
import tkinter.simpledialog as tkstd;
import tkinter.filedialog as tkfd;

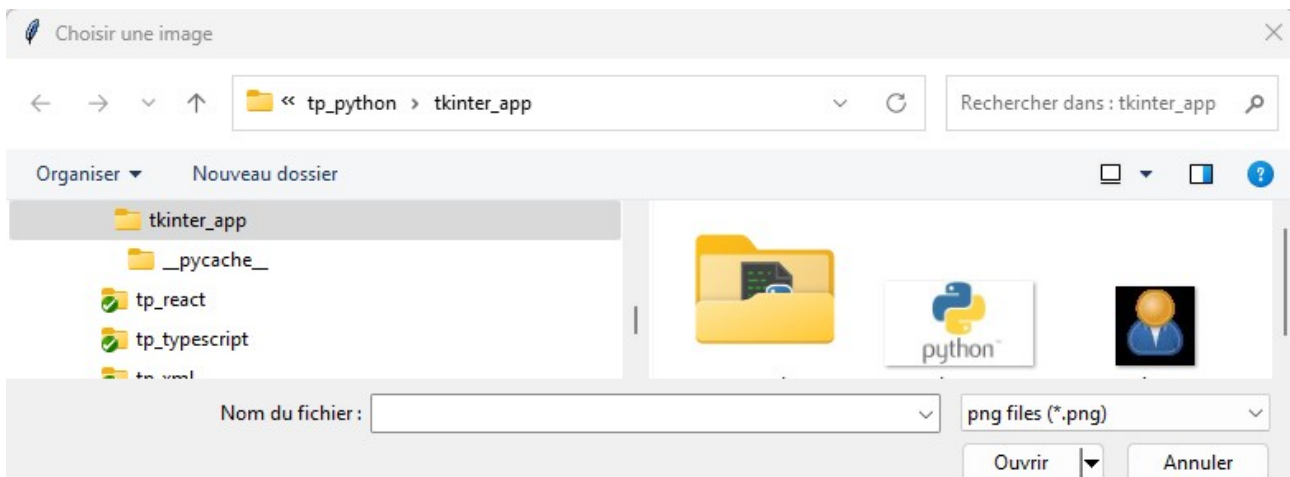
#username = tkstd.askstring("saisir username", "username:")
age = tkstd.askinteger("saisir age", "age:") # (titre, message)
#taille = tkstd.askfloat("saisir taille (cm)", "taille:")

if age < 0 :
    tkmb.showerror("erreur de saisie", "age négatif invalide")
else:
    tkmb.showinfo("info", f"age={age}") # (titre, message)
    if age < 18 :
        tkmb.showwarning("mineur", "vente d'alcool interdite")

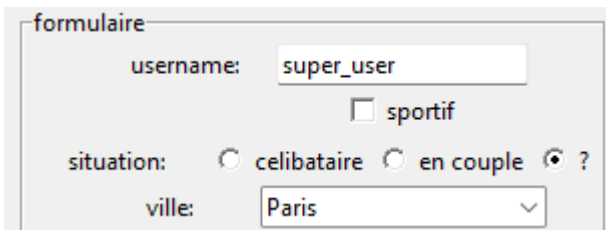
responseFouOuPas = tkmb.askyesno(message="Etes vous fou ?")
print(f"responseFouOuPas={responseFouOuPas}") # True or False
```



```
def cmde_choose_imageName():
    global fileImageName
    fileImageName=tkfd.askopenfilename(title="Choisir une image",
                                       filetypes=[('png files','*.png'),('all files','*')])
```



1.7. Widgets fondamentaux (Label, Entry , ...)



```
import tkinter as tk # partie principale/native de tkinter
from tkinter import ttk # extension classique de tkinter (Combobox, ...)
from my_tkinter_util import buttonsInRowFrame, withinFrameWithLabel
...
formLabelFrame = tk.LabelFrame(mainWindow, text="formulaire", padx=2, pady=2)
```

```
panelRowUsername = tk.Frame(formLabelFrame)
usernameStrVar = tk.StringVar(value="super_user")
usernameEntry = ttk.Entry(panelRowUsername, textvariable=usernameStrVar)
withinFrameWithLabel(panelRowUsername, "username:", usernameEntry)
```

```
panelRowSportif = tk.Frame(formLabelFrame)
sportifStrVar = tk.BooleanVar()
sportifCheckbox = tk.Checkbutton(panelRowSportif, text="sportif", variable=sportifStrVar)
withinFrameWithLabel(panelRowSportif, " ", sportifCheckbox)
```

```
panelRowEnCouple = tk.Frame(formLabelFrame)
situationStrVar = tk.StringVar(value="?")
inconnuRadioButton = tk.Radiobutton(panelRowEnCouple, text="?", value="?", variable=situationStrVar)
enCoupleRadioButton = tk.Radiobutton(panelRowEnCouple,
                                     text="en couple", value="en_couple", variable=situationStrVar)
celibataireRadioButton = tk.Radiobutton(panelRowEnCouple, text="celibataire",
                                         value="celibataire", variable=situationStrVar)
withinFrameWithLabel(panelRowEnCouple, "situation:", inconnuRadioButton,
                     enCoupleRadioButton, celibataireRadioButton)
```

```
panelRowVilles = tk.Frame(formLabelFrame)
villeStrVar = tk.StringVar(value="Paris")
villeComboBox = ttk.Combobox(panelRowVilles, textvariable=villeStrVar)
villeComboBox["values"] = ("Lyon", "Paris", "Toulouse")
withinFrameWithLabel(panelRowVilles, "ville:", villeComboBox)
```

```
formLabelFrame.pack( padx=10, pady=10)
#formLabelFrame.pack(fill="x", expand="yes", padx=10, pady=10)
#formLabelFrame.pack(fill="both", expand="yes", padx=10, pady=10)
```

1.8. Widgets secondaires de tKinter (Spinbox, Scale, ...)

```
panelRowLevel = tk.Frame(formLabelFrame)
levelStrVar = tk.StringVar(value="1")
levelSpinbox= tk.Spinbox(panelRowLevel,from_="1",to="5",textvariable=levelStrVar)
withinFrameWithLabel(panelRowLevel,"level:",levelSpinbox)
```



```
panelRowTemperature = tk.Frame(formLabelFrame)
temperatureVar = tk.DoubleVar(value=22.5)
temperatureScale= tk.Scale(panelRowTemperature,from_=-30,to=50,
                           variable=temperatureVar,orient=tk.HORIZONTAL,resolution=0.5)
withinFrameWithLabel(panelRowTemperature,"temperature:",temperatureScale)
```



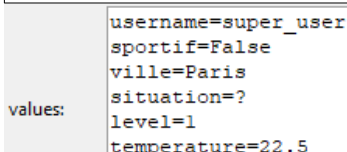
```
panedWindow = tk.PanedWindow(mainWindow, orient=tk.HORIZONTAL)
#panedWindow = Frame with resizable subcomponents (via slider)

panedWindow.add(tk.Label(panedWindow, text='Resizable part 1', background='lightgreen',
anchor=tk.CENTER , height=5))
panedWindow.add(tk.Label(panedWindow, text='Resizable part 2', background='lightblue',
anchor=tk.CENTER , height=5) )
panedWindow.pack(side=tk.BOTTOM, expand="yes", fill=tk.X, pady=2, padx=2)
```



Text (Entry multi-lignes) :

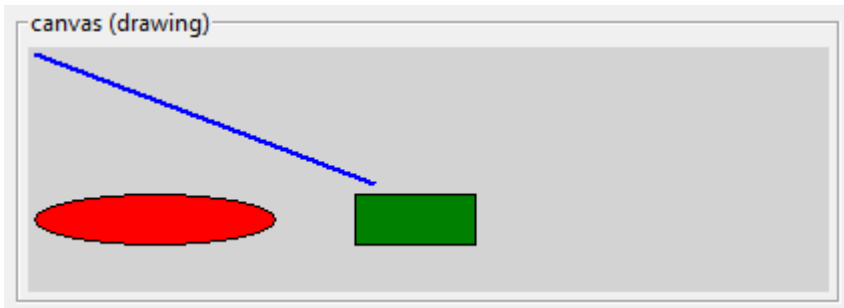
```
panelValues = tk.Frame(mainWindow)
valuesText= tk.Text(panelValues,width=50,height=8)#width,height in number of lines/columns
withinFrameWithLabel(panelValues,"values:",valuesText)
```



```
def dualDisplay(label,val):
    print(label,val)
    valuesText.insert(tk.END, f'{label}={val}\n')
```

1.9. Canvas tKinter (drawing area)

```
canvasLabelFrame = tk.LabelFrame(mainWindow,text="canvas (drawing)",padx=2,pady=2)
canvas = tk.Canvas(canvasLabelFrame, width=400, height=150, background='lightgrey')
canvas.pack()
canvasLabelFrame.pack(padx=2, pady=2)
```



```
def drawInCanvas():
    ligneIndex = canvas.create_line(5, 5, 175, 70 , width=2 ,fill="blue")
    cercleIndex = canvas.create_oval(5,75,125,100,fill="red")
    rectangleIndex = canvas.create_rectangle(165,75,225,100,fill="green")
    print(f'ligneIndex={ligneIndex} cercleIndex={cercleIndex} rectangleIndex={rectangleIndex}')
```

```
def clearCanvas():
    #canvas.delete(1,3) # delete elements of index 1 and 3 s'ils existent
    canvas.delete(tk.ALL)
```

1.10. Affichage d'image tKinter



```
#photo1 = tk.PhotoImage(file="python.png")
photo1 = tk.PhotoImage(file=fileName)

