Exercice 1 :

On se donne une liste stock correspondant à un inventaire des produits d'un magasin.

```
>>> stock
 [('Pommes', 10),
  ('Carottes', 5),
  ('Radis', 23),
  ('Lentilles', 534),
  ('Poivrons', 12)]
>>> maj_inventaire(stock,'Carottes',3)
>>> stock # il n'y a maintenant plus que 2 carottes !
 [('Pommes', 10),
  ('Carottes', 2),
  ('Radis', 23),
  ('Lentilles', 534),
  ('Poivrons', 12)]
```

1) Écrire une procédure `maj_inventaire(stock : list , produit : str, n : int )` qui modifie la liste *stock* en supprimant n occurences du produit *produit*.

2) Modifier la fonction pour qu'elle lance IndexError si le produit n'existe pas et lance ValueError s'il n'y a pas assez de produits disponibles.

3) Écrivez une procédure `achat(stock :list ,produit : str ,n : integer)` qui fait appel à la fonction *maj_inventaire* et qui rattrape les éventuelles exceptions pour afficher un des trois messages suivant :

- " Merci pour votre achat."
- " Produit inexistant. "
- " Produit en quantité insuffisante."

Exercice 2 :

L'objectif de cet exercice est d'implémenter une fonction permettant la saisie d'un code PIN par l'utilisateur, tout en limitant le nombre de tentatives possibles (2eme partie).

Vous utiliserez :

- La gestion des exceptions avec un *try-except*.
- Une boucle pour redemander l'entrée en cas d'erreur.
- Un mécanisme limitant le nombre d'essais (2eme question).
- Un retour sous forme de tuple indiquant si la saisie a réussi ou échoué (2eme question).

1) L'instruction *input* renvoie une chaîne de caractère. Pour saisir un entier il faut convertir cette chaîne en type *int*. Si l'utilisateur saisit autre chose qu'un entier, cette conversion lèvera une exception.

Vous devez donc écrire une fonction `entree_code_boucle()-> int :` qui :

- Demande le code PIN et le renvoie dans le type *int* si la conversion est possible.
- Rattrape l'exception en cas d'erreur de saisie.
- En cas d'exception, affiche un message signalant l'erreur.
- Redemande le code PIN tant que la saisie est incorrecte.

Votre fonction ne doit pas vérifier que le code PIN est un entier positif sur 4 chiffres.

Cette fonction sera améliorée dans la prochaine question avec un nombre d'essais maximum.

Exemple d'éxécution :

```
>>> entree_code_boucle()
Entrer votre code PIN : abc
Erreur de saisie. Recommencez.
Entrer votre code PIN : 1234
Vous avez saisi 1234.
```

2) Vous améliorez la procédure précédente en écrivant une **NOUVELLE** fonction

`entree_code_Nessais(nb : int) -> tuple[int, bool]` qui :

- Demande à l'utilisateur de saisir un code PIN (entier !)
- Tente de convertir l'entrée en un entier.
- Si la conversion réussit, retourne un tuple contenant l'entier saisi et True.
- Rattrape l'exception et affiche un message qui indique le nombre d'essais restants.
- Autorise un nombre limité d'essais (nb tentatives).
- Au bout des nb tentatives affiche un message et retourne (0, False), pour un traitement ultérieur.

Votre fonction ne doit pas vérifier que le code PIN est un entier positif sur 4 chiffres.

Exemple d'éxécution où l'utilisateur réussit (ici la fonction renvoie (1234,True)) :

```
>>> entree_code_Nessais(3)
Entrer votre code PIN : 1234
Vous avez saisi 1234.
```

Exemple d'éxécution où l'utilisateur échoue (la fonction renvoie (0, False)) :

```
>>> entree_code_Nessais(3)
Entrer votre code PIN : abc
Essai 1 sur 3 : Erreur de saisie. Recommencez.
Entrer votre code PIN :  zut
Essai 2 sur 3 : Erreur de saisie. Recommencez.
Entrer votre code PIN : mince
Essai 3 sur 3 : Erreur de saisie.
Vous avez atteint le nombre maximum d essais possible : 3.
```

Exercice 3 : L'objectif de cet exercice est de charger les données contenues dans un fichier au format csv.

