Exercice 1 :

```python
def   somme (n : int ) -> int :
    """Renvoie la somme 1+2+...+n"""
    s : int =0
    for k in range(1,n+1):
        s =s + k
    return s

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( f"La somme 1+2+...+{n} = {s}")
```

Exercice 2 : Au début la somme est d'un montant S_0. Au bout d'un an la somme est augmentée de $p\%$. Ainsi $S_1 = S_0 + S_0 \times \frac{p}{100}$

De même au bout de 2 ans, on a $S_2 = S_1 + S_1 \times \frac{p}{100}$

Application numérique :

$$S_0 = 1000$$

$$S_1 = 1000 + \frac{2}{100} \times 1000 = 1020$$

$$S_2 = 1020 + 0.02 \times 1020 \approx 1040.40$$

$$S_3 = 1040.40 + 0.02 \times 1040.40 \approx 1061.21$$

$$S_4 = 1061.21 + 0.02 \times 1061.21 \approx 1082.43$$

$$S_5 = 1082.43 + 0.02 \times 1082.43.21 \approx 1104.08$$

La modélisation numérique ou codage est la suivante :

code.py

```python
def calculSomme(somme : float , taux : float , duree : int )-> float:

    for k in range(duree):
        somme = somme + somme * taux/100
    return somme

sommeTest=calculSomme(1000 , 2, 5 )
assert(1104.07<sommeTest  and sommeTest < 1104.09)



duree=5
taux = 2 # en pourcent
sommeInitial = 1000

sommeFinale = calculSomme(sommeInitial,taux,duree)
print(f"La somme de {sommeInitial} euros devient"
        f" au bout de {duree} ans "
        f": {sommeFinale:.2f} euros")
```

Exercice 3 : Le code suivant :

```
for i in range (2,7,1) :
    for j in range (2,5,1) :
        print( f"{j:2} x{i:2}={i*j:2}", end="  ")
    print()
print('xxxxxxxxxxxxxx')
for i in range (2,5,1) :
    for j in range (2,7,1) :
        print( f"{j:2} x{i:2}={i*j:2}", end="  ")
    print()
```

Affiche :

SHELL

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

Exercice 4 :

1) Copier et coller l'énoncé tel quel dans l'éditeur Thonny

2) afficheStock()

```
def afficheStock(dico : dict)-> None :
    """Affiche le dictionnaire"""
    for clé in dico :
        print(f"Il y a {dico[clé]} boite(s) de {clé}.")
```

Pour les tests, cette fonction est une procédure qui renvoie None, donc pour la tester, faire des appels et vérifier dans l'interpréteur que l'affichage est conforme au contenu du dictionnaire.

3) listeBoitesArenouveller() et afficheListe()

```
def listeBoitesArenouveller(dico : dict , seuil : int ) -> list[str] :
    """Cette fonction renvoie la liste de médicament du dictionnaire
    dont le nombre de boites est inférieur ou égal au seuil"""
    listeSousSeuil : list[str] =[]
    for clé in dico :
        if dico[clé]<=seuil :
            listeSousSeuil=listeSousSeuil+[clé]
    return listeSousSeuil

listeVerif=listeBoitesArenouveller(NbreDeBoites,1)
assert("Dofolgon" in listeVerif   )
assert("Profdinfo" in listeVerif   )
assert('Saleuvopa' in listeVerif )

def afficheListe (message :str , Liste) -> None :
    """Affiche le contenu de la liste précédé du message"""
    print( f" {message}  " , end="")
    for medoc in Liste[0:len(Liste)-1] :
        print( f" {medoc }", end=", ")
    print(f" {Liste[len(Liste)-1]}.")
```

4) consulteStock()

```python
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 ! """
    if medicament in dico :
        if dico[medicament]>0 :
            print(f"Il y a {dico[medicament]} boite(s) de {medicament}.")
        else :
            print(f"La référence existe mais il n'y a plus de {medicament}.")
    else :
        print(f"Ce médicament ( {medicament} ) n'existe pas dans le stock.")
```

Pas de test avec assert, c'est une procédure. Vous pouvez faire des appels et comparer le résultat avec le contenu du dictionnaire. Le code sur la page suivante est une variante plus longue (qui n'utilise pas le test " if medicament in dico").