#photo1=photo1.zoom(2,2) #coefX, coefY as int (larger image)
#photo1=photo1.subsample(2,2) # smaller image
```

```
image_label = tk.Label(mainWindow, image=photo1,width=100,height=100)
image_label.image=photo1
image_label.pack()
```

ou bien

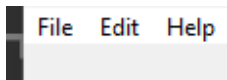
```
image1 = canvas.create_image(0, 0, anchor=tk.NW, image=photo1)
canvas.image=photo1 # ??? but necessary !!!
```

Si besoin d'afficher une image ".jpeg" (et pas ".png") :

#NB: PIL is for Python Image Library , cette extension necessite de lancer pip install pillow
#avec python3 , PIL/pillow sert à gérer plus de formats d'images (ex: .jpeg et autres en plus de .png)

```
from PIL import Image as PilImage, ImageTk as PilImageTk #necessite pip install pillow
pilImage = PilImage.open('fleur.jpeg')
tkImage = PilImageTk.PhotoImage(pilImage)
...
```

1.11. Menu tKinter



#build and add menubar:

```
def showMessageDialog(message):
    tkmb.showinfo("info",message)

def cmde_choose_imageName():
    showMessageDialog("choose_imageName")

def cmde_about():
    showMessageDialog("about / ex_widgets / python")

menubar = tk.Menu(mainWindow)

subMenuFile = tk.Menu(menubar, tearoff=0)
subMenuFile.add_command(label="Choose image", command=cmde_choose_imageName)
subMenuFile.add_separator()
subMenuFile.add_command(label="Quit", command=mainWindow.quit)
menubar.add_cascade(label="File", menu=subMenuFile)

subMenuEdit = tk.Menu(menubar, tearoff=0)
subMenuEdit.add_command(label="displayValues", command=displayValues)
menubar.add_cascade(label="Edit", menu=subMenuEdit)

subMenuHelp = tk.Menu(menubar, tearoff=0)
subMenuHelp.add_command(label="about", command=cmde_about)
menubar.add_cascade(label="Help", menu=subMenuHelp)

mainWindow.config(menu=menubar)
```

XII - 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'éléments 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'éléments 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 conjugué (meme partie réelle , 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]]
```


XIII - Accès aux bases de données

1. Accès aux bases de données en python

1.1. Principaux modules d'accès aux bases de données

mysql-connector-python <i>ou bien mysqlclient</i>	Pour se connecter à MySQL (ou MariaDB compatible)
sqlite3	Pour se connecter à une base embarquée SQLite
psycopg2	Pour se connecter à une base Postgres
pymongo	Pour se connecter à une base MongoDB
SQLAlchemy	Framework "Object-Relational-Mapping" (comme JPA/Hibernate en java) qui peut également déclencher des requêtes SQL en direct.
panda + SQLAlchemy ou autre	Combinaison possible avec la librairie panda (pour l'analyse de données) .

1.2. 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 données 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 données:
    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 modifiées :", 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) # departement 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 modifiées : 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 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;

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

ALTER TABLE departement
ADD CONSTRAINT FK_RegionValidePourDepartement
FOREIGN KEY (ref_region) REFERENCES region(num) ;

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, ref_region )
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");
```

1.3. accès SQL via SqlAlchemy

SqlAlchemy est une API assez généraliste qui permet soit de déclencher des requêtes simples en pur SQL, soit de manipuler la base de données en mode **ORM** "Object Relational Mapping".

Voici quelques premiers exemples (sans ORM) où SqlAlchemy est manipulé en mode "SQL" :

```
pip install sqlalchemy
```

```
pip install mysqlclient
```

init_geo_db_mysql.py (petite appli python qui lance un script sql pour initialiser une base mysql)

```
from sqlalchemy import create_engine, text

# db_url="database+dialect://username:password@hostname:port/databasename"
# for mysql+mysqldb this is required : pip install mysqlclient

username="root"
pwd="root"
db_hostname="localhost"
db_name="geoDB"
db_url=f"mysql+mysqldb://{username}:{pwd}@{db_hostname}:3306/{db_name}"
engine = create_engine(db_url)

with engine.connect() as cnx:
    with open("init_geoDB_mysql.sql") as file:
        query = text(file.read())
        cnx.execute(query)
    cnx.commit()
```

Variante pour sqlite (base de données embarquées sans serveur) :

init_geo_db_sqlite.py (petite appli python qui lance un script sql d'initialisation)

```
from sqlalchemy import create_engine, text

db_name="geoDB"
db_url=f"sqlite:///.{db_name}"
engine = create_engine(db_url)

with engine.connect() as cnx:
    with open("init_geoDB_sqlite.sql") as file:
        for ligne in file:
            query = text(ligne)
            cnx.execute(query)
    cnx.commit()

'''
Valid SQLite URL forms are:
sqlite:///memory: (or, sqlite://)
```

```
sqlite:///relative/path/to/file.db
sqlite:///absolute/path/to/file.db
'''
```

Eventuelle utilisation des "méta-data" de SqlAlchemy pour créer les structures de (nouvelles) tables (déclencher indirectement des ordres "CREATE TABLE" :

```
from sqlalchemy import create_engine, text
from sqlalchemy import MetaData
from sqlalchemy import Integer, String, Column, Table

metadata_object=MetaData()

region_table = Table(
    "region",
    metadata_object,
    Column('num', String(8), primary_key=True),
    Column('nom', String(64)),
    Column('chef_lieu', String(32))
)

departement_table = Table(
    "departement",
    metadata_object,
    Column('numero', String(6), primary_key=True),
    Column('nom', String(48)),
    Column('population', Integer),
    Column('superficie', Integer),
    Column('prefecture', String(48)),
    Column('ref_region', String(6))
)

db_name="geoDB"
db_url=f"sqlite:///./{db_name}" # ou bien url mysql ou autre (ex : postgres)
engine = create_engine(db_url)

# emitting DDL :
metadata_object.create_all(engine)
```

→ ce script python déclenche des ordres "CREATE TABLE ...".

Utilisation essentielle de SqlAlchemy (ici en mode SQL) en mode CRUD :

crud_geo.py

```
#pip install sqlalchemy, mysql+mysqldb required pip install mysqlclient
from sqlalchemy import create_engine, text
import json

db_name="geoDB"

#db_url=f"mysql+mysqldb://root:root@localhost:3306/{db_name}"
#engine = create_engine(db_url)
```

```

engine = db_url=f'sqlite:///./{db_name}'
engine = create_engine(db_url)

with engine.begin() as cnx:
    select_all_regions_sql="SELECT * FROM region"
    results=cnx.execute(text(select_all_regions_sql)).fetchall()
    for record in results:
        print(f'region={record}')

    insert_request = """insert into departement
        (numero, nom, population, superficie, prefecture)
        values (:numero, :nom, :population, :superficie, :prefecture)"""
    st_params = { "numero" :57, "nom" : "Moselle", "population" : 0, "superficie" : 0, "prefecture" : "Metz"}
    insert_statement = text(insert_request)
    res=cnx.execute(insert_statement,st_params)
    print("nombre de lignes affectées par insertion:",res.rowcount)

    update_request = """update departement
        set population=:population, superficie=:superficie
        where numero=:numero"""
    st_params = { "numero" :57, "population" : 1049942, "superficie" : 6216}
    update_statement = text(update_request)
    res=cnx.execute(update_statement,st_params)
    print("nombre de lignes affectées par update:",res.rowcount)

    #select_all_departements_sql="SELECT * FROM Departement"
    select_all_departements_sql="SELECT numero, nom, population, superficie, prefecture FROM departement"
    results=cnx.execute(text(select_all_departements_sql)).fetchall()
    for record in results:
        #print(f'departement={record}') #departement=('60', 'Oise', 818380, 5860, 'Beauvais', 'FR-HDF')
        column_names = ("numero", "nom", "population", "superficie", "prefecture")
        dep_as_dict = dict(zip(column_names,record))
        #print(dep_as_dict) # {'numero': '60', 'nom': 'Oise', 'population': 818380, 'superficie': 5860, 'prefecture': 'Beauvais'}
        dep_as_json_string = json.dumps(dep_as_dict)
        print (dep_as_json_string) # {"numero": "60", "nom": "Oise", "population": 818380, "superficie": 5860, "prefecture": "Beauvais"}

    delete_request = """delete from departement where numero=57"""
    delete_statement = text(delete_request)
    res=cnx.execute(delete_statement)
    print("nombre de lignes affectées par suppression:",res.rowcount)

    #commit automatique si pas d'exception en fin de bloc with ...as cnx :
    #fermeture automatique de la connexion en fin de bloc with ...as cnx :

```

Requêtes SQLAlchemy avec table meta-data :

```

from sqlalchemy import text , insert , update , delete
xyz_table = Table("xyz", metadata_object, ...) # exemple au sein des pages précédentes

```

```

insert_stmt = insert(xyz_table).values(colonne1=valeur_c1, colonne2=valeur_c2)
res=cnx.execute(insert_stmt)
auto_incr_pk=res.inserted_primary_key[0] # ici pk avec une seule colonne

```

```

update_stmt = update(xyz_table).values(colonne1=valeur_c1, ...)
        .where(xyz_table.c.id==val_id_to_update)

```

```
cnx.execute(update_stmt)
```

```
delete_stmt = delete(xyz_table).where(xyz_table.c.id == val_id_to_delete)  
cnx.execute(delete_stmt)
```

1.4. accès "ORM" via SQLAlchemy

geo_db_orm.py

```
from sqlalchemy import String, create_engine, select
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy.orm import Session