Nous allons utiliser le fichier csv fourni contenant les données de l'exemple vu en cours (CM4) pour le code factorisé. La visualisation avec Libre Office par exemple donne les rendus suivants (à gauche sans mise en forme et à droite avec) :

	A	B	C	D	E
1	Nom	Type	Dimension 1	Dimension 2	
2	R1	rectangle	5	10	
3	R2	rectangle	7	3	
4	C1	cercle	4	nil	
5	C2	cercle	6	nil	
6	T1	triangle	8	5	
7	T2	triangle	6	4	
8					
9					
10					

	A	B	C	D	E
1	**Nom**	**Type**	**Dimension 1**	**Dimension 2**	
2	R1	rectangle	5	10	
3	R2	rectangle	7	3	
4	C1	cercle	4	nil	
5	C2	cercle	6	nil	
6	T1	triangle	8	5	
7	T2	triangle	6	4	
8					
9					

Pour charger le fichier vous allez écrire une fonction qui lit un fichier csv (voir le CM3). La signature de la fonction est la suivante :

```
1 def chargeFichierCsv(nom : str) -> list[list[str]] :
```

L'utilisation de de cette fonction sur le fichier d'exemple (data_formes.csv), avec quelques retours à la ligne après chaque élément de la liste principale est le suivant :

```
1 >>>chargeFichierCsv("data_formes.csv")
2 [['Nom', 'Type', 'Dimension 1', 'Dimension 2'],
3  ['R1', 'rectangle', '5', '10'],
4  ['R2', 'rectangle', '7', '3'],
5  ['C1', 'cercle', '4', 'nil'],
6  ['C2', 'cercle', '6', 'nil'],
7  ['T1', 'triangle', '8', '5'],
8  ['T2', 'triangle', '6', '4']]
```

Veuillez noter, que la fonction renvoie une liste qui contient 7 éléments. Le premier élément (une liste) correspond à l'entête de description des paramètres. `['Nom', 'Type', 'Dimension 1', 'Dimension 2']`.

Le deuxième élément de la liste et tous les suivants `['R1', 'rectangle', '5', '10']` contiennent les données pour une forme géométrique. Chaque données élémentaire est une chaîne de caractères.

1) Travail à faire : écrire la fonction chargeFichierCsv dont la signature est :

```
1 def chargeFichierCsv(nom : str) -> list[list[str]] :
```

2) Tester votre fonction sur le fichier fourni : "data_formes.csv".

3) Le code suivant est une version modifiée du code présenté en CM4 pour montrer un code bien factorisé à l'aide de fonctions. Ici vous constaterez l'ajout d'une procédure *CalculsAvecCr*. Cette procédure lit la liste de données obtenue en retour de la fonction chargeFichierCsv. Les calculs sont effectués pour chaque forme et le résultat est stocké dans data (voir ligne 20 : data[k].append(str(aire))).

```
1  def CalculsAvecCr( data : list[list[str]]) -> None  :
2      data[0].append('Aire')    # ajoute l'intitulé des résultats
3      for k in range(1,len(data)):
4          parametre :list[str] = data[k] # liste des paramètres
5                                       #pour une seule forme
6          nom=parametre[0] # extraction du nom de la forme
7          nature = parametre[1] # nature de la forme rectangle, triangle ou cercle
8          if nature == "rectangle":
9              l = int ( parametre[2] )
10             L = int ( parametre[3] )
11             aire = aire_rectangle( l , L )
12         elif nature == "cercle":
13             rayon = int (parametre[2])
14             aire = aire_cercle(rayon)
15         elif nature == "triangle":
16             base    = int ( parametre[2] )
17             hauteur = int ( parametre[3] )
18             aire = aire_triangle(base, hauteur)
19         aire = int(aire*100)/100 # arrondis à 2 chiffres
20         data[k].append(str(aire))  # ajoute la valeur de l'aire dans data
21         texte = f"Aire de {nom}=  {aire} m^2"
22         print(texte) # affichage pour voir.
```

Travail à faire :

- lire, comprendre cette procédure
- éxécuter cette procédure (vous trouverez les procédures dans le code sur la page suivante : aire_rectangle(), aire_cercle() et aire_triangle())
- passer à la question suivante.