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

    # On parcourt le dictionnaire pour chercher le médicament
    for cle in dico:
        if cle == medicament:
            trouve = True
            quantite :int  = dico[cle]  # On récupère directement la quantité
            if quantite > 0:
                print(f"Il y a {quantite} boîte(s) de {medicament}.")
                return # on sort de la procédure les lignes suivantes ne
                       # seront pas éxécutées.
            else:
                print(f"La référence existe mais il n'y a plus de {medicament}.")
                return # on sort de la procédure
    # Si on sort de la boucle c'est que le médicament cherché n'est pas dans
    #le stock
    print(f"Ce médicament ({medicament}) n'existe pas dans le stock.")
```

Ce code est plus long...

5) ajouteBoite() retireBoite()

```python
def ajouteBoite( medicament : str , nb :int, dico : dict) -> None :
    """Augmente le stock pour un medicament donné"""
    dico[medicament]=dico[medicament]+nb


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"""
    if nb<= dico[medicament] :
        dico[medicament]=dico[medicament]-nb
    else :
        print("Attention il n'y a pas assez de boites. "
              "Pas de modifications du stock.")
```

TESTS POSSIBLES

```python
##
## test avec assert
##
n_avant : int = NbreDeBoites["Kardabof"]
retireBoite   ("Kardabof",10,NbreDeBoites)
n_apres : int = NbreDeBoites["Kardabof"]
assert(n_apres==n_avant-10)
#
n_avant : int = NbreDeBoites["Kardabof"]
retireBoite   ("Kardabof",1000,NbreDeBoites)
n_apres : int = NbreDeBoites["Kardabof"]
assert(n_apres==n_avant)
##
n_avant : int = NbreDeBoites["Kardabof"]
ajouteBoite   ("Kardabof",10,NbreDeBoites)
n_apres : int = NbreDeBoites["Kardabof"]
assert( n_apres==n_avant+10)
```

Faites des appels divers et variés dans l'interpéteur pour prouver et être convaincu(e) de la validité de votre procédure.

6) supprimeMedicament() ajouteMedicament()

```
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 """
    if medicament not in dico :
        print(f"Le médicament {medicament} ne peut pas être supprimé"
              f"car il n'est pas référencé en stock !")
    else :
        dico.pop(medicament)
        print(f"Le médicament {medicament} a été bien supprimé.")

supprimeMedicament("IgrecPrim",NbreDeBoites)
assert("IgrecPrim"  not in NbreDeBoites)

def ajouteMedicament(medicament : str , dico : dict )-> None :
    """ajoute un médicament au stock, et affiche un message
    pour confirmer la création ou signaler que le médicament existe déja """
    if medicament  in dico :
        print(f"Le médicament {medicament} est déja en stock.")
    else :
        dico[medicament]=0

ajouteMedicament("IgrecPrim",NbreDeBoites)
assert("IgrecPrim"   in NbreDeBoites)
```

Remarque : vous pouvez aussi, voir le fonctionnement du code en faisant des appels des fonctions dans l'interpréteur comme proposé dans l'énoncé. A vous de faire preuve d'initiative.

Ne pas tester une fonction procédure que vous utiliserez plus tard, peut sembler être une perte de temps. En fait plus tard en ré-utilisant ce code, une recherche d'anomalie ou bug sera plus longue si elle est causée par des fonctions ou procédures "qui semblaient marcher correctement". Donc pensez à bien valider votre code au moment de son écriture, à laisser les traces de vos tests soit en commentaire, soit avec une série d'assert judicieux.Cela vous simplifiera l'écriture de la suite du code et vous épargnera du temps de recherche d'anomalie.

Exercice 5 : Manipulation d'ensembles.

1) On a les ensembles : $A = \{'\text{Luc}','\text{Paul}','\text{Théo}','\text{Léa}','\text{Zoé}'\}$, et $B = \{'\text{Luca}','\text{Léa}','\text{Paul}'\}$.

- L'union, est $A \cup B = \{'\text{Luc}','\text{Paul}','\text{Théo}','\text{Léa}','\text{Zoé}','\text{Luca}'\}$
- L'intersection est $A \cap B \{'\text{Paul}','\text{Léa}',\}$
- La différence de A et B est $A \setminus B = \{'\text{Théo}','\text{Zoé}','\text{Luc}'\}$
- Les cardinaux de A puis de B sont : card(A)= 5 et card(B)=3

2) Le programme qui effectue ce traitement sur A et B avec les notations python est :

script

```python
A : set = {'Luc','Paul','Théo','Léa','Zoé'}

B : set = { 'Luca','Paul', 'Léa' }
# exemple de code permettant de visualiser les résultats.
print(f"L'union de A et B est   U = {A | B} ")
print(f"L'intersection  de A et B est  I = {A & B} ")
print(f"La différence de A par B est A - B = {A-B}")
print(f"Card(A)= {len(A)} et Card(B)= {len(B)}")
```

3) La fonction `Intersection(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len() peut s'écrire :

script

```python
def intesection ( A : set , B : set) -> set :
    I : set =set()
    for x in A :
        if x in B :
            I.add(x)
    return I

assert( intesection(A,B) == {'Léa', 'Paul'} )
```

4) La fonction `Union(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len(). peut s'écrire :

script