##### Definition d'une classe d'objet persistant (Modèle structurel / partie de l'ORM)

class Base(DeclarativeBase):
    pass

class Departement(Base):
    __tablename__ = "departement"

    numero: Mapped[str] = mapped_column(primary_key=True)
    nom: Mapped[str] = mapped_column(String(48))
    population: Mapped[int]
    superficie: Mapped[int]
    prefecture: Mapped[str]

    def __str__(self):
        return f'Departement(numero={self.numero}, nom={self.nom},
population={self.population}, superficie={self.superficie}, prefecture={self.prefecture})"
```

```
##### engine, connexion, ...
...
db_name="geoDB"
#db_url=f'mysql+mysqldb://{username}:{pwd}@{db_hostname}:3306/{db_name}'
#engine = create_engine(db_url)

engine = db_url=f'sqlite:///./{db_name}'
#engine = create_engine(db_url, echo=True) # avec affichage des requêtes déclenchées
engine = create_engine(db_url)
```

AJOUT / INSERT:

```
with Session(engine) as session:
    dep57=Departement(numero=57,nom="Moselle",prefecture="Metz")
    session.add(dep57)
    session.commit() # with a subcall to session.flush()
```

→

after add, before update, dep57_relu= Departement(numero=57, nom=Moselle, population=None, superficie=None, prefecture=Metz)

UPDATE:

```
with Session(engine) as session:

    select_dep57_st=select(Departement).where(Departement.numero == "57")
    dep57_relu=session.scalars(select_dep57_st).one() #return one object or an exception
    print("after add, before update, dep57_relu=",dep57_relu)
```



```
dep57_relu.population=1049942
dep57_relu.superficie=6216
session.commit() #for update # with a subcall to session.flush()
```

→

after update,before delete, dep57_relu= Departement(numero=57, nom=Moselle, population=1049942, superficie=6216 ,prefecture=Metz)

```
### delete:
```

```
with Session(engine) as session:
```

```
    select_dep57_st=select(Departement).where(Departement.numero == "57")
    dep57_relu=session.scalars(select_dep57_st).one() #return one object or an exception
    print("after update,before delete, dep57_relu=",dep57_relu)
```

```
    session.delete(dep57)
    session.commit() # with a subcall to session.flush()
```

```
    nb_dep57_after_delete=len(session.scalars(select_dep57_st).all())
    print("nb_dep57_after_delete=",nb_dep57_after_delete) # nb_dep57_after_delete= 0
```

Quelques liens/urls pour approfondir :

<https://www.datacamp.com/tutorial/sqlalchemy-tutorial-examples>

1.5. accès à MongoDB en python

pip install pymongo

local-mongo.py (ici en mode bloquant/synchrone) :

```
from pymongo import MongoClient

uri = "mongodb://localhost:27017"
client = MongoClient(uri)

try:
    database = client.get_database("news") # database name
    news_col = database.get_collection("news") # collection name (with s)
    query = { "title": "myNews" }
    my_news = news_col.find_one(query)
    print(my_news)
    client.close()
except Exception as e:
    raise Exception("Unable to find the document due to the following error: ", e)
```

Variante en mode asynchrone (avec async/await nécessitant python ≥ 3.7):

local-mongo-async.py

```
import asyncio
from pymongo import AsyncMongoClient

async def fetch_new_in_mongo():
    uri = "mongodb://localhost:27017"
    client = AsyncMongoClient(uri)
    try:
        database = client.get_database("news") # database name
        news_col = database.get_collection("news") # collection name (with s)
        query = { "title": "myNews" }
        my_news = await news_col.find_one(query)
        print(my_news)
        await client.close()
    except Exception as e:
        raise Exception("Unable to find the document due to the following error: ", e)

# Run the async function :
asyncio.run(fetch_new_in_mongo())
```

XIV - 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) Existe depuis 2004
flask	pour api REST ou bien appli web simple	Bon compromis fonctionnalités/complexités. Existe depuis 2010, beaucoup utilisé
<i>falcon</i>	<i>pour api REST</i>	<i>assez peu utilisé et maintenant concurrencé par fastapi</i>
fastapi	Pour api REST , très performant	Existe depuis 2018 , utilisation en progression
... autres

1.2. Django (bases élémentaires)

pip install Django

éventuellement/idéalement au sein d'un environnement virtuel .

Création d'un projet django (groupe d'applications gérées par un serveur django) :

django-admin startproject my_django_project

```
---> generate --->
my_django_project/
  manage.py
  my_django_project/
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
```

```
-----
cd my_django_project
py manage.py migrate
py manage.py runserver
```

url en phase de développement: <http://127.0.0.1:8000/>



The install worked successfully! Congratulations!

View [release notes](#) for Django 5.2

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

django

Création d'une application django (gérée par le serveur my_django_project) :

```
python manage.py startapp my_app
```

⇒ ce génère la sous arborescence suivante :

```
├── db.sqlite3
├── my_django_project
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py
└── my_app
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

my_django_project/my-app/views.py

```
from django.http import HttpResponse
def index(request):
    return HttpResponse('Hello, World!')
```

my_django_project/my_django_project/urls.py

```
from django.contrib import admin
from django.urls import path

from my_app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name="homepage")
]
```

relancer (si besoin) : `py manage.py runserver`

```
← → ↺ 🔍 http://127.0.0.1:8000
```

Hello, World!

1.3. Api rest avec flask (simple , classique)

pip install Flask

éventuellement/idéalement au sein d'un environnement virtuel

hello.py

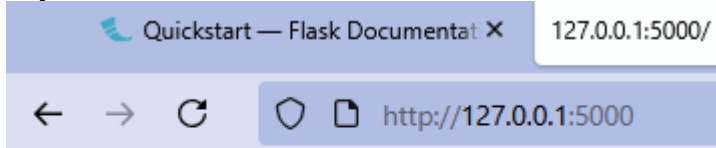
```
from flask import Flask

app = Flask(__name__)

@app.route("/hello")
def hello_world():
    return "<p>Hello, World!</p>"
```

flask --app hello run

http://127.0.0.1:5000/hello



Hello, World!

DeviseService (ici en mode Singleton avec dictionnaire en mémoire sans base de données) :

devise.py

```
class Devise() :
    #constructeur avec valeurs par défaut:
    def __init__(self,code=None,name=None,change=0) :
        self.code=code
        self.name=name
        self.change=change
```

singleton.py

```
class Singleton(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]

# class MyClass(BaseClass, metaclass=Singleton):
```

devise service.py

```

from devise import Devise

# pip install singleton
from singleton import Singleton

#version simulee sans base de données , simple map/dictionnary en memoire

class DeviseService(metaclass=Singleton):
    def __init__(self):
        self.devisesDict = {};
        self.devisesDict["EUR"] = Devise('EUR', 'Euro', 1)
        self.devisesDict["USD"] = Devise('USD', 'Dollar', 1.1)
        self.devisesDict["JPY"] = Devise('JPY', 'Yen', 130)
        self.devisesDict["GBP"] = Devise('GBP', 'Livre', 0.9)

    def getDevises(self):
        listeDevises = list(self.devisesDict.values());
        print(">>> listeDevises=", listeDevises);
        return listeDevises;

    def getDeviseById(self , id ):
        return self.devisesDict.get(id);

    def createDevise(self , dev: Devise):
        key = dev.code;
        if key in self.devisesDict:
            raise Exception("conflict : an existing Devise have same key/code:"+key );
        else:
            self.saveDevise(dev);

    def updateDevise(self, dev: Devise):
        key = dev.code;
        if key in self.devisesDict:
            self.saveDevise(dev);
        else:
            raise Exception("cannot update : no Devise found for key/code:"+ key);

    def saveDevise(self , dev : Devise ):
        self.devisesDict[dev.code]=dev;

    def deleteDeviseById(self, id):
        return self.devisesDict.pop(id,None);

```

devise_api.py

```

from flask import Flask , jsonify , abort , request
app = Flask(__name__)

from devise import Devise
from devise_service import DeviseService

api_prefix="/devise-api/v1/devises" #default sever url in dev mode: http://localhost:5000

```

*#NB: by default flask and fastapi produces JSONResponse so dictionary or list of dictionary are automatically
transform as json string , no need of json.dumps(..., ensure_ascii=False) or falsk jsonify()*

util function for convert data object to dictionary

```
def asJsonSerializableDict(objOrCol):
    #print("type=",type(objOrCol))
    if isinstance(objOrCol,(list,tuple)):
        return [ obj.__dict__ for obj in objOrCol ]
    else:
        return objOrCol.__dict__
```

```
def asDict(objOrCol): # shorter function name for frequent calls
    return asJsonSerializableDict(objOrCol)
```

@app.get(f"{api_prefix}/<id>") *#NB method param id should have same name that PathVariable <id>*

```
def get_devise_by_id(id):
    try:
        print("(get_devise_by_id() call with id=" , id)
        return asDict(DeviseService().getDeviseById(id))
    except Exception as e:
        abort(404,f"Devise not found for id={id}")
```

@app.post(f"{api_prefix}")

```
def create_devise():
    try :
        data = request.get_json() # (as dict if possible) from POST request body
        #print("jsonData received(as dict) in post mode",data)
        dev=Devise(data['code'], data['name'], data['change'])
        #print("devise received in post mode",dev)
        DeviseService().createDevise(dev)
        #return asDict(dev) # with default status code = 200/OK
        return asDict(dev) , 201 # with specific status code = 201/CREATED
    except Exception as e:
        abort(409,str(e))
```

@app.put(f"{api_prefix}/<id>")

```
def update_devise(id):
    try:
        data = request.get_json() # (as dict if possible) from PUT request body
        #print("jsonData received(as dict) in put mode",data)
        dev=Devise(data['code'],data['name'],data['change'])
        #print("devise received in put mode",dev)
        dev.code =id
        DeviseService().updateDevise(dev)
        return asDict(dev) # with default status code = 200/OK
    except Exception as e:
        abort(404,str(e))
```

