Exercice 1 :

1) La fonction demandée est la suivante :

question1.py

```
def maj_inventaire(stock : list , produit : str, n : int ) -> None :
    for k in range(len(stock)) :
        element : tuple =stock[k]
        prod : str =element[0]
        nb : int = element[1]

        if prod==produit :
            stock[k]=(produit, nb-n)
            print("Stock modifié")  # non demandé, mais aide le développeur.
```

Vérification :

```
>>> stock
[('Pommes', 2),
 ('Carottes', 5),
 ('Radis', 23),
 ('Lentilles', 534),
 ('Poivrons', 12)]
>>> maj_inventaire(stock,'Carottes',3)
Stock modifié
>>> stock
[('Pommes', 2),
 ('Carottes', 2),
 ('Radis', 23),
 ('Lentilles', 534),
 ('Poivrons', 12)]
>>>
```

2) La fonction modifiée pour qu'elle lance IndexError si le produit n'existe pas et lance ValueError s'il n'y a pas assez de produits disponibles est la suivante :

```
def maj_inventaire(stock : list , produit : str, n : int ) -> None :
    for k in range(len(stock)) :
        element : tuple =stock[k]
        prod : str =element[0]
        nb : int = element[1]
        if prod==produit :
            if n>nb :
                message=f"Vous demandez {n} {produit}, et il n'y en a que {nb}. "
                raise ValueError(message)
            else :
                stock[k]=(produit, nb-n)
                print("Stock modifié")
                return
    # le produit n'a pas été trouvé donc erreur.
    message : str=f"Le produit {produit} n est pas en stock"
    raise IndexError(message)
```

Vérification , testons les 3 cas possibles :

```
>>> maj_inventaire(stock  ,'Pommes', 1)
Stock modifié

>>> maj_inventaire(stock  ,'Pommes', 100)
'Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise ValueError(message)
ValueError: Vous demandez 100 Pommes, et il n y en a que 1.'

>>> maj_inventaire(stock  ,'Chocolat', 2)
'Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    raise IndexError(message)
IndexError: Le produit Chocolat n est pas en stock.'
```

3) 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."

```
def  achat(stock: list ,produit : str ,n : int ) -> None:

    try :
        maj_inventaire_raise(stock  , produit , n  )
        print(" Merci pour votre achat.")
    except ValueError :
        print(" Produit en quantité insuffisante.")
    except IndexError:
        print(" Produit inexistant.")
```

Vérifions les 3 cas. On voit que le programme se termine en affichant des messages définis par le développeur et non des exceptions levées par le système.

```
>>> achat(stock  ,'Pommes', 8)
Stock modifié
 Merci pour votre achat.
>>> achat(stock  ,'Pommes', 800)
 Produit en quantité insuffisante.
>>> achat(stock  ,'P', 800)
 Produit inexistant.
```

Exercice 2 :

1) La fonction `entree_code_boucle()-> int :` :

- 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.

La fonction ne vérifie pas que le code PIN est un entier positif sur 4 chiffres.

```
1 def entree_code_boucle()-> int :
2     while True :
3         try :
4             n=int(input("Entrer votre code PIN : "))
5             print (f"Vous avez saisi {n}.")
6             return n
7         except :
8             print('Erreur de saisie. Recommencez.')
```

Vérification :

```
1 >>> entree_code_boucle()
2 Entrer votre code PIN : abc
3 Erreur de saisie. Recommencez.
4 Entrer votre code PIN : 1234
5 Vous avez saisi 1234.
6 >>>
```

2) La fonction `entree_code_Nessais(nb : int) -> tuple[int, bool]`

- 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.

Cette fonction ne vérifie pas que le code PIN est un entier positif sur 4 chiffres.