```
def union ( A : set , B : set) -> set :
    U : set =set()
    for x in A :
        U.add(x)
    for x in B :
        U.add(x)
    return U
```

5) La fonction `Différence(A : set ,B : set)-> set` sans utiliser le codage python & ou | ou - ou len() peut s'écrire :

script

```
def difference ( A : set , B : set) -> set :
    D : set =set()
    for x in A :
        if x not in B :
          D.add(x)
    return D

assert( difference(A,B) == {'Théo', 'Zoé' , 'Luc'} )
```

6) La fonction `cardinal(A : set )-> int` sans utiliser le codage python & ou | ou - ou len() peut s'écrire :

script

```
def cardinal(A : set ) -> int :
    card : int = 0
    for x in A :
        card = card + 1
    return card

assert(cardinal(A)==5)
assert(cardinal(B)==3)
```

Exercice 6 :

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

1) Le programme permettant d'obtenir le sous-ensemble M inclus dans E des éléments inférieurs à 10 peut s'écrire :

code

```
E : set = {1,2,3,8,12,15,16,19} # définition de E

M : set =set()
for x in E :
    if x < 10 :
        M.add(x)
print(f"M={M} est l'ensemble des "
      f"éléments inférieurs à 10  de E ={E}.")
```

```
>>>
M= {8, 1, 2, 3} est l'énsemble des éléments inférieurs à 10 de E = {1, 2, 3, 8, 12, 15, 16, 19}.
```

2) Le programme permettant d'obtenir l'ensemble N qui est le complémentaire de M dans E peut s'écrire :

```
N : set =E-M
print()
print(f"N = {N} , est l'ensemble complémentaire"
      f" de M={M} dans l'ensemble E={E}." )
```

```
>>>
N = {16, 19, 12, 15} , est l'ensemble complémentaire de M = {8, 1, 2, 3}
dans l'ensemble E = {1, 2, 3, 8, 12, 15, 16, 19}.
```

3) Le complément de programme pour vérifier que l'union de M et de N est égale à E. Peut s'écrire :

```
if  ( M | N ) == E :
    print()
    print( f"L'union de M={M} et "
           f"de N ={N} est bien E={E}.")
```

```
>>>
L'union de M = {8, 1, 2, 3} et de N = {16, 19, 12, 15} est bien E = {1, 2, 3, 8, 12, 15, 16, 19}.
```

Pour vérifier que l'intersection de M et de N est l'ensemble vide on peut rajouter :

```
if (M&N) == set() :
    print()
    print(f"L'intersection de M={M} et de N ={N} "
          f"est bien vide car M&N renvoie {M&N}.")
```

```
>>>
L'intersection de M = {8, 1, 2, 3} et de N = {16, 19, 12, 15} est bien vide car M&N renvoie set().
```

4) Le complément de programme pour obtenir le sous-ensemble P inclus dans E des éléments pairs peut s'écrire :

```

P : set =set()
for x in E:
    if x%2==0 :
        P.add(x)
print(f"P = {P} , est l'ensemble des éléments"
      f" pairs de l'ensemble E={E}." )
```

```
>>>
P = {8, 16, 2, 12}, est l'ensemble des éléments pairs de l'ensemble E={1, 2, 3, 8, 12, 15, 16, 19}.
```

Pour obtenir aussi le sous-ensemble J qui est le complémentaire de P dans E on peut rajouter :

```
I : set = E-P
print(f"J = {J} est le complémentaire"
      f" de P={P} dans E={E}.")
```

```
>>>
J = {19, 1, 3, 15} est le complémentaire de P={8, 16, 2, 12} dans E={1, 2, 3, 8, 12, 15, 16, 19}.
```

5) Il est possible de compléter le programme pour vérifier 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).

```
if N&J   ==   E -( M | P ) :
    print(f"L'intersection de N et de J est N & J ={N & J}")
    print(f"L'union de M et P est  M | P = {M | P } ")
    print(f"Le complémentaire dans E de "
          f"l'union de M et P est   "
          f"E -( M | P ) = {E -( M | P )}")
    print(f"On voit que N & J est identique à E -( M | P ).")
```

```
>>>
L'intersection de N et de J est N & J ={19, 15}
L'union de M et P est M | P = {1, 2, 3, 8, 12, 16}
Le complémentaire dans E de l'union de M et P est E -( M | P ) = {19, 15}
On voit que l'ensemble correspondant à l'intersection de N et J et se codant N & J est identique à E -( M | P ).
```

Exercice 7 :

1) Si il y a un seul tirage, l'estimation du contenu sera vraie avec une probabilité de $\frac{1}{10}$.

Si l'on augmente le nombre de tirage, il est quasiment certain d'obtenir toutes les issues , à savoir un ensemble $E = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Il faut essayer avec N=100,500,1 000,10 000.

2) La fonction `RemplitEnsemble(nbTirages : int ) ->set[int] :` qui effectue un grand nombre de tirages aléatoires de nombres compris entre 1 et 10 peut s'écrire :

script