@app.delete(f"{api_prefix}/<id>")

```
def delete_devise(id):
```



```

deletedDevise = DeviseService().deleteDeviseById(id)
if deletedDevise==None :
    abort(404,f'no Devise (to Delete) found for key/code=={id}')
else:
    #return "devise was sucessfully deleted" # with default status code = 200/OK
    return '', 204 # with specific status code = 204/NO_CONTENT

@app.get(f'{api_prefix}/' )
def get_devises():
    return asDict(DeviseService().getDevises()) #will be automatically JSON serializd by jsonify() of flask

#default flask behavior :
# if return resAsDict or resAsCollectionOfDict
# automatic return jsonify(resAsDict)

'''
#V1:
return [{"code": "USD", "name": "Dollar", "change":1},
        {"code": "EUR", "name": "Euro", "change": 1.1} ]
'''

```

flask --app devise_api run

⇒

Urls en phase de développement :

<http://127.0.0.1:5000/devise-api/v1/devises>

<http://127.0.0.1:5000/devise-api/v1/devises/USD>

⇒ {"change":1.1,"code":"USD","name":"Dollar"}

<http://127.0.0.1:5000/static/index.html>

Par défaut le framework flask considère que le sous répertoire "**static**" est l'endroit où l'on dépose les fichiers web statiques (index.html , ...css , ...js).

Via une URL de type <http://127.0.0.1:5000/static/index.html> la page d'accueil est ainsi automatiquement téléchargée vers le navigateur (ainsi que les fichiers annexes .css, .js) .

1.4. Api rest avec fastapi (spécialisé api-rest, performant)

```
python -m pip install fastapi[standard] uvicorn
```

éventuellement/idéalement au sein d'un environnement virtuel

Lancement en phase de dev:

```
fastapi dev main.py
```

Lancement en phase de prod:

```
uvicorn main:app --reload
```

default url: <http://127.0.0.1:8000/>

singleton.py , devise_service.py → même code qu'en version flask

devise.py

```
class Devise(object) :
    #constructeur avec valeurs par défaut:
    def __init__(self,code=None,name=None,change=0) :
        self.code=code
        self.name=name
        self.change=change
```

```
from pydantic import BaseModel
```

```
class DeviseModel(BaseModel):
    code: str
    name: str
    change: float
```

devise_api.py

```
# HTTPException not from http.client but from fastapi
from fastapi import APIRouter , HTTPException

from devise import Devise, DeviseModel
from devise_service import DeviseService

router = APIRouter(prefix="/devise-api/v1/devises")

#NB: by default fastapi produces JSONResponse so dictionnary or list are automatically
# transform as json string , no need of json.dumps(..., ensure_ascii=False)

@router.get("/{id}") #NB method param id should have same name that PathVariable {id}
async def get_devise_by_id(id):
    return DeviseService().getDeviseById(id);
```

```

@router.post("/", status_code=201 )
async def create_devise(dev : DeviseModel):
    try :
        DeviseService().createDevise(dev);
        return dev;
    except Exception as e:
        raise HTTPException(status_code=409 , detail = str(e))

@router.put("/{id}" )
async def update_devise(id,dev : DeviseModel):
    try:
        dev.code =id;
        DeviseService().updateDevise(dev);
        return dev;
    except Exception as e:
        raise HTTPException(status_code=404 , detail = str(e))

@router.delete("/{id}", status_code=204 )
async def delete_devise(id):
    deletedDevise = DeviseService().deleteDeviseById(id);
    if deletedDevise==None :
        raise HTTPException(status_code=404,
                             detail="no Devise (to Delete) found for key/code="+id)

@router.get("/")
async def get_devises():
    return DeviseService().getDevises();

"""
#V1:
return [{"code": "USD", "name": "Dollar" , "change":1},
        {"code": "EUR", "name": "Euro", "change": 1.1} ];

#V2:
devisesObjects = [Devise('EUR', 'Euro', 1), Devise('USD', 'Dollar', 1.1)];
return devisesObjects;
"""

```

main.py

```

from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles

from devise_api import router as devise_router
from fastapi.responses import HTMLResponse

app = FastAPI()

```

```

app.mount("/static", StaticFiles(directory="static"), name="static")

app.include_router(devise_router);

@app.get("/", response_class=HTMLResponse)
async def root():
    return """
    <html>
      <head><title>main</title></head>
    <body>
      <h3>devise-api rest</h3>
      <a href="/devise-api/v1/devises/EUR"> devise EUR</a> <br/>
      <a href="/devise-api/v1/devises"> all devises</a> <br/>
      <hr/>
      <a href="/docs" target="_new" >devise-api docs</a><br/>
      <a href="/static/index.html" target="_new" >index of embedded mini front-end</a><br/>
    </body>
    </html>
    """

```

avec **static/index.html** et autres (.html , .css, .js)

fastapi dev main.py

<http://127.0.0.1:8000/>

<http://127.0.0.1:8000/devise-api/v1/devises/EUR>

⇒ {"code":"EUR","name":"Euro","change":1}

XV - sys , os , ...

1. programmation système en python

1.1. Arguments sur ligne de commande

app_args.py

```
import sys
print("sys.argv=",sys.argv) # ['app_args.py', '2', '3']
print("app name=",sys.argv[0]) # app_args.py
if len(sys.argv)>1 :
    print("premier arg=",sys.argv[1]) # 2
if len(sys.argv)>2 :
    print("second arg=",sys.argv[2]) # 3
    try :
        a = int(sys.argv[1])
        b = int(sys.argv[2])
        c=a+b
        resMessage=f"a={a} b={b} c=a+b={c}"
        print(resMessage) # a=2 b=3 c=a+b=5
    except:
        print("erreur: arguments numeriques attendus" , file = sys.stderr)
```

py app_args.py 2 3

1.2. Accès aux variables d'environnement

```
import os
CASE_ENV=os.environ.get("CASE","") // with default value " ?" if CASE was not set
print(f"CASE_ENV={CASE_ENV}")
#...
```

set CASE=min ou bien MAJ # sous windows

export CASE=min ou bien MAJ # sous linux

py app_args.py 2 3

==>

CASE_ENV=min ou bien MAJ

a=2 b=3 c=a+b=5 ou bien A=2 B=3 C=A+B=5

1.3. Ecriture d'erreurs sur stderr*

```
import sys

print("erreur: arguments numeriques attendus" , file = sys.stderr)

sys.stderr.write("direct write in stderr ...\n") # possible mais avec formatage plus basique
```

py app_args.py 2 3r 2> ficErr.txt

XVI - Annexe – Test

1. Test unitaire python

1.1. Installation du module pytest

```
pip install pytest
```

1.2. Exemple de test élémentaire

basic.py

```
def addition(a,b):
    return a+b
```

test_basic.py

```
from basic import addition
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger()

def test_addition():
    resAdd=addition(3,5)
    assert resAdd == 8
    logger.info("addition(3,5)=" + str(resAdd))
```

1.3. Lancement d'un test unitaire simple

```
python -m pytest test_basic.py
```

ou bien

```
pytest test_basic.py
```

⇒

```
===== test session starts =====
platform win32 -- Python 3.13.3, pytest-8.3.5, pluggy-1.6.0
rootdir: C:\tp\local-git-mycontrib-repositories\tp_python\bases
collected 1 item
```

```
test_basic.py . [100%]
```

```
===== 1 passed in 0.02s =====
```

Autres possibilités :

- option `--html` pour rapport de test au format html si extension `pytest-html` installée via `pip`
- ...

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

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

1.1. Vue d'ensemble sur les alternatives possibles

<i>alternatives</i>	<i>principes</i>	<i>caractéristiques (avantages, inconvénients)</i>
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 interfacier)
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 interfacier
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_mylib_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_mylib.bat

```

REM .\myclib
.\myclibbis
pause

