I - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TD/TP python

2.1. Installer si besoin python et VisualStudioCode ou pyCharm

https://www.python.org

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

Pour tester l'installation :

python --version

2.2. Utilisation directe de python en mode interactif

python

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

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

Exo A1	Ecrire un script a1.py qui affiche "Hello world"
Exo A2	Ecrire un script a2.py qui : - initialise 3 variables (ex : jour=5 , mois = "avril" , annee = 2025) - construit par concaténation un message de type "aujourdhui=5 avril 2025" - affiche ce message
Exo A3	Ecrire un script a3.py qui :

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

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

Exo B1 (list)	Ecrire un script b1.py qui : - initialise une liste l1 avec les valeurs [-1, 2 , -8 , 7 , 4 , 3 , -9 , 9 , 12 , -5 , 5] - construit et affiche la liste positifs des valeurs positives - construit et affiche la liste impairs des valeurs impaires - initialise une liste noms avec les valeurs ["jean" , "Luc" , "eric" , "Julie"] - retire toutes les valeurs ne commençant pas par une majuscule dans la liste des noms - affiche la liste des noms restants
ExoB2	Ecrire un script b2.py qui :

(list)	- 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)	Ecrire un script b3.py qui va: - 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)	Ecrire un script b4.py qui va : - partir de values = [-2 , -7 , 2 , 8 , -9 , 10 , 1 , 3 , -3 , 5] 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 .

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

ExoB6 (list/dict)	Ecrire un script b6.py qui va : - partie de 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 },

	] - calculer et afficher la population totale (319529060) .
ExoB7 (str , palindrome)	Ecrire un script b7.py qui va vérifier (et afficher) si oui ou non une chaine de caractères saisie est un palindrome (se lit dans les 2 sens à l'identique tel que kayak ou bob , radar, elle) . pour tester des phrases entières , on supprimera d'abord tous les espaces exemple : esope reste ici et se repose
ExoB8 (str)	nom_fichiers=["fichierA.txt" , "configB.json" , "b8.py"] # boucler sur chaque nom complet de fichier # et afficher séparément nom et extension
ExoB9 (.format)	liste_dico_dates =[{ 'jour' : 4 , 'mois' : 'avril' , 'annee' : 2024} , { 'jour' : 12 , 'mois' : 'juin' , 'annee' : 2023} , { 'jour' : 22 , 'mois' : 'octobre' , 'annee' : 2025} ,] # construire et afficher des dates sous forme de chaines (via .format()) # récupérer et afficher la date d'aujourd'hui (et l'heure aussi)

2.5. Fonction et appels , lambdas

Exo C1	Ecrire un script qui : - comporte une fonction volume_sphere() permettant de calculer le volume d'une sphere selon son rayon et la formule 4/3 * PI * rayon * rayon * rayon - appelle en boucle cette fonction a partir de rayons= [5.0 , 10, 100] - affiche les rayons et les volumes calculés
Exo C2 (recursive)	Ecrire dans le script c2.py la fonction ma_factorielle(n) qui calcule la factorielle de n : 1 si n vaut 0 et 1 * 2 * 3 * n-1 * n sinon Effectuer en boucle des appels de ma_factorielle avec n=0,1, 2,3, 4,5 ,6, 7 Coder une version récursive ma_factorielle_recursive(n) qui s'appelle elle même avec n-1 Effectuer des appels de ma_factorielle_recursive avec n=0,1, 2,3, 4,5 ,6, 7 pour obtenir normalement les mêmes résultats
Exo C3 (lambda)	Ecrire dans un script c3.py une fonction enchainer_transformation_et_affichage_formate(s,f_trans ,f_format) qui va enchainer une transformation et un affichage formaté d'une chaine de caractère s . Le paramètre f_trans sera une référence sur une fonction de transformation de chaine de caractères et le paramètre f_format sera une référence sur une fonction effectuant un formatage (ex : ajout de préfixe , d'encadrement, …) Appeler ensuite plusieurs fois cette fonction en passant des lambdas expressions en paramètres qui vont par exemple appeler s.upper() ou s.lower() et "**" + s+ "**"
Exo C4 (params	Au sein de c4.py coder une fonction create_cercle_dict(rayon,xc,yc,couleur)