```
def RemplitEnsemble(nbTirages : int  ) ->set[int]:
    E=set()
    for k in range(nbTirages) :
        valeur =randint(1,10)
        E.add(valeur)
    return E

# réponse à la question 3
assert( RemplitEnsemble( nbTirages =1000 ) == {1,2,3,4,5,6,7,8,9,10})
assert( RemplitEnsemble( nbTirages =10000 ) == {1,2,3,4,5,6,7,8,9,10})
```

3) voir assert dans cadre ci-dessus.

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

Il est possible d'écrire une fonction intermédiaire avant pour avoir un code moins long.

script

```
1
2  def RemplitListeValeurs(nbTirages : int) -> list[int] :
3          """Cette fonction effectue nbTirages aléatoires d'entier entre 1
4           et 10 inclus. Les nbTirages sont renvoyées dans une liste"""
5      V : list[int]=nbTirages *[0]
6      for k in range(nbTirages) :
7          V[k] =randint(1,10)
8      return V
9
10
11 def  CalcEffectif( nbTirages : int ) ->dict[int:int] :
12     Effectif : dict[int:int]={}   # dictionnaire clé entière (le nombre issu du tirag
13                           # la valeur : le nb d'apparition (ou effectif)
14     for issue in range(1,11,1) :  # on remplit le dictionnaire avec toutes les sortie
15         Effectif [issue] = 0
16     ListeTirages = RemplitListeValeurs(nbTirages)
17     for k in range(len(ListeTirages)) : # parcours la liste et compte les valeurs.
18         cle=ListeTirages[k]
19         Effectif[cle]=Effectif[cle]+1
20     return Effectif
```

5) La fonction `CalcFreqPourcent` qui renvoie pour chaque tirage , la fréquence d'apparition en pourcentage peut s'écrire comme ci-dessous. Le test avec assert et un encadrement est dans le code :

Script

```
def  CalcFreqPourcent (Effectif : dict[int:int]) -> dict[int:float] :

    FreqPourcent : dict[int:float]={}

    effectifTotal : int =0
    for cle in Effectif :
        effectifTotal = effectifTotal + Effectif[cle]

    for cle in Effectif :
        FreqPourcent [cle] = Effectif [cle] / effectifTotal *100

    return FreqPourcent

Dtest= CalcEffectif(1000)
Resu = CalcFreqPourcent(Dtest)
assert(len(Resu)==10)     # on vérifie qu'il y a 10 issues analysées
somme : int = 0
for cle in Resu :
    somme = somme + Resu[cle]
print(somme)
# on travaille sur des flottants donc leur somme ne fait pas 100
 mais presque 100 !!!
#donc on peut définir un intervalle à +/- 0.0001 par exemple.
assert(100-10**-4 <= somme  and   somme<=100+10**-4)
```

6) Ecrivons 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). Avec par exemple : si le 1 représente le plus fréquent des tirages avec 12% et le 3 le moins fréquent avec 9% alors l'étendue est $E = 12 - 9 = 3\%$.

Cette fonction peut s'écrire :

Script

```
def   CalcMoyenneEtendue  (DicoTiragePourcent : dict[int:float]) -> tuple[ float , fl
] :

    somme : int =0
    # calcul de la moyenne ( a priori 10 % !!!)
    for cle in DicoTiragePourcent :
        somme = somme + DicoTiragePourcent[cle]
    moyenne : float = somme /10

    mini : float = DicoTiragePourcent[1]
    maxi : float = DicoTiragePourcent[1]


    for cle in DicoTiragePourcent :
        if DicoTiragePourcent[cle] < mini :
            mini=DicoTiragePourcent[cle]
        if DicoTiragePourcent[cle] > maxi :
            maxi=DicoTiragePourcent[cle]
    Etendue =maxi-mini

    return moyenne ,Etendue

assert(CalcMoyenneEtendue({1:1,2:1, 3:1 , 4:1 , 5:1 , 6:1 , 7:1, 8:1, 9:1 , 10:1} )
                          ==(1,0))
assert(CalcMoyenneEtendue({1:10, 2:1, 3:1, 4:1, 5:1, 6:1 , 7:1, 8:1, 9:1 , 10:1} )
                          ==(1.9,9))
```

7) La seule chose qu'il est possible de dire est que si l'on a un grand nombre de tirages aléatoires avec chaque issue équiprobable, alors la fréquence observée converge vers la probabilité. Donc il n'est pas possible de prévoir rigoureusement le résultat mais d'avoir simplement une tendance pour les grands nombres.