```

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

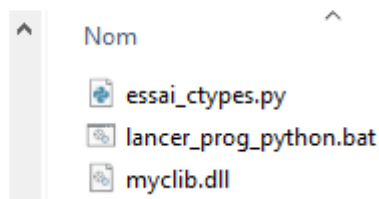
```

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
isEqual(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)

tp_python > with_ctypes



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 "..."
currentPlatform=platform.system()

#sharedLibraryExt ".dll" on windows or ".so" on linux
sharedLibraryExt=".dll" if currentPlatform=="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)
#display <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
 #(compatible 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 isEqual(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_ctype>python essai_ctype.py

D:\tp\tp_python\with_ctype/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

isEqual(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 "..."
currentPlatform=platform.system()

#sharedLibraryExt ".dll" on windows or ".so" on linux
sharedLibraryExt=".dll" if currentPlatform=="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)
#display <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/>

XVIII - 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 "aujourd'hui=5 avril 2025" - affiche ce message

Exo A3	<p>Ecrire un script a3.py qui :</p> <ul style="list-style-type: none"> - initialise une variable largeur à la valeur 5 et une variable longueur à 10 - qui calcul et affiche le périmètre et la surface d'un rectangle
Exo A4	<p>Ecrire un script a4.py qui :</p> <ul style="list-style-type: none"> - 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	<p>Ecrire un script a4.py qui :</p> <ul style="list-style-type: none"> - 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	<p>Ecrire un script a5.py qui :</p> <ul style="list-style-type: none"> - 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 valeur impaire
Exo A6	<p>Ecrire un script a6.py qui :</p> <p>boucle (via while nombre!= 33) sur :</p> <ul style="list-style-type: none"> - 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"
	<p>Ecrire un script a7.py qui :</p> <p>génère par concaténation du caractère '*' et affiche en boucle 8 lignes d'étoiles formant globalement un triangle :</p> <pre> * ** *** **** ***** ***** ***** ***** </pre>

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

Exo B1 (list)	<p>Ecrire un script b1.py qui :</p> <ul style="list-style-type: none"> - 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)	<p>Ecrire un script b2.py qui :</p> <ul style="list-style-type: none"> - initialise une chaîne de caractères s_suite = "2;7;8;9;26;5" - transforme s_suite 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)	<p>Ecrire un script b3.py qui va:</p> <ul style="list-style-type: none"> - partir de liste = ["rouge" , "vert" , "bleu" , "rouge" , "vert"] - générer l'ensemble color_set des valeurs sans doublon - afficher les valeurs et le type de color_set - initialise color_set2 avec color_set2 = { "jaune" , "vert" , "orange"} - afficher l'union des ensemble color_set et color_set2 - initialiser jour_set avec { "erreur" , "lundi" , "mardi" , "mercredi" , "jeudi"} - supprimer l'élément "erreur" de jour_set - ajouter l'élément "vendredi" dans jour_set - ré-afficher jour_set - ajouter d'un coup les l'éléments { "samedi" , "dimanche" } dans jour_set - ré-afficher jour_set - vider tous les éléments de jour_set - ré-afficher jour_set
ExoB4 (dict)	<p>Ecrire un script b4.py qui va :</p> <ul style="list-style-type: none"> - partir de <pre>values = [-2 , -7 , 2 , 8 , -9 , 10 , 1 , 3 , -3 , 5]</pre>

	<pre>dico_stats = { '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 .</pre>
ExoB5 (dict)	<p>Ecrire un script b5.py qui va :</p> <ul style="list-style-type: none"> - partir de <pre>dico_produits={ 'fruit' : ["banane" , "orange" , "pomme"] , 'legume' : ["carotte" , "choux" , "tomate"], 'divers' : ['stylo' , 'cahier'] }</pre> <ul style="list-style-type: none"> - 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)	<p>Ecrire un script b6.py qui va :</p> <ul style="list-style-type: none"> - partie de <pre>liste_pays=[{ '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 },]</pre> <ul style="list-style-type: none"> - calculer et afficher la population totale (319529060) .
ExoB7 (str , palindrome)	<p>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) .</p> <p>pour tester des phrases entières , on supprimera d'abord tous les espaces exemple : esope reste ici et se repose</p>
ExoB8 (str)	<pre>nom_fichiers=["fichierA.txt" , "configB.json" , "b8.py"] # boucler sur chaque nom complet de fichier # et afficher séparément nom et extension</pre>
ExoB9 (.format)	<pre>liste_dico_dates =[{ 'jour' : 4 , 'mois' : 'avril' , 'annee' : 2024} ,</pre>

```
{ '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	<p>Ecrire un script qui :</p> <ul style="list-style-type: none"> - comporte une fonction <code>volume_sphere()</code> permettant de calculer le volume d'une sphere selon son rayon et la formule $\frac{4}{3} * \text{PI} * \text{rayon} * \text{rayon} * \text{rayon}$ - appelle en boucle cette fonction a partir de rayons= [5.0 , 10, 100] - affiche les rayons et les volumes calculés
Exo C2 (recursive)	<p>Ecrire dans le script <code>c2.py</code> la fonction <code>ma_factorielle(n)</code> qui calcule la factorielle de <code>n</code> : 1 si <code>n</code> vaut 0 et <code>1 * 2 * 3 * n-1 * n</code> sinon</p> <p>Effectuer en boucle des appels de <code>ma_factorielle</code> avec <code>n=0,1, 2,3, 4,5 ,6, 7</code></p> <p>Coder une version récursive <code>ma_factorielle_recursive(n)</code> qui s'appelle elle même avec <code>n-1</code></p> <p>Effectuer des appels de <code>ma_factorielle_recursive</code> avec <code>n=0,1, 2,3, 4,5 ,6, 7</code> pour obtenir normalement les mêmes résultats</p>
Exo C3 (lambda)	<p>Ecrire dans un script <code>c3.py</code> une fonction <code>enchainer_transformation_et_affichage_formaté(s,f_trans ,f_format)</code> qui va enchaîner une transformation et un affichage formaté d'une chaîne de caractère <code>s</code> .</p> <p>Le paramètre <code>f_trans</code> sera une référence sur une fonction de transformation de chaîne de caractères et le paramètre <code>f_format</code> sera une référence sur une fonction effectuant un formatage (ex : ajout de préfixe , d'encadrement, ...)</p> <p>Appeler ensuite plusieurs fois cette fonction en passant des lambdas expressions en paramètres qui vont par exemple appeler <code>s.upper()</code> ou <code>s.lower()</code> et <code>***" + s+ "***</code></p>
Exo C4 (params nommés)	<p>Au sein de <code>c4.py</code> coder une fonction <code>create_cercle_dict(rayon,xc,yc,couleur)</code></p> <p>qui construira et retournera un dictionnaire de de type <code>{'rayon': 50, 'xc': 0, 'yc': 0, 'couleur': 'blue'}</code></p> <p>On donnera des valeurs par défaut aux 3 derniers paramètres de la fonction.</p> <p>On appellera ensuite plein de fois cette fonction avec plein de variantes sur les paramètres d'entrée (précisés ou pas , nommés ou pas, ...).</p>

Exo C5 (ng args variable)	<p>Au sein de c5.py coder la fonction <code>mon_maxi(*args)</code> et effectuer les appels suivants</p> <pre>print(mon_maxi(12,6,89,3)) #89 print(mon_maxi(8,3,9,4,23,7,12)) #23</pre> <p>Coder également <code>display_carre(**kwargs)</code> devant avoir à peu près ce comportement :</p> <pre>display_carre(x=6,y=9) # x=6 , carre(x=36) y=9 , carre(y=81) display_carre(x=2,y=4,z=6) # x=2 , carre(x=4) y=4 , carre(y=16) z=6 , carre(z=36)</pre>
Exo C6 (import)	<p>Au sein du module <code>c6_util.py</code> coder les fonctions <code>traduire_rgb_fr_en(s)</code> et <code>traduire_rgb_en_fr(s)</code> permettant de traduire du français à l'anglais (et vice versa) des noms de couleurs élémentaires telles que "rouge" , "vert" , "bleu"</p> <p>Au sein de <code>c6_app.py</code> ajouter l'instruction <code>import</code> nécessaire à ce genre d'appels :</p> <pre>print("vert en anglais=",color_util.traduire_rgb_fr_en("vert")) # vert en anglais= green</pre> <p>On pourra facultativement tester plusieurs variantes de l'instruction <code>import</code></p>
Exo C7 (math)	<p>En se basant sur la formule suivante</p> $Mensualité = \frac{\text{Capital Emprunté} \times \frac{\text{Taux Annuel}}{12}}{1 - \left(1 + \frac{\text{Taux Annuel}}{12}\right)^{-Durée \text{ en mois}}}$ <p>coder dans <code>c7.py</code> la fonction <code>calcul_mensualite_emprunt(...)</code> permettant de calculer les mensualités constantes à rembourser chaque mois pour un emprunt .</p> <p>Comportement attendu avec <code>taux_annuel_pct</code> exprimé en % :</p> <pre>mensualite = calcul_mensualite_emprunt(montant=100000 , nb_mois=120 , taux_annuel_pct=2.5) # 942.69</pre>
Exo C8 (dates)	<p>Au sein de <code>c8.py</code> coder un affichage de votre date d'anniversaire ainsi que le jour de la semaine pour votre naissance (ex : "Monday" ou "Friday" ou ...)</p>
Exo C9 (tris)	<p>Ecrire un script <code>c9.py</code> qui va trier plusieurs fois</p> <pre>liste_produits = [{ 'id' : 1 , 'label' : 'cahier' , 'price' : 2.4 }, { 'id' : 11 , 'label' : 'classeur' , 'price' : 4.4 }, { 'id' : 2 , 'label' : 'stylo' , 'price' : 1.4 }, { 'id' : 4 , 'label' : 'gomme' , 'price' : 2.41 },</pre>

	<pre>{ 'id' : 3 , 'label' : 'trousse' , 'price' : 5.4 },]</pre> <p>selon différents critères (par id, par label , par price)</p>
Exo C10 (filter,map)	<p>Ecrire un script c10.py qui a partir de la même liste_products que celui du tp c9 va</p> <ul style="list-style-type: none"> - calculer et afficher la liste_filtree des produits dont le prix est ≤ 4 - calculer et afficher la liste des prix de liste_filtree - calculer et afficher le prix total <p>On utilisera filter() , map() , reduce() et des lambdas expressions</p>

2.6. Exceptions , Fichiers et with et bases de prog système

Exo D1 exceptions prédéfinies	<p>Au sein de d1.py , à partir de values= ["81" , "-12" ,"36" , "abc" , "64"] effectuer une boucle for qui pour chaque valeur va :</p> <ul style="list-style-type: none"> - convertir la chaîne en valeur numérique - calculer et afficher la racine carrée <p>NB : on intégrera au sein de cette boucle un traitement d'exception qui affichera l'exception et son type sans plantage du programme</p>
D2 Exceptions soulevées et rattrapées	<p>Au sein de d2.py , écrire une fonction my_str_len(s) qui va retourner la longueur de s si la condition isinstance(s,str) est vérifiée et qui va soulever une exception de TypeError avec un message d'erreur détaillé tel que "s is not as string but <class 'int'>"</p> <p>A partir de values = [5 , "abc" , "defgh" , 6.6] effectuer une boucle for (avec traitement d'exception) qui pour chaque valeur va : appeler my_str_len() et affichera soit le résultat soit l'exception détaillée</p>
D3 finally et re-propager exception	<p>Au sein de d3.py , à partir de liste_taches=[<pre>{ 'x': 81 , 'op': 'racine' , 'res' : '?' , 'status' : "todo" }, { 'x': -81 , 'op': 'racine' , 'res' : '?' , 'status' : "todo"}, { 'x': 6 , 'op': 'carre' , 'res' : '?' , 'status' : "todo"}]</pre> <p>écrire une fonction do_task(task) (avec try : except : et finally :) qui va :</p> </p>

- analyser task['op'] et déclencher le calcul adéquat
- placer dans task['res'] le résultat du calcul si tout va bien
- placer dans task['res'] l'exception en cas de problème
- placer dans task['status'] la valeur 'done' dans tous les cas.

