NSI *Première* -> Terminale

Le programme de terminale est très chargé avec de nombreuses notions à voir.

Nous vous proposons donc de reprendre et chercher quelques exercices de programmation en Python (fin août), afin d'être opérationnel dès la reprise.

- ➔ La correction de quelques exercices sera donnée à partir du lundi 24 août 2026 sur le site du lycée
- ➔ D'autres seront corrigés lors des premières séances de NSI
- ➔ **Les exercices 9 et 10** sont à rendre lors de la première séance de NSI

Ces notions seront évaluées, notamment à l'occasion d'une épreuve pratique ou questions flash dès la semaine de la rentrée.

Bonnes vacances à toutes et à tous !

Exercice 1

Objectif : *print | input | boucle | spécification, assertion*

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 en précondition.

➔ On utilisera des f-strings pour l'affichage en console.

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

```
>>> table()
Saisir un entier compris entre 1 et 10 : 6
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
```

Un clavier après quelques semaines de vacances

2. Apprendre par cœur les tables de multiplication 😊

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
```

En effet : 24 est divisible 3 fois par 2 : 24->12->6->3

```
>>> nb_de_div_par_2(5)
0
```

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 chaine et la séquence et qui retourne le nombre de séquence dans la chaine [ne pas oublier la documentation et les assertions].

Exemple :

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

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
...
>>> T2.append(4)
>>> T2
...
>>> T
...
```

2. Compléter le code Python de la fonction positif ci-dessous ⬇ 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 = ...(T)      # copie de T
    T3 = ...         # liste vide
    while T2 != []:
        x = ...      # on retire le dernier élément de T2
        if ... >= 0:
            T3.append(...)
    # on replace les éléments de T3 dans l'ordre dans T2
    while T3 != ... :
        x = T3.pop()
        ...
    print('T =', T) # on vérifie que T est inchangée
    return T2
```

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
>>>
```

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}`

Exercice 6

Objectif : *écriture binaire / décimale d'un entier*

1. On modélise la représentation binaire d'un entier non signé par un tableau d'entiers dont les éléments sont 0 ou 1.

Par exemple, le tableau `[1, 0, 1, 0, 0, 1, 1]` représente l'écriture binaire de l'entier 1010011_2 dont l'écriture décimale est `2**6 + 2**4 + 2**1 + 2**0 = 83`.

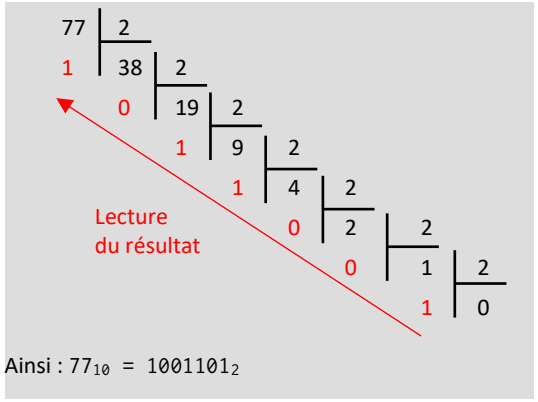
À l'aide d'un parcours séquentiel, écrire la fonction `convertir` répondant aux spécifications suivantes ⬇

```
def convertir(tab):
    """tab est un tableau d'entiers, dont les éléments sont 0 ou 1 et
    représentant un entier écrit en binaire.
    Renvoie l'écriture décimale de l'entier positif dont la représentation
    binaire est donnée par le tableau tab"""
    # à compléter
```

Exemple

```
>>> convertir([1, 0, 1, 0, 0, 1, 1])
83
>>> convertir([1, 0, 0, 0, 0, 0, 1, 0])
130
```

2. Pour rappel, la conversion d'un nombre entier positif en binaire peut s'effectuer à l'aide des divisions successives comme illustré ici ⬇



Ainsi : $77_{10} = 1001101_2$

```
def binaire(a):
    bin_a = [...]
    a = a//2
    while a ... :
        bin_a = ... + ...
        a = ...
    return bin_a
```

Voici une fonction python basée sur la méthode des divisions successives ⬆ permettant de convertir un nombre entier positif en binaire.

Compléter la fonction `binaire`.