4) Collez votre fonction (chargeFichierCsv) dans le fichier EX3.py présenté ci-dessous ainsi que le code de la page précédente pour la fonction CalculsAvecCr.

```python
import csv
def aire_rectangle(long, larg):
    return long * larg

def aire_cercle(r):
    return 3.14159 * r * r

def aire_triangle(b, h):
    return (b * h) / 2

def chargeFichierCsv(nom : str) -> list[list[str]] :
    # votre code écrit à la question 1 ICI

def CalculsAvecCr( data : list[list[str]])-> list[str] :
    # le code de la page précédente

def ecritFichierResultat(nomFichierSortie : str, data :list[list[str]] ) :
    #  votre future code de la question 5
    # en attendant laisser pass ci-après.
    pass

nomFichier : str ="data_formes.csv"
# liste pour recueillir le contenu du fichier compte rendu.
data=chargeFichierCsv(nomFichier)
CalculsAvecCr(data)
print(data)
nomFichierSortie : str ="resultats_"+nomFichier # nom du fichier à écrire.

ecritFichierResultat(nomFichierSortie,data) # ne fera rien avec pass.
```

5) Testez votre programme (EX3.py) sur le fichier data_formes et vérifiez que vous obtenez le résultat suivant :

(Remarque pour raisons de lisibilité, des retours à la ligne ont été ajoutés).

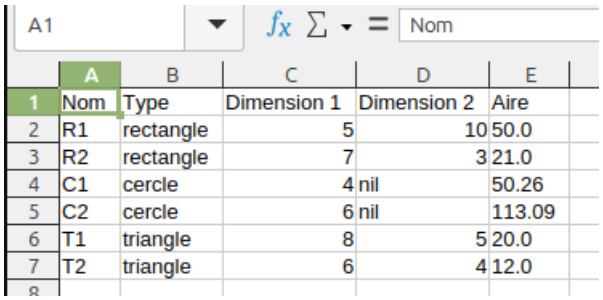
```
>>> %Run EX3.py
Aire de R1=  50.0 m^2
Aire de R2=  21.0 m^2
Aire de C1=  50.26 m^2
Aire de C2=  113.09 m^2
Aire de T1=  20.0 m^2
Aire de T2=  12.0 m^2
[['Nom', 'Type', 'Dimension 1', 'Dimension 2', 'Aire'],
['R1', 'rectangle', '5', '10', '50.0'],
['R2', 'rectangle', '7', '3', '21.0'],
['C1', 'cercle', '4', 'nil', '50.26'],
['C2', 'cercle', '6', 'nil', '113.09'],
['T1', 'triangle', '8', '5', '20.0'],
['T2', 'triangle', '6', '4', '12.0']]
```

6) Ecrire le code de la procédure dont la signature est la suivante.

```
def ecritFichierResultat(nomFichierSortie : str, data :list[list[str]] ) -> None :
        # vos quelques lignes de code ICI.
        print("Fichier CSV écrit avec succès.") #à écrire en fin de procédure
```

Tester votre code et vérifier qu'un nouveau fichier est bien crée.

7) Ouvrir le fichier resultat_data_formes.csv avec libre office. Vérifier que le contenu est similaire aux rendus suivants :



	A	B	C	D	E
1	Nom	Type	Dimension 1	Dimension 2	Aire
2	R1	rectangle	5	10	50.0
3	R2	rectangle	7	3	21.0
4	C1	cercle	4	nil	50.26
5	C2	cercle	6	nil	113.09
6	T1	triangle	8	5	20.0
7	T2	triangle	6	4	12.0
8					

	A	B	C	D	E
1	Nom	Type	Dimension 1	Dimension 2	Aire
2	R1	rectangle	5	10	50.0
3	R2	rectangle	7	3	21.0
4	C1	cercle	4	nil	50.26
5	C2	cercle	6	nil	113.09
6	T1	triangle	8	5	20.0
7	T2	triangle	6	4	12.0
8					

Normallement le résultat des calculs doit être visible dans la colonne E du tableau.

8) travail à faire :

- Copiez le fichier data_formes.csv sous un nom de votre choix.
- Dans ce nouveau fichier, remplacez les valeurs 5 et 10 du rectangle R1 par 5.5 et 10.5 (avec un point et pas une virgule !).
- Modifiez la variable *nomFichier* avec le nom de votre nouveau fichier.
- Exécutez EX3.py.
- Patatras !!! Une exception du type ValueError : invalid etc... est levée par la procédure *CalculsAvecCr*
- Corrigez le code de votre enseignant !
- Testez votre code corrigé, et vérifiez que les résultats sont corrects.
- Ajoutez les types des variables dans les signatures des fonctions suivantes du programme.

```
1 def aire_rectangle(long, larg):
2     return long * larg
3
4 def aire_cercle(r):
5     return 3.14159 * r * r
6
7 def aire_triangle(b, h):
8     return (b * h) / 2
```

- Complétez votre fichier d'entrée et ajoutez des formes géométriques avec de nouvelles dimensions de votre choix.