Appeler ensuite en boucle cette fonction sur liste_taches

Afficher ensuite la nouvelle valeur de liste_taches

ex :

```
task= {'x': 81, 'op': 'racine', 'res': 9.0, 'status': 'done'}
```

```
task= {'x': -81, 'op': 'racine', 'res': ValueError('math domain error'), 'status': 'done'}
```

```
task= {'x': 6, 'op': 'carre', 'res': 36.0, 'status': 'done'}
```

Soit la fonction élémentaire suivante (à recopier) :

```
def my_div(a,b):
```

```
    return a/b
```

Programmer une nouvelle fonction div_after_decrement()

qui appellera my_div(a-1,b-1) et qui en cas d'erreur va re-propager l'exception en y ajoutant une message via .add_note() avant d'invoquer raise.

Appeler div_after_decrement(9,5) et div_after_decrement(3,1) avec ou sans try/except

<p>D4 (gestion de fichier avec f.close())</p>	<p>Ecrire un script python d4_write.py qui va écrire dans un fichier my_fic.txt , la date , l'heure et un nombre entier aléatoire entre 1 et 100</p> <p><u>Exemple (my_fic.txt) :</u></p> <pre>date:23/06/2025 heure:10:36:10 aleatoire:3</pre> <p>Ecrire un script d4_read.py qui va lire ligne par ligne le fichier my_fic.txt et qui va afficher toutes les lignes (non vides) entre ** et ** (et sans sauts de lignes inutiles).</p> <p><u>Exemple :</u></p> <pre>**date:23/06/2025** **heure:10:36:10** **aleatoire:3**</pre>
<p>D5 (gestion de fichier</p>	<p>Sans f.close() et plutôt avec with ... as f , ecrire le script python d5.py qui va (sans try/except) au fil le l'eau</p>

avec with)	<p>lire (ligne par ligne) des valeurs numériques au sein de d5inputs.txt et générer des lignes de racines carrées calculées au sein de d5outputs.txt</p> <p>le fichier <i>d5inputs.txt</i> comportera les lignes suivantes :</p> <pre>81 64 9 -8 25</pre> <p>avec le -8 volontairement erroné en avant dernière ligne.</p> <p>On ignorera donc le message d'erreur "<i>racine = math.sqrt(val)</i> <i>ValueError: math domain error</i>" prévisible au lancement de python d5.py</p> <p>fichier <i>d5outputs.txt</i> normalement généré :</p> <pre>9.0 8.0 3.0</pre>
D6 prog système avec répertoire, fichiers, ...	<p>Ecrire un script d6.py qui va récupérer un nom de répertoire existant d'une manière ou d'une autre (exemple : par saisies ou bien par paramètres passés au programmes),</p> <p>Ce script va ensuite effectuer une boucle sur tous les fichiers de ce répertoire via <code>os.listdir(repertoire)</code> de manière à calculer et afficher les statistiques suivantes :</p> <ul style="list-style-type: none"> - nom et taille du fichier le plus petit - nom et taille du fichier le plus grand - nombre total de fichiers - taille totale (somme de la taille de tous les fichiers du répertoire) <p><u>NB</u>: on ne tiendra pas compte des éventuels sous-répertoires .</p> <p><i>Indications</i> : <code>os.path.getsize()</code> et <code>os.path.isfile()/isdir()</code> , <code>c=os.path.join(repertoire,c)</code></p>
	<p>.... NB : les exercices suivants (moins fondamentaux) pourront être considérés comme facultatifs et à effectuer en fin de formation selon le temps disponible et les centres d'intérêts</p>
D7	<p>Essais libres sur os, sys et shutil ou bien fichier binaire en accès direct</p>
	<p>Points communs des exercices D8,D9,D10 :</p> <p>ouvrir un fichier films.xyz (ou xyz est où est un format ".csv" ou ".json" ou ".xml") et générer stats_films.xyz dans le même format.</p> <p>Dans le point de départ on aura quelques films tels que <i>films.csv</i> <i>titre;date;realisateur;acteurs</i> <i>Titanic;1998-01-07;James Cameron;Leonardo DiCaprio, Kate Winslet, Billy Zane</i></p>

	<p>Arrete-moi si tu peux;2003-02-12;Steven Spielberg;Leonardo DiCaprio, Tom Hanks, Christopher Walken</p> <p>Raison et sentiments;1996-02-28;Ang Lee;Emma Thompson, Kate Winslet, Hugh Grant</p> <p>Seul au monde;2001-01-17;Robert Zemeckis;Tom Hanks, Helen Hunt, Nick Searcy</p> <p>La Ligne verte;2000-03-01;Frank Darabont;Tom Hanks, Michael Clarke Duncan, David Morse</p> <p>Forrest Gump;1994-10-05; Robert Zemeckis;Tom Hanks, Gary Sinise, Robin Wright</p> <p>Un Singe en hiver;1962-05-11;Henri Verneuil;Jean-Paul Belmondo, Jean Gabin, Noel Roquevert</p> <p>L'Homme de Rio;2013-05-14;Philippe de Broca;Jean-Paul Belmondo, Françoise Dorleac, Jean Servais</p> <p>The Artist;2011-10-12; Michel Hazanavicius;Jean Dujardin, Berenice Bejo, John Goodman</p> <p>Et dans les stats, des statistiques par acteurs telles que <i>stats-films.csv</i></p> <p><i>acteur;principaux_films</i></p> <p>Leonardo DiCaprio;Titanic, Arrete-moi si tu peux</p> <p>Kate Winslet;Titanic, Raison et sentiments</p> <p>Tom Hanks;Arrete-moi si tu peux, Seul au monde, La Ligne verte, Forrest Gump</p> <p>...</p>
D8 (csv)	Films et stats au format CSV
D9 (json)	Films et stats au format JSON
D10 (xml)	Films et stats au format XML
...	...

2.7. Programmation orientée objet

Exo E1	<p>Classe Personne avec (prenom,nom,age) et <i>incrémenter_age()</i></p> <p>code de la classe Personne dans e1_Personne.py</p> <p>et utilisation de la classe Personne dans e1.py avec</p> <p>from e1_Personne import Personne</p>
Exo E2 (facultatif)	<p>Classe Voiture avec (marque,modele,couleur, vitesse) et <i>accélérer()</i> , <i>ralentir()</i></p> <p>à coder dans e2_Voiture.py et à utiliser dans e2.py</p>
Exo E3	<p>Classe Compte avec (numero,label,solde) et <i>débiter()</i> , <i>créditer()</i></p> <p>à coder dans e3_Compte.py et à utiliser dans e3.py</p>
Exo E4 (héritage et polymorphisme)	<p>Sous classe Employe héritant de Personne et ajoutant (salaire , fonction</p> <p>à coder dans e4_Employe.py et à utiliser dans e4.py</p> <p>Ajouter dans e1_Personne.py une méthode <i>.decrire()</i> qui affichera les valeurs internes d'une personne à la console via <code>print(f"...")</code></p> <p>Dans e4_Employe.py coder la méthode <i>.decrire()</i> qui rappellera la version héritée et qui affichera en plus fonction et salaire.</p>

Exo_E5	<p>Au sein de e4.py effectuer un test de polymorphisme sur une liste de Personne (avec Employe et Personne) en appelant en boucle la méthode .decire()</p> <p>Peaufiner le fichier e3_Compte.py de manière à y ajouter un attribut Compte.last_max_num_compte partagé au niveau de l'ensemble de la classe Compte et une méthode de classe auto_incr_num() (préfixée par @classmethod) qui va auto incrémenter la valeur de Compte.last_max_num_compte.</p> <p>Retourner si besoin le constructeur de compte pour garantir une cohérence</p> <p>Tester ceci dans e5.py via du code de ce genre :</p> <pre>c12=Compte(12,"compte_b",50.0) print("c12=",c12) nouveau_compte_c = Compte(Compte.auto_incr_num() ,"compte_c",120.0) print("nouveau_compte_c=",nouveau_compte_c)</pre>
Exo_E6 (classe abstraite)	<p>Coder au sein de e6_Porte.py :</p> <ul style="list-style-type: none"> - une classe abstraite Porte avec self.nom et self.fermee=True or False et comportant : <pre>def etat(self): return "fermee" if self.fermee else "ouverte"</pre> - les méthodes <code>__str__</code> et <code>.decire()</code> appelant entre autres <code>{self.type_porte()}</code> et <code>{self.type_etat()}</code> - les méthodes abstraites <code>ouvrir()</code> , <code>fermer()</code> et <code>type_porte()</code> - une classe concrète PortePivotante héritant de porte et affichant au sein de <code>ouvrir()</code> et <code>fermer()</code> des messages de ce type : "angle ouverture=90 ° pour porte" - une classe concrète PorteCoulissante héritant de porte et affichant au sein de <code>ouvrir()</code> et <code>fermer()</code> des messages de ce type : "ouverture sur glissière=80 cm pour porte" <p>Coder ensuite au sein de e6.py une utilisation de ce genre :</p> <pre>listePortes=[] listePortes.append(PorteCoulissante("porte1",fermee=True)) listePortes.append(PortePivotante("porte2",fermee=True))</pre> <p>devant conduire à des messages qui ressemblent à ceux-ci :</p> <pre>ouverture sur glissière=80 cm pour porte porte1 angle ouverture=90 ° pour porte porte2 Porte de type=coulissante de nom=porte1 qui est ouverte Porte de type=pivotante de nom=porte2 qui est ouverte</pre>
Exo_E7	Retoucher/améliorer la classe Personne (dans e1_Personne.py) de manière à ce que

(@property)	l'age soit toujours positif. On utilisera self.__age , @property() et @age.setter() Effectuer ensuite un test dans e7.py
Exo E8 (essais libres)	Autres expérimentations libres sur la programmation orientée objet en python.

