Corrections

Exercice 1

Objectif : *print* | *input* | *boucle*

1. Pour aider le fils de votre voisin qui entre en CE1, compléter le script ci-dessous.

```
def table(n):
    # à compléter

n = input("Saisir un entier compris entre 1 et 10")
n = ...
table(n)
```

La fonction table prend en paramètre un entier n compris entre 1 et 10 (compris) et affiche en console la table de multiplication de n.

➔ Ne pas oublier la docstring et d'éventuelles assertions

Exemple : si l'utilisateur saisit 6, on obtient :

```
>>>
Table de 6
6 * 0 = 0
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
```

2. Finalement, le petit voisin a besoin de toutes les tables de multiplication de 1 jusqu'à 10. Modifier le script.

3. Apprendre par cœur les tables de multiplication 😊

```
# Question 1, version 1
def table(n):
    """renvoie la table de n"""
    assert type(n)==int
    assert 1<=n<=10, "n doit-être compris entre 1 et 10"
    print("Table de", n)
    for i in range(11):
        print(n, "*", i, "=", n*i)

n = input("Saisir un entier compris entre 1 et 10")
n = int(n)
table(n)
```

```
# Question 1, version 2
def table(n):
    affichage = "Table de "+str(n)+"\n"
    for i in range(11):
        affichage = affichage + str(n)+" * "+str(i)+" = "+str(n*i)+"\n"
    return affichage

n = 6
print(table(n))
```

Remarque

"\n" permet un retour à la ligne

```
# Question 2
def table(n):
    print("Table de", n)
    for i in range(11):
        print(n, "*", i, "=", n*i)

for n in range(1,11):
    table(n)
    print()
```

Exercice 2

Objectif : *boucle | spécification, assertion | jeu de tests*

Écrire la fonction `nb_de_div_par_2` qui prend en paramètre un entier non nul et qui renvoie combien de fois de suite cet entier est divisible par 2.

➔ On créera la docstring, des assertions en précondition et postcondition et un jeu de tests.

Exemples

```
>>> nb_de_div_par_2(24)
3
>>> nb_de_div_par_2(5)
0
```

```
def nb_de_divisions_par_2(nb):
    """nb est un entier non nul
       Renvoie le nombre de fois où 'nb' est divisible pas 2
       Par exemple 24 : 24 est divisible 3 fois par 2 : 24->12->6->3
                        on renvoie 3
                        remarque : 24 = 3*(2**3)"""
    assert type(nb)==int   # précondition
    assert nb > 0          # précondition
    nombre = nb      # sauvegarde -> réutilisation en postcondition
    nb_div = 0
    reste = nb%2
    while reste==0:
        nb_div+=1
        nb = nb//2
        reste = nb%2
    assert nombre == nb*2**nb_div   # postcondition
    return nb_div

def jeu_de_tests():
    assert nb_de_divisions_par_2(24)==3
    assert nb_de_divisions_par_2(7)==0
                # exemple d'un cas où nb n'est pas divisible par 2
    assert nb_de_divisions_par_2(1)==0
    print("semble juste...")

nb = int(input("choisir un entier"))
print(nb_de_divisions_par_2(nb))

jeu_de_tests()
```

Exercice 3

Objectif : *chaîne de caractères | Recherche d'un élément*

Il s'agit d'écrire un programme qui demande de saisir une chaîne d'ADN valide et une séquence d'ADN valide (*valide* signifie qu'elles ne sont pas vides et sont formées exclusivement d'une combinaison arbitraire de "a", "t", "g" ou "c") et de dénombrer le nombre de séquences dans la chaîne.

▪ Écrire la fonction `valide` qui prend en paramètre une chaine de caractère et qui renvoie `True` si la saisie est valide, `False` sinon.

Exemples

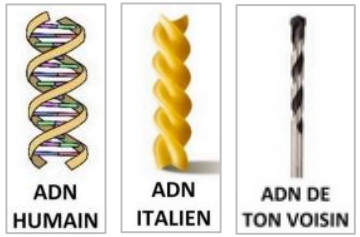
```
>>> valide("agtcc")
True
```

```
>>> valide("accbtg")
False
```

▪ Écrire la fonction `nombre` qui reçoit deux paramètres, la *chaîne* et la *séquence* et qui retourne le nombre de *séquence* dans la *chaîne* [ne pas oublier la documentation et les assertions].

