Exercice 1 :

Objectif de l'exercice : Calculer la somme $1 + 2 + ... + n$ pour un entier n donné.

1) Écrire (et tester) la fonction qui calcule et renvoie la somme à l'aide d'une boucle for. Ecrire plusieurs tests en complément de ceux écris ci dessous.

```
def   somme (n : int ) -> int :
        """Renvoie la somme 1+2+...+n"""

assert (somme(1)==1 )
assert (somme(2)==3 )
```

Faites quelques appels de la fonction dans l'interpréteur python.

SHELL

```
>>> somme(1)

1

>>>
```

2) Afficher le résultat avec une phrase d'explication.

```
def   somme (n : int ) -> int :
        """Renvoie la somme 1+2+...+n"""
        # votre super code ICI
assert (somme(1)==1 )
assert (somme(2)==3 )

n : int = 10 # ou une valeur de votre choix
# appel de la fonction
s : int = somme(n)
print( .......# A compléter par vos soins
```

Exercice 2 :

Objectif : calculer la sommme d'argent obtenue après l'avoir laissée en dépôt dans une banque avec un taux d'intérêt annuel de t % après n années. Dit autrement, au bout d'un an, la somme S_0 de départ est augmentée de t%, on obtient S_1. Après 2 ans la somme de l'année précédente(S_1) est augmentée de t%, on obtient S_2. Et ainsi de suite d'année en année.

1) Sur papier, résoudre le problème. Vérifier que l'exemple suivant est correct : au bout de 5 années, pour une somme de 1 000 euros placée avec un taux annuel de 2%, la banque retournera environ 1 104,08 euros.

2) Compléter le code python ci-dessous pour résoudre le problème.

```python
def calculSomme(####Compléter le code ici en écrivant les bons types !!
        #### écrire votre code ici (avec une boucle for )
        ...

# Définition des entrées
duree=5
taux = 2 # en pourcent
sommeInitiale = 1000

# appel de la fonction
sommeFinale =...
# affichage du résultat ci après :
print(f"La somme de {sommeInitiale} euros "
        f"devient au bout de {duree} ans : {sommeFinale:.2f} euros")
```

3) Et les tests avec assert ???

Pour tester la fonction il est tentant et naturel dans un premier d'écrire :

assert (calculSommme(#vos paramètres) == 1 104.08)

mais ici 1104.08 est l'approximation d'un résultat. Donc vous ne pouvez qu'encadrer votre résultat par 2 valeurs par exemple.

assert (1104 < calculSommme(#vos paramètres) and calculSomme(#vos paramètres) < 1105)

Ajouter des test de la fonction avec assert dans votre code, si possible en utilisant d'autres valeurs que 5 ans, 2 pourcents et 1000 euros.

Exercice 3 :

Voici le résultat de l'éxécution d'un programme affiché dans l'interpréteur python (ici avec l'ajout des numéros de lignes)

SHELL

```
1 >>> %Run EX_tables_de_multiplication.py
2  2 x 2= 4   3 x 2= 6   4 x 2= 8
3  2 x 3= 6   3 x 3= 9   4 x 3=12
4  2 x 4= 8   3 x 4=12   4 x 4=16
5  2 x 5=10   3 x 5=15   4 x 5=20
6  2 x 6=12   3 x 6=18   4 x 6=24
7 xxxxxxxxxxxxxx
8  2 x 2= 4   3 x 2= 6   4 x 2= 8   5 x 2=10   6 x 2=12
9  2 x 3= 6   3 x 3= 9   4 x 3=12   5 x 3=15   6 x 3=18
10 2 x 4= 8   3 x 4=12   4 x 4=16   5 x 4=20   6 x 4=24
```

Le code python correspondant vous est donné ci-après. Malheureusement il ne reste que les commandes d'affichage correctement paramétrées. Sauriez-vous ajouter les commandes for sur les lignes 1 ,2 puis 6 et 7 pour obtenir exactement l'affichage des tables de multiplication présentées ci-dessus.

```
1 #    code manquant ici
2 #    code manquant
3         print( f"{j:2} x{i:2}={i*j:2}", end="  ")
4     print()
5 print('xxxxxxxxxxxxxx')
6 #    code manquant ici
7 #    code manquant  ici
8         print( f"{j:2} x{i:2}={i*j:2}", end="  ")
9     print()
```

Pour reporter ces lignes de codes dans l'éditeur Thonny, vous pouvez sélectionner avec la souris le code texte et le coller dans l'éditeur. Les numéros de lignes seront peut être également copiés (à enlever pour python !).