nommés)	qui construira et retournera un dictionnaire de de type {'rayon': 50, 'xc': 0, 'yc': 0, 'couleur': 'blue'} On donnera des valeurs par défaut aux 3 derniers paramètres de la fonction. On appellera ensuite plein de fois cette fonction avec plein de variantes sur les paramètres d'entrée (précisés ou pas , nommés ou pas, …).

Exo C5
(ng args variable)
	Au sein de c5.py coder la fonction mon_maxi(*args)
	et effectuer les appels suivants
	print(mon_maxi(12,6,89,3)) #89
	print(mon_maxi(8,3,9,4,23,7,12)) #23
	
	Coder également display_carre(**kwargs)
	devant avoir à peu près ce comportement :
	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)

Exo C6 (import)	Au sein du module c6_util.py coder les fonctions traduire_rgb_fr_en(s) et traduire_rgb_en_fr(s) permettant de traduire du francais à l'anglais (et vice versa) des noms de couleurs élémentaires telles que "rouge" , "vert" , "bleu" Au sein de c6_app.py ajouter l'instruction import nécessaire à ce genre d'appels : print("vert en anglais=",color_util.traduire_rgb_fr_en("vert")) # vert en anglais= green On pourra facultativement tester plusieurs variantes de l'instruction import
Exo C7 (math)	En se basant sur la formule suivante $$Mensualité = \frac{\text{Capital Emprunté x } \frac{Taux\ Annuel}{12}}{1 - (1 + \frac{Taux\ Annuel}{12})^{-Durée\ en\ mois}}$$ coder dans c7.py la fonction calcul_mensualite_emprunt(...) permettant de calculer les mensualités constantes à rembourser chaque mois pour un emprunt . Comportement attendu avec taux_annuel_pct exprimé en % : mensualite = calcul_mensualite_emprunt(montant=100000 , nb_mois=120 , taux_annuel_pct=2.5) # 942.69
Exo C8 (dates)	Au sein de c8.py coder un affichage de votre date d'anniversaire ainsi que le jour de la semaine pour votre naissance (ex : "Monday" ou "Friday" ou ...)
Exo C9 (tris)	Ecrire un script c9.py qui va trier plusieurs fois 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}, { 'id' : 3 , 'label' : 'trousse' , 'price' : 5.4 },] selon différents critères (par id, par label , par price)
Exo C10 (filter,map)	Ecrire un script c10.py qui a partir de la même liste_produits que celui du tp c9 va

	- 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 On utilisera filter() , map() , reduce() et des lambdas expressions