2.8. pip, venv, et aspects avancés (décorateurs, ...)

Exo F1 (venv)	<p>Utilisation de la librairie markdown dans venv (sans pipenv)</p> <p>-----</p> <p>soit fl.py comportant ce code :</p> <pre>import markdown my_html_text = markdown.markdown("***Bonjour**") print("my_html_text",my_html_text)</pre> <p>Exécuter python fl.py en direct génère normalement une erreur de ce type :</p> <pre>ModuleNotFoundError: No module named 'markdown'</pre> <p>Lancer la commande pip list pour visualiser que le module markdown n'est pas dans la liste des modules installés au niveau global</p> <p>Créer un env python virtuel :</p> <pre>python -m venv .venv</pre> <p>Activer ce .venv via la commande adéquate (de windows ou linux ou autre) et attendre le prompt (.venv)</p> <p>Après le prompt (.venv) lancer la commande suivante :</p> <pre>pip install markdown</pre> <p>puis l'application fl.py :</p> <pre>python fl.py</pre> <p>Après le prompt (.venv) lancer la commande deactivate pour sortir de l'environnement virtuel</p> <p>Vérifier via pip list que le module markdown n'est pas installé au niveau global</p> <p>Réactiver .venv (sous windows ou linux ou autre)</p> <p>Après le prompt (.venv) , relancer pip list</p> <p>Après le prompt (.venv) ; relancer python fl.py</p>
------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Exo F2 (pipenv)	<p>Après le prompt (.venv) ; relancer deactivate</p> <p>Utilisation de la librairie markdown avec pipenv</p> <p>-----</p> <p>soit f2.py comportant ce code :</p> <pre>import markdown my_html_text = markdown.markdown("*Hello*") print("my_html_text",my_html_text)</pre> <p>Exécuter python f2.py en direct génère normalement une erreur de ce type :</p> <pre>ModuleNotFoundError: No module named 'markdown'</pre> <p>-----</p> <p>Dans un terminal, au niveau global, lancer la commande suivante :</p> <p><i>python -m pip install pipenv</i></p> <p>Vérifier l'installation via la commande suivante :</p> <p><i>python -m pipenv --version</i></p> <p>Lancer ensuite cette commande :</p> <p><i>python -m pipenv install markdown</i></p> <p>puis enfin cette nouvelle commande :</p> <p><i>python -m pipenv run f2.py</i></p> <p>Vérifier via pip list que le module markdown n'est toujours pas installé au niveau global</p> <p>Vérifier via la commande <i>python -m pipenv run pip list</i> que markdown est installé au sein de l'environnement virtuel géré par pipenv .</p> <p>NB: * Le module markdown-it existe et est un peu mieux que le module markdown. * Il est éventuellement possible d'effectuer des réglages dans l'IDE (vscode ou autre pour que l'environnement virtuel soit bien pris en compte)</p>
Exo f3 (essais libres)	Quelques essais libres de syntaxes avancées au sein de f3.py
Exo f4 (comprehension)	<p>A partir du contenu initial suivante de f4.py</p> <pre>phrases=["Bonjour", "Python est un serpent efficace", "sans majuscule au debut", "Ok , tout va bien", "pasBien car commençant par une minuscule"]</pre>

	<pre>print("toutes_les_phrases",phrases)</pre> <p>Ajouter des listes en compréhension de manière à calculer et afficher :</p> <ul style="list-style-type: none"> - tailles_de_toutes_les_phrases - phrases_avec_majuscules (sur premier caractère) - tailles_phrases_majuscules (sur premier caractère)
Exo f5 (surcharge opérateur)	<p>Au sein de f5.py , créer deux instances c1 et c2 de la classe Compte (de e3_Compte.py) avec des soldes différents. Comparer ensuite ces deux comptes via l'opérateur ></p> <pre>if c1 > c2 : print(" c1 est plus grand que c2 d'un point de vue solde courant") else : print(" c1 n'est pas plus grand que c2 d'un point de vue solde courant")</pre> <p>NB : Lors d'un premier essai, sans amélioration de la classe Compte dans e3_Compte.py , on obtient le message d'erreur suivant :</p> <p><i>TypeError: '>' not supported between instances of 'Compte' and 'Compte'</i></p> <p>Ajouter le code de la surcharge de l'opérateur > au sein de la classe Compte (dans e3_Compte.py via la méthode spéciale __gt__(self, other)) Puis relancer la comparaison au sein de f5.py</p>
Exo f6 (décorateur)	<p>En partant de ce code initial dans f6.py</p> <pre>import re # expressions regulieres def camel_to_snake_case(s): return re.sub(r'([a-z])([A-Z])', r'\1_\2', s).lower() s1="unChameauAvecDesBossesPourChaqueMajuscule" s2 = camel_to_snake_case(s1) print("en snake_case:",s2) def encadrer1(chaine): return f">>>{chaine}<<<"</pre> <p>Ajouter un décorateur camel_to_snake_decorator(func) appliquant la conversion camel case vers snake case sur le résultat d'un fonction retournant une chaîne de caractères .</p> <p>Appliquer ensuite ce décorateur sur une copie/variante encadrer2() de la fonction</p>

	<p>encadrer1().</p> <p>Appeler en boucle encadrer1() et encadrer2() sur ceci</p> <pre>liste_phrases_avec_bosses = ["JeVeuxMaMaman", "PythonQueJaime", "PhraseQuiVaBien"]</pre> <p>pour normalement obtenir un résultat de ce genre :</p> <pre>[>>>JeVeuxMaMaman<<<, '>>>PythonQueJaime<<<', '>>>PhraseQuiVaBien<<<'] [>>>je_veux_ma_maman<<<, '>>>python_que_jaime<<<', '>>>phrase_qui_va_bien<<<']</pre>
Exo f7 (itérateur)	Programmer la classe MyCountDown sous forme d' itérateur effectuant un compte à rebours et utiliser celle ci pour effectuer un compte à rebours de 5 à 0.
Exo f8 (générateur)	Au sein de f8.py code un générateur count_down_generator basé sur le mot clef yield et offrant la même fonctionnalité de compte à rebours que la classe MyCountDown du tp précédent. Effectuer un compte à rebours de 5 à 0 en guise de test .
Exo f9 (expression régulière)	<p>Au sein de f9.py</p> <pre>tab_ref_prod = ["Aa123xy" , "bb7ttt" , "ab674Ua" , "45yy" , "au5" , "ry345yxt" , "zy145Yx"]</pre> <p>Effectuer en boucle un appel à re.match() pour vérifier si les éléments du tableau ci dessus sont bien (ou pas) des références correctes de produits.</p> <p>On considérera qu'une référence correcte est formulée par :</p> <ul style="list-style-type: none"> 2 caractères alphabétiques (minuscule ou majuscule) au début 3 chiffres (au milieu) 2 caractères alphabétiques (minuscule ou majuscule) à la fin
Exo f10 (expression régulière)	<p>Ecrire et appliquer une fonction basée sur une expression régulière qui remplace le caractère "," par le caractère "." ce qui peut être pratique pour manipuler des valeurs numériques saisies approximativement avec une virgule à la française.</p> <p>Appliquer cela sur un tableau de valeurs de ce type</p> <pre>tab_val = ["12.6" , "14,78" , "3.6" , "67,89" , "1.4" , "124,677" , "134,67"]</pre>
Exo f11 (essais libres)	Essais libres sur expressions régulières ou autres syntaxes avancées de python