Le codage {j:2} dans le f-string est expliqué sur la prochaine page.

Le codage j:2} dans le f-string veut dire que la valeur de j sera affichée sur 2 espaces (pour mieux aligner les tables). Vous pouvez le tester avec les commandes suivantes pour vous en convaincre.

SHELL

```
>>>
k : int = 3
print(f"{k:1} est affiché sur 1 caractère avec   :1")
print(f"{k:2} est affiché sur 2 caractères :2")
print(f"{k:6} est affiché sur 6 caractères :6")
print(f"{k:10} est affiché sur 10 caractères :10")
print("la table de multiplication sera bien présentée avec les :2  .")
>>>
3 est affiché sur 1 caractère avec   :1
 3 est affiché sur 2 caractères :2
     3 est affiché sur 6 caractères :6
         3 est affiché sur 10 caractères :10
la table de multiplication sera bien présentée avec les :2  .
```

Exercice 4 : Vous allez gérer un petit stock d'une pharmacie. Ce stock est représenté par un dictionnaire dont les clés sont les noms des médicaments et les valeurs le nombre de boite.

1) Vous allez créer le dictionnaire dans l'éditeur :

```
NbreDeBoites : dict = {
                      "Joliprune":10,
                      "Iffelegalan":3,
                      "Dofolgon":1,
                      "Saleuvopa":0,
                      "Kardabof":15,
                      "Pasfaim":20,
                      "Pabondutout":11,
                      "Voltampere":4,
                      "Methaveste":5,
                      "Kilédrol":6,
                      "IgrecPrim":3,
                      "Puréecétrobon":2,
                      "Profdinfo":1
                       }
```

2) Ecrire (et tester) les fonctions suivantes :

```
def afficheStock(dico : dict)-> None :
    """Affiche le dictionnaire"""
```

Voici le début de l'affichage dans l'interpréteur python, vous pouvez changer la phrase d'explication si vous le souhaitez.

SHELL

```
>>> %Run EX_medicaments.py
Il y a 10 boite(s) de Joliprune.
Il y a 3 boite(s) de Iffelegalan.
...
```

3) Ecrire (et tester) les fonctions suivantes :

```
def listeBoitesArenouveller(dico : dict , seuil : int ) -> list[str] :
    """Cette fonction renvoie la liste de médicament du dictionnaire
            dont le nombre de boite est inférieur ou égal au seuil"""

def afficheListe (message :str , Liste) -> None :
    """Affiche le contenu de la liste précédé du message"""

# exemple de test possible
Liste=listeBoitesArenouveller(NbreDeBoites,1)
afficheListe("Les boites à renouveller sont : ", Liste )
```

Voici le début de l'affichage dans l'interpréteur python, vous pouvez changer le message si vous le souhaitez.Vous devez présenter un affichage lisible par un non-informaticien. Un affichage dans le style python ['Dofolgon','Saleuvopa'] n'est pas correct. La sortie suivante vous montre un exemple compact, sur une ligne, et avec un point à la fin de la phrase ! A vous de voir.