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

Exo D1 exceptions prédéfinies	Au sein de d1.py , à partir de values= ["81" , "-12" ,"36" , "abc" , "64"] effectuer une boucle for qui pour chaque valeur va : - convertir la chaîne en valeur numérique - calculer et afficher la racine carrée NB : on intégrera au sein de cette boucle un traitement d'exception qui affichera l'exception et son type sans plantage du programme
D2 Exceptions soulevées et rattrapées	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'>" 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
D3 finally et re-propager exception	Au sein de d3.py , à partir de liste_taches=[{ 'x': 81 , 'op': 'racine' , 'res' : '?' , 'status' :"todo" }, { 'x': -81 , 'op': 'racine' , 'res' : '?' , 'status' :"todo"}, { 'x': 6 , 'op': 'carre' , 'res' : '?' , 'status' :"todo"}] écrire une fonction do_task(task) (avec try : except : et finally :) qui va : - 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'execption 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
D4 (gestion de fichier avec f.close())	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 *Exemple (my_fic.txt) :* date:23/06/2025 heure:10:36:10 aleatoire:3 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). *Exemple :* **date:23/06/2025** **heure:10:36:10** **aleatoire:3**
D5 (gestion de fichier avec with)	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 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 le fichier *d5inputs.txt* comportera les lignes suivantes : 81 64 9 -8 25 avec le -8 volontairement erroné en avant dernière ligne. On ignorera donc le message d'erreur *"racine = math.sqrt(val)* *ValueError: math domain error"* prévisible au lancement de python d5.py fichier *d5outputs.txt* normalement généré : 9.0 8.0 3.0
D6 prog système avec répertoire, fichiers, ...	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), Ce script va ensuite effectuer une boucle sur tous les fichiers de ce répertoire via os.listdir(repertoire) de manière à calculer et afficher les statistiques suivantes : - 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) NB: on ne tiendra pas compte des éventuels sous-répertoires . *Indications* : os.path.getsize() et os.path.isfile()/isdir() , c=os.path.join(repertoire,c)
	…. 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 …..
D7	Essais libres sur os, sys et shutil ou bien fichier binaire en accès direct
	Points communs des exercices D8,D9,D10 : 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. Dans le point de départ on aura quelques films tels que *films.csv* t*itre;date;realisateur;acteurs* *Titanic;1998-01-07;James Cameron;Leonardo DiCaprio, Kate Winslet, Billy Zane* *Arrete-moi si tu peux;2003-02-12;Steven Spielberg;Leonardo DiCaprio, Tom Hanks, Christopher Walken* *Raison et sentiments;1996-02-28;Ang Lee;Emma Thompson, Kate Winslet, Hugh Grant* *Seul au monde;2001-01-17;Robert Zemeckis;Tom Hanks, Helen Hunt, Nick Searcy* *La Ligne verte;2000-03-01;Frank Darabont;Tom Hanks, Michael Clarke Duncan, David Morse* *Forrest Gump;1994-10-05; Robert Zemeckis;Tom Hanks, Gary Sinise, Robin Wright* *Un Singe en hiver;1962-05-11;Henri Verneuil;Jean-Paul Belmondo, Jean Gabin, Noel Roquevert* *L'Homme de Rio;2013-05-14;Philippe de Broca;Jean-Paul Belmondo, Françoise Dorleac, Jean Servais* *The Artist;2011-10-12; Michel Hazanavicius;Jean Dujardin, Berenice Bejo, John Goodman* Et dans les stats, des statistiques par acteurs telles que *stats-films.csv* *acteur;principaux_films* *Leonardo DiCaprio;Titanic, Arrete-moi si tu peux* *Kate Winslet;Titanic, Raison et sentiments* *Tom Hanks;Arrete-moi si tu peux, Seul au monde, La Ligne verte, Forrest Gump* *...*
D8 (csv)	Films et stats au format CSV
D9 (json)	Films et stats auformat JSON
D10 (xml)	Films et stats au format XML
...	...

2.7. Programmation orientée objet

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

	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 .decrire()
Exo_E5	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. Retourcher si besoin le constructeur de compte pour garantir une cohérence Tester ceci dans e5.py via du code de ce genre : *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)*
Exo_E6 (classe abstraite)	Coder au sein de e6_Porte.py : - une classe abstraite Porte avec self.nom et self.fermee=True or False et comportant : def etat(self): return "fermee" if self.fermee else "ouverte" - les méthodes __str__ et .decrire() appelant entre autres {self.type_porte()} et {self.type_etat()} - les méthodes abstraites ouvrir() , fermer() et type_porte() - une classe concrète PortePivotante héritant de porte et affichant au sein de ouvrir() et fermer() des messages de ce type : "angle ouverture=90 ° pour porte …." - une classe concrète PorteCoulissante héritant de porte et affichant au sein de ouvrir() et fermer() des messages de ce type : "ouverture sur glissière=80 cm pour porte …." Coder ensuite au sein de e6.py une utilisation de ce genre : listePortes=[] listePortes.append(PorteCoulissante("porte1",fermee=True)) listePortes.append(PortePivotante("porte2",fermee=True)) devant conduire à des messages qui ressemblent à ceux-ci : *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*
Exo_E7 (@property)	Retoucher/améliorer la classe Personne (dans e1_Personne.py) de manière à ce que 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.