Exercice 4 : Importation de module. Objectif de cet exercice : développer et tester des fonctions dans des modules, puis les importer par la suite dans un autre programme.

1) travail à faire :

- Copier , coller le code suivant dans un fichier vide.
- Enregistrer ce fichier sous le nom de somme.py.
- Exécuter ce programme. Utiliser d'autres valeurs numériques pour x et y.
- Dans l'interpréteur python regarder le retour de la commande *print(__name__)* (vu en CM3).

somme.py

```python
def addition (a : int, b : int)-> int :
    return a+b

def difference(a : int, b : int)-> int:
    return a-b

if __name__=="__main__" :
    # test des fonctions ici
    x=3
    y=5
    s : int =addition(x,y)
    d : int =difference(x,y)
    print("Ce code s'est éxécuté depuis somme.py")
    print(f"La sommme  {x} + {y} est {s} .")
    print(f"La différence  {x} - {y} est {d} .")
```

2) travail à faire (analogue à celui de la question 1) :

- Copier , coller le code suivant dans un fichier vide.
- Enregistrer ce fichier sous le nom de produit.py (dans le même répertoire que somme.py)
- Exécuter ce programme. Utiliser d'autres valeurs numériques pour x et y.
- Dans l'interpréteur python regarder le retour de la commande *print(__name__)* (vu en CM3).

produit.py

```python
def multiplication (a : int, b : int)-> int :
    return a*b

def quotientEntier(a : int, b : int)-> int :
    return a//b

if __name__=="__main__" :
    # test des fonctions ici
    x:int = 3
    y:int = 5
    p : int =multiplication(x,y)
    q : int =quotientEntier(y,x)
    print(f"Le produit de {x} * {y} est {p}.")
    print(f"Le quotient de {y} par {x} est {q}.")
```

3) travail à faire :

- Ouvrir un fichier vide.
- L'enregistrer dans le même répertoire que les fichiers somme.py et produit.py.
- importer et renommer les fonctions des modules précédents avec les commandes suivantes.
- Exécuter ce programme avec l'exemple fourni.
- Compléter ce code avec des exemples de votre choix pour utiliser les 4 fonctions importées.

EX4.py

```
from somme   import addition       as add
from somme   import difference     as diff
from produit import multiplication as mult
from produit import quotientEntier as  qE

a :int=12
b :int=24
# exemple :
s :int = add(a,b)
assert(s==36)
```

Vous avez importé renommé et utilisé 4 fonctions écrites dans des modules extérieurs à ce code.

4) travail à faire :

- Ouvrir un fichier vide.
- L'enregistrer sous "EX4_bis.py" dans le même répertoire que les fichiers somme.py et produit.py
- Compléter ce code avec des exemples de votre choix pour utiliser les 4 fonctions importées.

EX4_bis.py

```
import somme
import produit

a :int=12
b :int=24
# exemple :
s :int = somme.addition(a,b) # nouvelle synthaxe
assert(s==36)
```

5) Dans le code EX4_bis.py ajouter une nouvelle fonction dont la signature est la suivante :

```
def addition (a : int)-> int :
    """ ajoute 1 à a"""

assert(addition(3)==4 )
```

6) Compléter le code EX4_bis.py avec les commandes suivantes. Lancer le code et vérifier si il y a conflit de nom ou pas étant donné qu'il y a 2 fonctions "addition".

```
s :int = somme.addition(a,b)
assert(s==36)

s2 :int = addition(9)
assert(s2==10)
```

Conclusion : Avec l'importation de modules, vous disposez de nouvelles fonctions que vous pouvez appeler. Dans le fichier EX4_bis.py vous avez 2 fonctions d'addition bien distinctes grâce à leur nom qui les différencie : "somme.addition()" et "addition()". Avec l'import de module, vous avez ajouté à l'espace de nommage une nouvelle fonction : "somme.addition()" sans conflit de nom avec des fonctions existantes.

Vous pourriez avoir également des importations comme ci-après.

```
import module_UN
import module_DEUX

module_UN.fonction1()
module_DEUX.fonction1()
```

L'import de fonctions permet de développer et tester des fonctions à part ou d'importer des fonctions développées (et validées bien sûr) par d'autres développeurs Python.