SHELL

```
>>> %Run EX_medicaments.py
 Les boites à renouveller sont :    Dofolgon,  Saleuvopa,  Profdinfo.
```

4) Ecrire la procédure suivante pour consulter le stock pour un médicament.

```
def consulteStock(medicament :str , dico : dict)-> None :
    """Cette fonction affiche un message avec le nombre de boite
    pour le medicament passé en paramêtre. Si le médicament n'est
    pas en stock un message le signale ! """
```

Exemple d'utilisation ou de test rapide (au moins 3 tests)

SHELL

```
>>> consulteStock('Poudre de sorcière',NbreDeBoites)
Ce médicament ( Poudre de sorcière ) n existe pas dans le stock.
>>> consulteStock("IgrecPrim",NbreDeBoites)
Il y a 3 boite(s) de IgrecPrim.
>>> consulteStock("Saleuvopa",NbreDeBoites)
La référence existe mais il n y a plus de Saleuvopa.
```

5) Ecrire les procédures suivantes pour modifier le stock.

```
def ajouteBoite( medicament : str , nb :int, dico : dict) -> None :
    """Augmente le stock pour un médicament donné"""

def retireBoite( medicament : str , nb :int, dico : dict) -> None :
    """Reduit le stock , ou affiche un message et ne modifie
    pas le stock si il n'y a pas assez de boites"""
```

Exemple de vérification (avec une fonction déja testée !!) , d'autres vérifications sont possibles, à vous de tester ce qui vous paraît judicieux.

SHELL

```
>>>  consulteStock("Pasfaim",NbreDeBoites)
Il y a 20 boite(s) de Pasfaim.
>>> ajouteBoite("Pasfaim" ,100,NbreDeBoites)
>>> consulteStock("Pasfaim",NbreDeBoites)
Il y a 120 boite(s) de Pasfaim.
```

6) Ecrire (et tester) les fonctions suivantes :

```
def supprimeMedicament(medicament : str , dico : dict )-> None :
    """supprime un médicament du stock, et affiche un message
    pour confirmer la suppression ou signaler que ce n'est pas
    possible """

def ajouteMedicament(medicament : str , dico : dict )-> None :
    """supprime un médicament du stock, et affiche un message
    pour confirmer la suppression ou signaler que ce n'est pas
    possible """
```

Exemple de vérification :

SHELL

```
>>> consulteStock("Pasfaim",NbreDeBoites)
Il y a 1 boite(s) de Pasfaim.
>>> supprimeMedicament('Pasfaim',NbreDeBoites)
Le médicament Pasfaim a été bien supprimé.
```

SHELL

```
>>> consulteStock("Pasfaim",NbreDeBoites)
Ce médicament ( Pasfaim ) n existe pas dans le stock.
>>> supprimeMedicament('Perlimpinpin',NbreDeBoites)
Le médicament Perlimpinpin ne peut pas être supprimé car
il n est pas référencé en stock !
```

SHELL

```
>>> consulteStock("PoudreZ",NbreDeBoites)
Ce médicament ( PoudreZ ) n existe pas dans le stock.
>>> ajouteMedicament("PoudreZ",NbreDeBoites)
Le médicament PoudreZ vient d être ajouté.
>>> consulteStock("PoudreZ",NbreDeBoites)
La référence existe mais il n y a plus de PoudreZ.
```

A vous de rajouter des boîtes de ce médicament qui est référencé.

7) Exécuter les commandes suivantes une à une en les copiant collant dans l'interpréteur ou en utilisant le débogeur (Shift + F5). Vous pouvez ainsi créer et gérer un nouveau stock.

```
# création d'un nouveau stock
NouveauStock : dict = { }

# on vérifie si le médicament existe
consulteStock("Alpha",NouveauStock)

# on l'ajoute
ajouteMedicament("Alpha",NouveauStock)

# on vérifie
consulteStock("Alpha",NouveauStock)

# on le supprime car on a fait une erreur
supprimeMedicament("Alpha",NouveauStock)

# on vérifie
consulteStock("Alpha",NouveauStock)

# on ajoute le bon médicament
ajouteMedicament("Béta",NouveauStock)

# on augmente le nombre de boite :
ajouteBoite( "Béta"   , 10 , NouveauStock)

# on vérifie
consulteStock("Béta",NouveauStock)
```

Exercice 5 : Manipulation d'ensemble.

Dans le cours pour les ensembles vous avez vu les opérations suivantes.