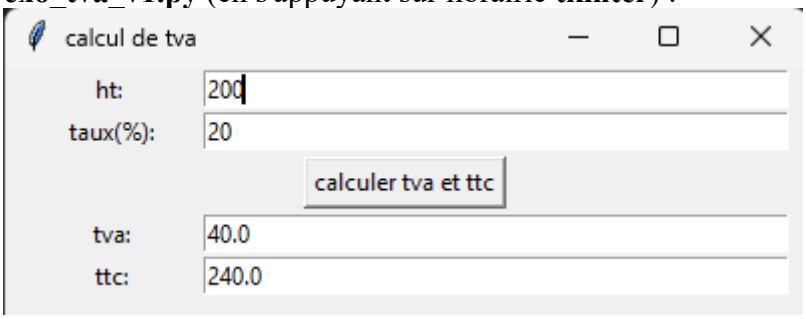
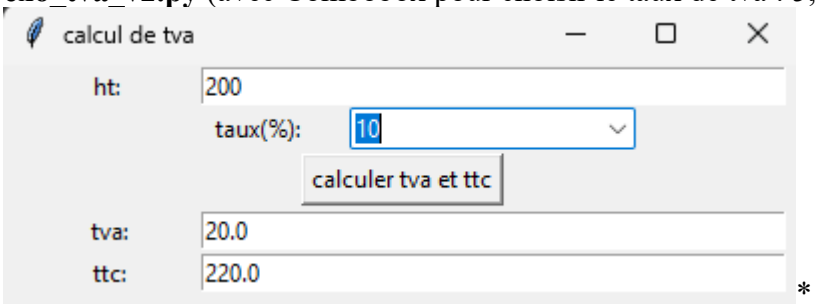
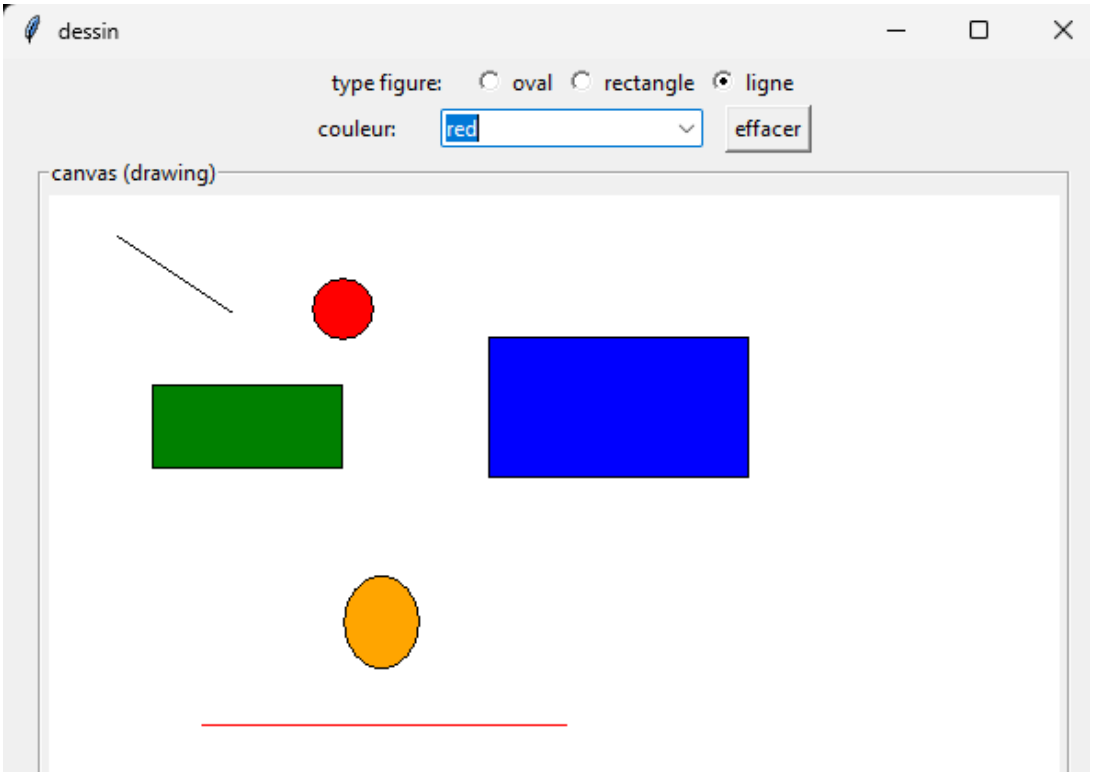
2.8. pip, venv, et aspects avancés (décorateurs, ...)