Exemple :

```
>>> chaine = "attgcaatcgtggtacatgc"
>>> sequence = "ca"
>>> nombre(chaine, sequence)
2
```




```python
import doctest

def valide(chaine:str)->bool:
    """vérifie si la chaîne est valide
    c'est à dire qu'elle n'est pas vide
              et qu'elle ne contient que des caractères a, t, g et c
    renvoie True ou false"""
    ens_caracteres = ['a', 't', 'g', 'c']
    if chaine=="":
        return False
    else:
        for caractere in chaine:
            if caractere not in ens_caracteres:
                return False
        return True

def nombre(chaine:str, sequence:str)->int:
    """on cherche le nombre d'occurrences 'sequence' dans 'chaine'
    Exemple :
    >>> chaine = 'attgcaatcgtggtacatgc'
    >>> sequence = 'ca'
    >>> nombre(chaine, sequence)
    2
    """
    assert valide(chaine)
    assert valide(sequence)
    n = len(chaine)
    m = len(sequence)
    if m > n:
        return 0
    else:
        compteur = 0
        for i in range(n-m+1):
            if chaine[i:i+m]==sequence:
                compteur+=1
        return compteur

###### Programme principal

chaine = "attgcaatcgtggtacatgc"
sequence = "ca"

print("Il y a ", nombre(chaine, sequence), " séquence(s) '",\
        sequence, "' \ndans la chaîne '", chaine, "'", sep="")

doctest.testmod()
```

Exercice 4

Objectif : *listes | pop | append*

1. Préliminaire. On rappelle que :

- l'expression `T1 = list(T)` fait une copie de T indépendante de T,
- l'expression `x = T.pop()` enlève le sommet de la liste T et le place dans la variable x
- l'expression `T.append(v)` place la valeur v au sommet de la liste T.

Tester ⬇, éventuellement, en console les instructions suivantes :

Console

```
>>> T = [1, 2, 3]
>>> T2 = list(T)
>>> x = T2.pop()
>>> x
3
>>> T2.append(4)
>>> T2
[1, 2, 4]
>>> T
[1, 2, 3]
```

2. Compléter le code Python de la fonction positif ci-contre ➡

qui prend une liste T de nombres entiers en paramètre et qui renvoie la liste des entiers positifs dans le même ordre, sans modifier la variable T.

```
def positif(T):
    T2 = list(T)          # copie de T
    T3 = []               # liste vide
    while T2 != []:
        x = T2.pop()      # on retire le dernier élément de T2
        if x >= 0:
            T3.append(x) # on l'ajoute à T3 si x >= 0
    # on replace les éléments de T3 dans l'ordre dans T2
    while T3 != [] :      # T3 contient bien des éléments qu'il faut
                          # mais dans le sens inverse
        x = T3.pop()      # on retire le dernier élé de T3
        T2.append(x)      # pour le mettre dans T2

    print('T =', T)       # on vérifie que T reste inchangé
    return T2

print(positif([-1, 0, 5, -3, 4, -6, 10, 9, -8]))
```

Exemple

```
>>> positif([-1, 0, 5, -3, 4, -6, 10, 9, -8])
T = [-1, 0, 5, -3, 4, -6, 10, 9, -8]
[0, 5, 4, 10, 9]
```

Exercice 5

Objectif : *dictionnaires*

Partie 1

On considère le dictionnaire suivant :

```
mondict = {"device": "laptop" , "constructeur": "acer" , "ram": "8G" ,
           "processeur": "Intel core i5", "stockage": "1 T"}
```

Écrire les inscriptions permettant :

- de corriger l'erreur `"stockage": "2 T"`
- d'afficher ➔ la liste des clés,
  ➔ la liste des valeurs
  ➔ la liste des paires de clés et valeurs
- d'ajouter la pair clé-valeur : `"Système d'exploitation" : "Windows 11"`
- à l'aide d'une boucle, d'afficher l'ensemble des couples.

```
*** Console ***
>>>
processeur : Intel core i5
constructeur : acer
device : laptop
stockage : 2 T
Système d'exploitation : windows 11
ram : 8G
>>>
```

```
mondict = {"device": "laptop" , "constructeur": "acer" , "ram": "8G" ,
           "processeur": "Intel core i5", "stockage": "1 T"}

mondict["stockage"] = "2 T"
print(mondict)

print()

print("liste des clés : ")
print(mondict.keys())

print()

print("liste des valeurs : ")
print(mondict.values())

print()

print("listes des items")
print(mondict.items())
print("ou : ")
print(mondict)

print()

mondict["Système d'exploitation"] = "windows 11"
print(mondict)

print()

for i in mondict.keys():    # ou -> for i in mondict.keys():
    print(i, ":", mondict[i])
```

Partie 2

Écrire une fonction Python qui prend en paramètre un entier n et qui renvoie un dictionnaire formé des entiers de 1 à n et de leurs carrées.

Exemple

Pour n = 7 le dictionnaire sera : `{1: 1, 2: 4, 3: 9, 4: 16, 5: 25 , 6: 36 , 7: 49}`

```
# par compréhension
def dico_carre(n):
    return {i:i**2 for i in range(n+1)}

print(dico_carre(7))
```

```
# à l'aide d'une boucle
def dico_carre_v2(n):
    dico = {}   # initialisation
    for i in range(n+1):
        dico[i] = i**2
    return dico

print(dico_carre_v2(7))
```