	Union	Intersection	Différence	Cardinal
Codage mathématique	$A \cup B$	$A \cap B$	$A \setminus B$	$card(A)$
Codage python	A | B	A & B	A - B	len(A)

1) Soit les ensembles : $A = \{'\text{Luc}','\text{Paul}','\text{Théo}','\text{Léa}','\text{Zoé}'\}$, et $B = \{'\text{Luca}','\text{Léa}','\text{Paul}'\}$. Déterminer sur PAPIER, l'union, l'intersection, la différence de A et B. Déterminer le cardinal de A puis de B.

2) Ecrire un programme qui effectue ce traitement sur A et B.

3) Ecrire la fonction `Intersection(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len().

4) Ecrire la fonction `Union(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len().

5) Ecrire la fonction `Différence(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len().

6) Ecrire la fonction `cardinal(A : set )-> int` sans utiliser le codage python & ou | ou - ou len().

Exercice 6 :

Soit l'ensemble suivant : $E = \{1, 2, 3, 8, 12, 15, 16, 19\}$

1) Écrire un programme pour obtenir le sous-ensemble M inclus dans E des éléments inférieurs à 10.

2) Compléter le programme pour obtenir l'ensemble N qui est le complémentaire de M dans E.

3) Compléter le programme pour vérifier que l'union de M et de N est égale à E. Vérifier que l'intersection de M et de N est l'ensemble vide.

4) Compléter le programme pour obtenir le sous-ensemble P inclus dans E des éléments pairs. Le programme doit obtenir aussi le sous-ensemble J qui est le complémentaire de P dans E.

5) Vérifier en complétant le programme que l'intersection des deux sous-ensembles complémentaires (N et J) est égale au complémentaire de l'union des sous-ensembles M et P (loi de De Morgan (écrite dans le CM1)).

Exercice 7 : Création d'ensemble et dictionnaire.

Le but de cet exercice est d'étudier un tirage aléatoire de nombres entiers. Vous commencerez par écrire une fonction qui effectue un grand nombre de tirages aléatoires de nombres entiers compris entre 1 et 10. La première fonction retourne les tirages obtenus dans une variable de type ensemble (set), la deuxième dans une variable de type dictionnaire (dict) et enfin les troisièmes et quatrièmes fonctions analysent les résultats.

script

```
1 from random import randint  # importation de module
2 valeur : int =randint(1,10)  # tirage d'un nombre pseudo aléatoire entre 1 et 10
```

1) Avant de coder la fonction pouvez vous estimer le contenu de l'ensemble obtenu en fonction de nbTirages ? Pourquoi ?

2) Ecrire la fonction `RemplitEnsemble(nbTirages : int ) ->set[int] :` qui effectue un grand nombre de tirages aléatoires de nombres compris entre 1 et 10. La fonction renvoie les résultats dans une variable de type ensemble (set).

3) Vérifier votre estimation de la question 1 en éxécutant votre fonction pour différentes valeurs de tirages (10, 100, 1000 , 10 000 par exemple).

4) Ecrire la fonction `Effectifs(nbTirages : int = 1000 ) ->dict[int] :` qui effectue toujours un grand nombre de tirages aléatoires de nombres compris entre 1 et 10. La fonction renvoie les résultats dans une variable de type dictionnaire (dict). La clé étant le résultat du tirage, et la valeur étant le nombre d'apparitions du tirage (ou l'effectif).

5) Définir la signature puis écrire la fonction `CalcFreqPourcent` qui renvoie pour chaque tirage (toujours un nombre entier compris entre 1 et 10) , la fréquence d'apparition en pourcentage. Vous choisirez judicieusement la structure de données. Peut on tester la fonction avec la commande *assert* ?

6) Définir la signature puis écrire la fonction `CalcMoyenneEtendue` qui calcule la moyenne des fréquences d'apparition et l'étendue des fréquences (l'écart entre la plus grande valeur et la plus petite).

Ex : si le 1 représente le plus fréquent des tirages avec 12% et le 3 le moins fréquent avec 9% alors l'étendue est $E = 12 - 9 = 3\%$

7) Auriez vous pu prévoir le résultat pour nbTirages=10 nbTirages=100 nbTirages=1000 nbTirages=10000 ?