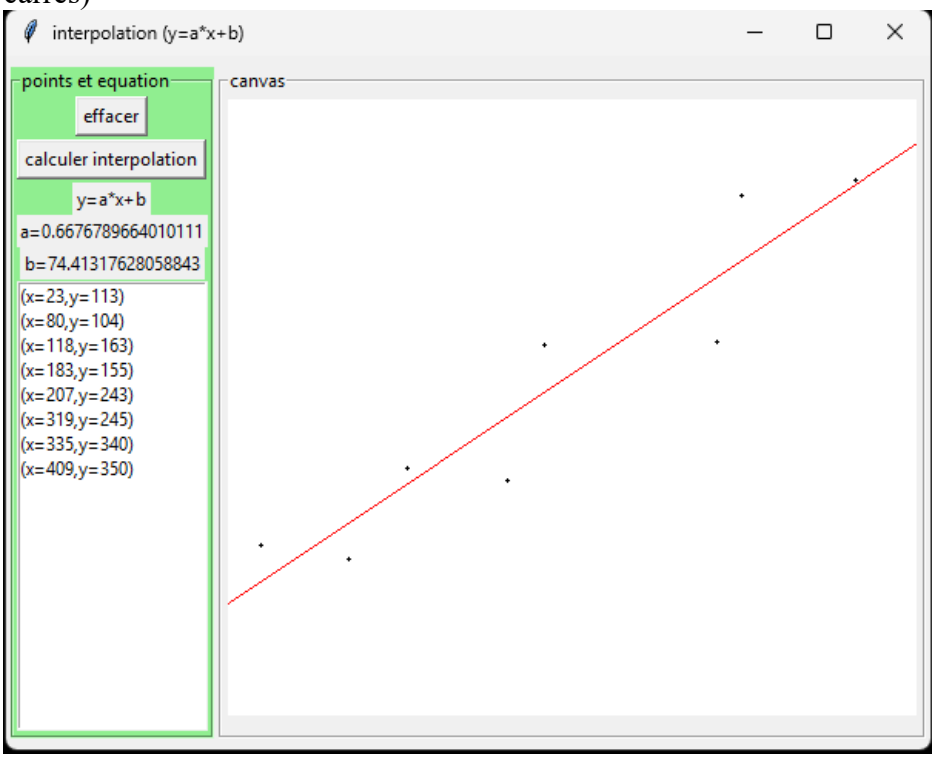
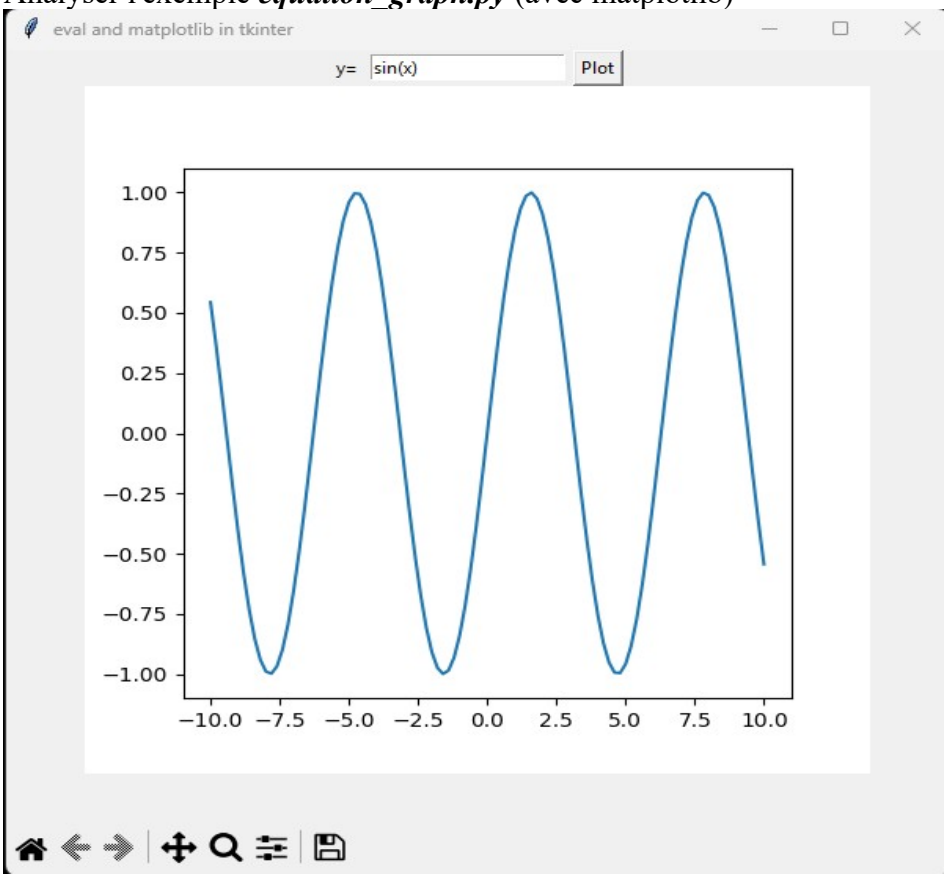
Exo F1 (venv)	Utilisation de la librairie markdown dans venv (sans pipenv) ---------- soit f1.py comportant ce code : *import markdown* *my_html_text = markdown.markdown("**Bonjour**")* *print("my_html_text",my_html_text)* Exécuter python f1.py en direct génère normalement une erreur de ce type : *ModuleNotFoundError: No module named 'markdown'* Lancer la commande ***pip list*** pour visualiser que le module markdown n'est pas dans la liste des modules installés au niveau global Créer un env python virtuel : python -m venv .venv Activer ce .venv via la commande adéquate (de windows ou linux ou autre) et attendre le prompt (.venv) Après le prompt (.venv) lancer la commande suivante : pip install markdown puis l'application f1.py : python f1.py Après le prompt (.venv) lancer la commande deactivate pour sortir de l'environnement virtuel Vérifier via pip list que le module markdown n'est pas installé au niveau global Réactiver .venv (sous windows ou linux ou autre) Après le prompt (.venv) , relancer pip list Après le prompt (.venv) ; relancer python f1.py Après le prompt (.venv) ; relancer deactivate
Exo F2 (pipenv)	Utilisation de la librairie markdown avec pipenv ---------- soit f2.py comportant ce code : *import markdown* *my_html_text = markdown.markdown("*Hello*")* *print("my_html_text",my_html_text)* Exécuter python f2.py en direct génère normalement une erreur de ce type : *ModuleNotFoundError: No module named 'markdown'* ---------- Dans un terminal, au niveau global, lancer la commande suivante : ***python -m pip install pipenv*** Verifier