Corrections

Exercice 1

Objectif -> print / input / boucle

1. Pour aider le fils de votre voisin qui est en CE1, écrire un script qui affiche la table de multiplication de n, où n est choisit par l'utilisateur.

Attention : n doit être compris entre 0 et 10 (compris)

Exemple : si on choisit n = 6, alors on obtient en console ➡

2. Finalement, le petit voisin a besoin de toutes les tables de multiplication de 0 jusqu'à 10.
Écrire une autre version du script.

3. Apprendre par cœur les table de multiplication 😊

Exemple pour n = 6 ➡

```
*** Console ***
>>>
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
>>>
```

```
# Question 1
n = -1          # on initiale n à -1 pour être
                # sûr de rentrer dans la boucle
while n < 0 or n > 11:
    n = int(input("choisir un entier compris entre 0 et 10"))

print("Table de", n)
for i in range(11):
    print(n, "*", i, "=", n*i)
```

On peut aussi proposer une version avec une fonction ⬇

```
# Question 1, version 2
def table(n):
    """renvoie la table de n"""
    assert type(n)==int
    assert 0<=n<=10, "n doit-être compris entre 0 et 10"
    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

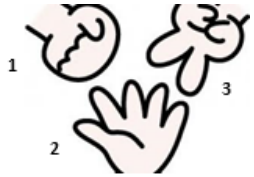
"\n" permet un retour à la ligne

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

Exercice 2

Objectif : *structure conditionnelle*

Réaliser le programme du jeu « pierre, feuille, ciseaux » :
le joueur joue contre l'ordinateur.
il peut faire 1, 2 ou 3 comme choix.




```
from random import *

def affichage(c):
    """Gestion de l'affichage du choix"""
    assert c in [1,2,3]
    if c==1:
        return "pierre"
    elif c==2:
        return "feuille"
    else:
        return "ciseaux"

def jeu(choix):
    """choix est un entier égal à 1 (pierre), 2 (feuille) ou 3 (ciseaux)
    """
    assert choix in [1,2,3]
    choix_ordi = randint(1,3)
    print("choix ordinateur :", affichage(choix_ordi))
    if choix==choix_ordi:
        return "match nul"
    elif (choix==1 and choix_ordi==3) \
         or (choix==2 and choix_ordi==1) \
         or (choix==3 and choix_ordi==2):
         return "joueur gagne"
    else:
        return "ordinateur gagne"

choix = int(input("choisir 1 (pierre), 2 (feuille) ou 3 (ciseaux)"))
print("votre choix :", affichage(choix))

print(jeu(choix))
```

Remarque : le caractère d'échappement \ permet d'écrire une instruction sur plusieurs lignes

Exercice 3

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

Écrire une fonction nb_de_div_par_2 qui prend en argument 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 4

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

Exemples

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

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

▪ Écrire une fonction `nombre` qui reçoit deux arguments, 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 5

Objectif : *liste*

Il s'agit d'écrire, d'une part, un programme principal et, d'autre part, deux fonctions utilisées dans le programme principal.

- La fonction `listAleaInt(n,a,b)` retourne une liste de n entiers aléatoires dans [a ; b] en utilisant la fonction `randint(a,b)` du module `random`.
- La fonction `mini_premierePosition(liste)` qui permet d'échanger le premier élément du tableau avec son minimum.

Prévoir un jeu de tests pour cette fonction 

- Dans le programme principal :
  construire une liste en appelant la fonction `listAleaInt()` ;
  puis l'appliquer à la fonction `mini_premierePosition(liste)`

```
from random import *

def listAleaInt(n, a, b):
    """retourne une liste de n entiers aléatoire
       tous compris dans l'intervalle [a, b]"""
    assert type(n)==int
    liste = []
    for i in range(n):
        liste.append(randint(a,b))
    return liste

def mini_premierePosition(liste):
    """échange le plus petit élément de la liste
       avec le premier élément"""
    rang_mini = 0
    mini = liste[0]
    n = len(liste)
    for i in range(1,n):
        if liste[i]<mini:
            rang_mini = i
            mini = liste[i]
    if rang_mini != 0:     # il reste à permuter les éléments
        liste[0], liste[rang_mini] = liste[rang_mini], liste[0]

def jeu_de_tests():
    l = [1,2,3,0,4,5]
    l2 = [0,2,3,1,4,5]
    mini_premierePosition(l)
    assert l==l2
    l = [0,1,2,3,4,5]
    l2 = [0,1,2,3,4,5]
    mini_premierePosition(l)
    assert l==l2
    print("fin")

### Programme principal

n = 10
a = 0
b = 100

jeu_de_tests()

L1 = listAleaInt(n,a,b)
print(L1)
mini_premierePosition(L1)
print(L1)
```