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

    for n in range(nb):
        try :
            code =int( input("Entrez votre code PIN : ") )
            print (f"Vous avez entré {code}")
            return (code , True )
        except :
            print(f'Erreur de saisie.Essai {n+1} sur {nb} Recommencez.')
    print(f"Vous avez atteint le nombre maximum d'essais possible : {nb}.")
    return ( 0, False)
```

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 :

1) la fonction chargeFichierCsv dont la signature est :

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

est :

```
import csv

def chargeFichierCsv(nom : str) -> list[list[str]] :
    # Liste pour stocker les lignes
    lignes: list[str] = []
    # ouverture
    fichier  = open(nom, "r", encoding="utf-8")
    reader = csv.reader(fichier) # lecture
    for ligne in reader:
        lignes.append(ligne) # Ajouter chaque ligne à la liste
    fichier.close() #fermeture.
    return lignes
```

2) Testons la fonction sur le fichier fourni "data_formes.csv".

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

3) A faire directement sur votre ordi. Pas de correction.

4) A faire directement sur votre ordi. Pas de correction.

5) Testons le programme (EX3.py) sur le fichier data_formes (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) Ecrivons le code de la procédure dont la signature est la suivante.

```
def ecritFichierResultat(nomFichierSortie : str, data :list[list[str]] ) -> None :

        print("Fichier CSV écrit avec succès.") #à écrire en fin de procédure
```

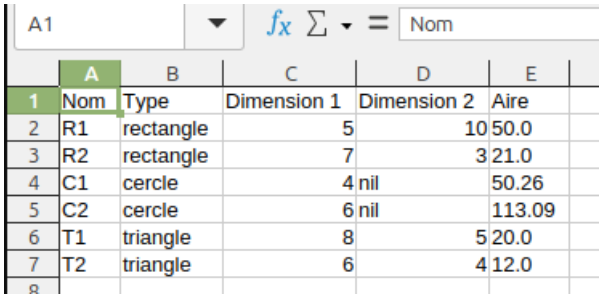
```
def ecritFichierResultat(nomFichierSortie : str, data :list[list[str]] ) :
    # Ouvrir le fichier en mode écriture
    fichier = open( nomFichierSortie, "w", encoding="utf-8", newline="")

    # Créer un writer CSV
    writer = csv.writer(fichier)
    # Écrire les lignes
    for ligne in data:
        writer.writerow(ligne)

    # Fermer le fichier
    fichier.close()

    print("Fichier CSV écrit avec succès.")
```

7) Le contenu du fichier resultat_data_formes.csv ouvert avec libre office est :



	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

Le résultat des calculs est visible dans la colonne E du tableau.

8) Le code corrigé (qui calcule avec des flottants et non des entier ou int) est le suivant . Les corrections sont sur les lignes 12 et 13 ou int a été remplacé par float.

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

Les fonctions avec les signatures bien renseignées sont :

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

Exercice 4 : Importation de module.

1) *print(__name__)* renvoie __main__ (vu en CM3).

2) *print(__name__)* renvoie encore __main__ (vu en CM3).

3) Le code de l'énoncé complété avec des exemples pour utiliser les 4 fonctions importées est

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

s :int = add(a,b)
assert(s==36)

d :int = diff(b,a)
assert(d==12)

p :int = mult(a,b)
assert(p==288)

q :int = qE(b,a)
assert(q==2)
```

4) Le code proposé est complété avec des exemples de pour utiliser les 4 fonctions importées.

EX4_bis.py

```
1  import somme
2  import produit
3
4  a :int=12
5  b :int=24
6
7  s :int = somme.addition(a,b)
8  assert(s==36)
9
10 d :int = somme.difference(b,a)
11 assert(d==12)
12
13 p :int = produit.multiplication(a,b)
14 assert(p==288)
15
16 q :int = produit.quotientEntier(b,a)
17 assert(q==2)
```

5) La fonction addition (qui ajoute 1 à l'entrée)s'écrit avec différents tests :

```
import somme
import produit
a :int=12
b :int=24

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

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

s2 :int = addition(3)
assert(s2==4)

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

Dans ce code il y a 2 fonctions addition, celle définie localement et celle importée du module qui s'appelle somme.addition().

6) Il n'y a pas conflit de nom 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.