l'installation via la commande suivante : ***python -m pipenv --version*** Lancer ensuite cette commande : **python -m pipenv install markdown** puis enfin cette nouvelle commande : ***python -m pipenv run f2.py*** Vérifier via **pip list** que le module markdown n'est toujours pas installé au niveau global Vérifier via la commande ***python -m pipenv run pip list*** que markdown est installé au sein de l'environnement virtuel géré par pipenv . 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)
Exo f3 (essais libres)	Quelques essais libres de syntaxes avancées au sein de f3.py
Exo f4 (comprehension)	A partir du contenu initial suivante de f4.py *phrases=[* *"Bonjour",* *"Python est un serpent efficace",* *"sans majuscule au debut",* *"Ok , tout va bien",* *"pasBien car commencant par une minuscule"* *]* *print("toutes_les_phrases",phrases)* Ajouter des listes en compréhension de manière à calculer et afficher : - tailles_de_toutes_les_phrases - phrases_avec_majuscules (sur premier caractère) - tailles_phrases_majuscules (sur premier caractère)
Exo f5 (surcharge opérateur)	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'operateur > **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") NB : Lors d'un premier essai, sans amélioration de la classe Compte dans e3_Compte.py , on obtient le message d'erreur suivant : *TypeError: '>' not supported between instances of 'Compte' and 'Compte'* 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
Exo f6 (décorateur)	En partant de ce code initial dans f6.py *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}<<<"* 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 .