2.9. Bibliothèques graphiques et scientifiques (tkinter, numpy , ...)

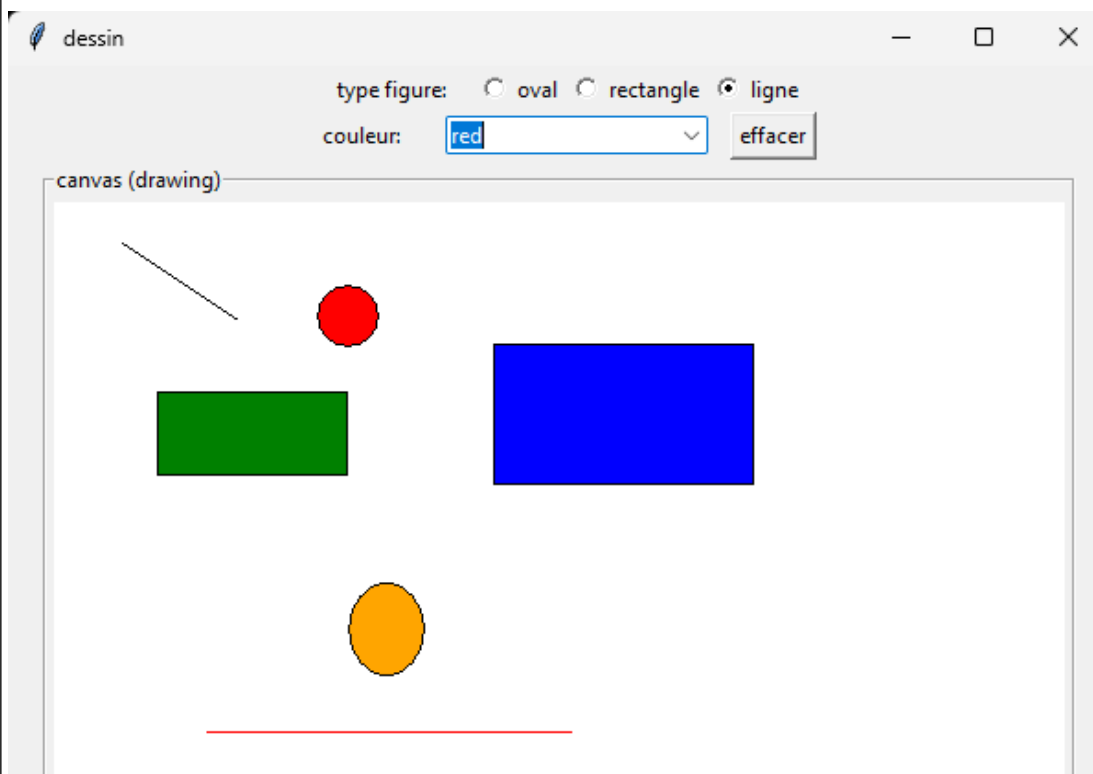
Exo G1

exo_tva_v1.py (en s'appuyant sur librairie **tkinter**) :

exo_tva_v2.py (avec Combobox pour choisir le taux de tva : 5,10, 20)

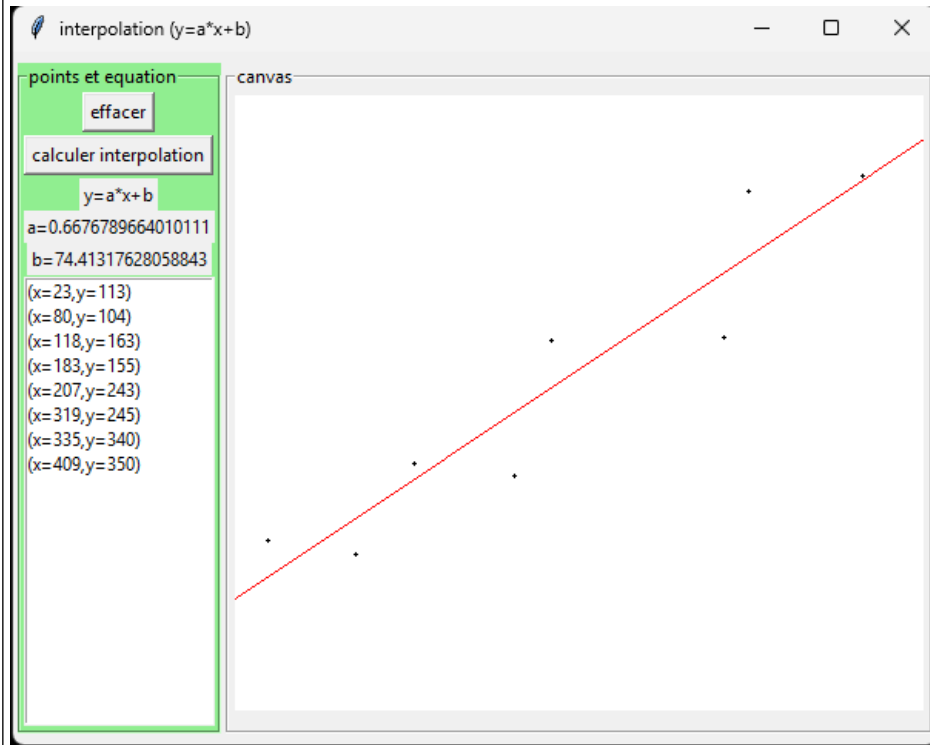
G2

Petite application de dessin avec tkinter.Canvas

exo_dessin.py

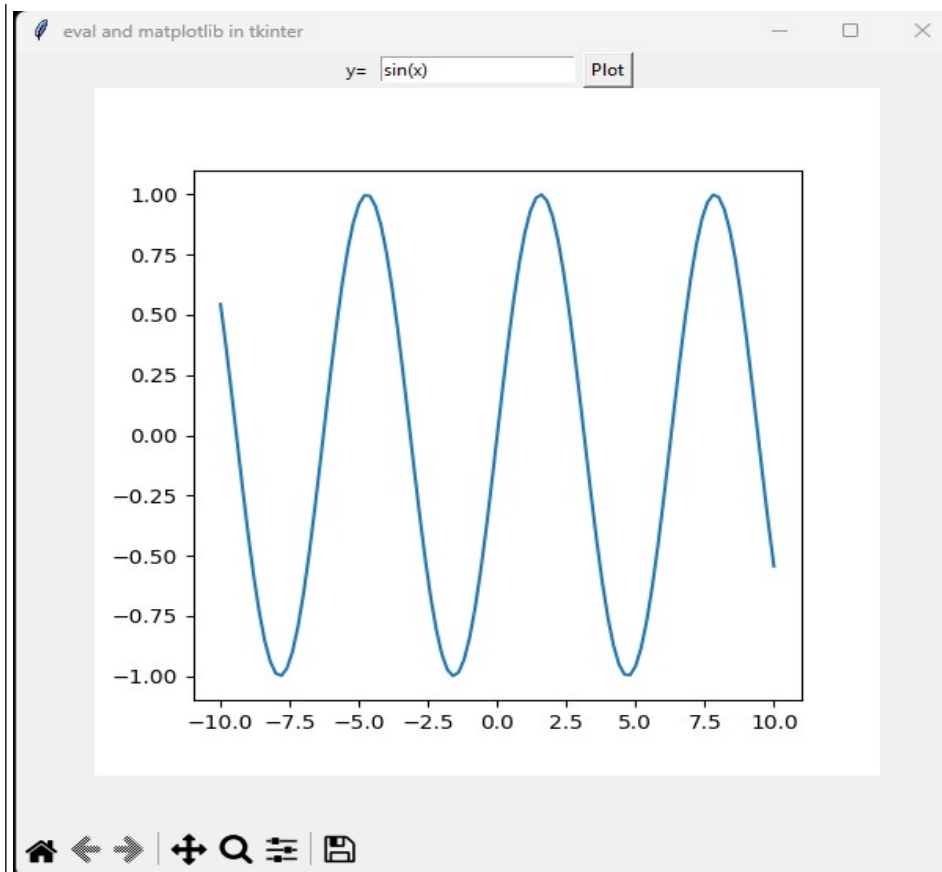
G3

Analyser l'exemple *interpolation.py* (calcul et affichage de la droite des moindres carrés)



G4

Analyser l'exemple *equation_graph.py* (avec matplotlib)



G5 Essais libres avec numpy

...

2.10. Accès aux bases de données depuis python

<p>Exo H1 (sqlite et sqlalchemy en mode sql)</p>	<p><u>NB</u> : au sein de l'exercice h1, on s'appuiera sur l'api SQLAlchemy.</p> <p>pip install sqlalchemy</p> <p>Ecrire un script python h1_init_db.py qui va initialiser une base de données au format sqlite et d'url <code>sqlite:///minibank_db</code> avec une table <code>compte(numero,label,solde)</code> compatible avec la structure de <code>e3_Compte.py</code></p> <p>Ecrire un script h1_crud.py qui va :</p> <ul style="list-style-type: none"> - créer une instance de Compte - stocker cela dans la base de données - effectuer une relecture pour vérifier - modifier la valeur du solde - mettre à jour les valeurs en base - effectuer une relecture pour vérifier
------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	NB: On pourra librement structurer le code en s'appuyant éventuellement sur des fichiers annexes (ex : h1_db_params.py, h1_metadata.py) et sur une approche éventuellement orientée objet (h1_CompteDao.py).
Exo h2	Essais libres sur l'accès aux base de données
...	

2.11. Http/web/restApi en python

Exo I1	<p>En s'appuyant sur le package Flask, programmer une api REST permettant d'effectuer des opérations CRUD (en mode POST;GET,PUT,DELETE) sur des éléments de type "Compte" .</p> <p>URLs proposées:</p> <p>http://127.0.0.1:5000/static/index.html et http://127.0.0.1:5000/minibank-api/v1/comptes</p> <p>NB : on pourra s'appuyer sur des constructions partielles des exercices e3 (classe Compte) et h1 (stockage en base de données)</p> <p>Pour effectuer des tests, on pourra éventuellement s'appuyer sur une application de type postman (ou un équivalent) .</p> <p>Démarrage proposé :</p> <p>flask --app i1_minibank_api run</p>
Exo I2	Autres essais libres sur web/http/api_rest
...	

2.12. Aspects avancés de python (thread, ...)

Exo J1 (démarrer un thread)	Énoncé à rédiger et solution à préparer
Exo J2 (Lock)	Énoncé à rédiger et solution à préparer
Exo J3	Énoncé à rédiger et solution à préparer

(ThreadPool
Executor et
Future)

Exo J4 (avec Queue)	Énoncé à rédiger et solution à préparer
Exo J5 (coroutine)	Énoncé à rédiger et solution à préparer
Exo J6 (coroutine)	Énoncé à rédiger et solution à préparer
Exo J7	autres essais libres sut Threads , coroutines et async/await