Exemple

```
>>> binaire(77)
[1, 0, 0, 1, 1, 0, 1]
```

Exercice 7 ***Problème du rendu de monnaie***

Objectif : *algorithme glouton*

La fonction `rendu_monnaie` prend en paramètres deux nombres entiers positifs `s_due` et `s_versee`. Elle procède au rendu de monnaie de la différence `s_versee - s_due` pour des achats effectués avec le système de pièces de la zone Euro. On utilise pour cela un algorithme glouton qui commence par rendre le maximum de pièces de plus grandes valeurs et ainsi de suite. Par la suite, on assimilera les billets à des pièces.

La fonction `rendu_monnaie` renvoie un tableau de type `list` contenant les pièces qui composent le rendu.

Toutes les sommes sont exprimées en euros. Les valeurs possibles pour les pièces sont donc contenues dans le tableau `pieces = [1, 2, 5, 10, 20, 50, 100, 200]`

Ainsi, l'instruction `rendu_monnaie(452, 500)` renverra la liste `[20, 20, 5, 2, 1]`.

En effet, la somme à rendre est de 48 centimes soit 20 + 20 + 5 + 2 + 1.

Le code de la fonction `rendu_monnaie` est donné ci-dessous :

```
def rendu_monnaie(s_due, s_versee):
    pieces = [1, 2, 5, 10, 20, 50, 100, 200]
    rendu = ...
    a_rendre = ...
    i = len(pieces) - 1
    while ... :
        if pieces[i] <= a_rendre :
            rendu.append(...)
            a_rendre = ...
        else :
            i = ...
    return rendu
```

Compléter ce code et le tester

```
>>> rendu_monnaie_centimes(700,700)
[]
>>> rendu_monnaie_centimes(102,500)
[200, 100, 50, 20, 20, 5, 2, 1]
```

Exercice 8
Objectif : *implémenter en Python un algorithme*

Connaître les quelques algorithmes du cours de première et une implémentation en Python :

- Déterminer le minimum / maximum d'une liste
- Tri par sélection, tri par insertion
- Rechercher un élément dans une liste par dichotomie

Exercices 9 et 10 à rendre

Exercice 9
Objectif : *dictionnaire*

Écrire une fonction `enumere(tab)` qui prend en paramètre un tableau `tab` (type `list`) et renvoie un dictionnaire `d` dont les clés sont les éléments de `tab` avec pour valeur associée la liste des indices de l'élément dans le tableau `tab`.

-> prévoir une ou des assertions en préconditions

-> écrire une fonction `tests_enumere()` réalisant plusieurs tests sous forme d'assertions

Exemples

```
>>> enumere([])
{}
>>> enumere([1, 2, 3])
{1: [0], 2: [1], 3: [2]}
>>> enumere([1, 1, 2, 3, 2, 1])
{1: [0, 1, 5], 2: [2, 4], 3: [3]}
```

Exercice 10
Objectif : *Tri à bulle*

Le principe :

- L'algorithme parcourt la liste et compare les éléments consécutifs. Lorsque deux éléments consécutifs ne sont pas dans l'ordre, ils sont échangés.
- Après un premier parcours complet de la liste, le plus grand élément est forcément en fin de liste, à sa position définitive.
- Le reste du tableau est en revanche encore en désordre. Il faut donc le parcourir à nouveau, en s'arrêtant à l'avant-dernier élément. Après ce deuxième parcours, les deux plus grands éléments sont à leur position définitive.
- Il faut donc répéter les parcours du tableau, jusqu'à ce que les deux plus petits éléments soient placés à leur position définitive (*pas d'échange sur un parcours*).

Exemple

L Init :	5	3	1	6	2	7	4	
Parcourt 1	3	5	1	6	2	7	4	
	3	1	5	6	2	7	4	
	3	1	5	6	2	7	4	
	3	1	5	2	6	7	4	
	3	1	5	2	6	7	4	
Fin prc 1	3	1	5	2	6	4	7	
Fin prc 2	1	3	2	5	4	6	7	
Fin prc 3	1	2	3	4	5	6	7	etc.

1. Effectuer à la main un tri à bulles du tableau suivant : L_1 = [2, 7, 4, 1, 8, 5] et L_2 = [2, 4, 3, 1]

2. Écrire une fonction prenant pour argument la liste à trier et qui renvoie la liste triée par la méthode du tri à bulles.

3. Quelle est le coût de cette méthode dans le pire des cas ?