	Appliquer ensuite ce décorateur sur une copie/variante encadrer2() de la fonction encadrer1(). Appeler en boucle encadrer1() et encadrer2() sur ceci *liste_phrases_avec_bosses = [* *"JeVeuxMaMaman",* *"PythonQueJaime",* *"PhraseQuiVaBien"* *]* pour normalement obtenir un résultat de ce genre : ['>>>JeVeuxMaMaman<<<', '>>>PythonQueJaime<<<', '>>>PhraseQuiVaBien<<<'] ['>>>je_veux_ma_maman<<<', '>>>python_que_jaime<<<', '>>>phrase_qui_va_bien<<<']
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)	Au sein de f9.py tab_ref_prod = ["Aa123xy" , "bb7ttt" , "ab674Ua" , "45yy", "au5" , "ry345yxt" , "zy145Yx"] 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. On considérera qu'une référence correcte est formulée par : 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)	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. Appliquer cela sur un tableau de valeurs de ce type tab_val = ["12.6" , "14,78" , "3.6" , "67,89", "1.4" , "124,677" , "134,67"]
Exo f11 (essais libres)	Essais libres sur expressions régulières ou autres syntaxes avancées de python

2.9. Librairies 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

Exo H1 (sqlite et sqlAlchemy en mode sql)	NB : au sein de l'exercice h1, on s'appuiera sur l'api SqlAchemy. **pip install sqlachemy** Ecrire un script python **h1_init_db.py** qui va initialiser une base de données au format sqlite et d'url sqlite:///./minibank_db avec une table compte(numero,label,solde) compatible avec la structure de e3_Compte.py Ecrire un script **h1_crud.py** qui va : - 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	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" . URLs proposées: http://127.0.0.1:5000/static/index.html et ***http://127.0.0.1:5000/minibank-api/v1/comptes*** NB : on pourra s'appuyer sur des constructions partielles des exercices e3 (classe Compte) et h1 (stockage en base de données) Pour effectuer des tests, on pourra éventuellement s'appuyer sur une application de type postman (ou un équivalent) . Démarrage proposé : **flask --app i1_minibank_api run**
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 (ThreadPool Executor et Future)	Énoncé à rédiger et solution à préparer
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