Première NSI (A, B, &	Épreuve blanche d'informatique nº 1	novembre 2023
C)		
	Durée : 3h30	

Documents et calculatrice non autorisée

Exercice 1

- 1. Écrire les instructions permettant d'afficher les séquences suivantes, sous la forme 4 p. de votre choix (affichage à l'écran, liste, tuple, ...)
 - a) La liste des 25 premiers multiples de 3.
 - b) Une suite de 50 éléments alternés 1,2
 - c) Les entiers relatifs de -12 à 6.
- 2. Écrire une fonction qui renvoie le minimum d'une liste ou d'un tuple passé en argument. On n'utilisera pas la fonction native min.

Fournir des exemples d'utilisation de la fonction.

- 3. Écrire une fonction qui renvoie la position du maximum d'une liste ou d'un tuple.
 - Fournir des exemples d'utilisation de la fonction.
- 4. Écrire une fonction qui renvoie la moyenne d'une liste ou d'un tuple d'éléments numérique.

Fournir des exemples d'utilisation de la fonction.

Exercice 2

- 1. Donner, en écrivant les calculs,
 - a) l'écriture binaire de 181.
 - b) l'écriture décimale du nombre (10011111)₂
- 2. Écrire, en langage algorithmique, en pseudo code ou en Python, l'algorithme permettant de transformer un nombre écrit en base 10 en sa représentation en base 2.

Exercice 3

1. On donne le code HTML suivant :

3 p.

```
<html>
       <titre>Une page à corriger</titre>
3
       <meta charset="utf-8" />
4
     </head>
5
     <body>
       <titre>Chapitre à corriger</titre>
       Dans cette page, il faut corriger certaines balises qui sont
8
       <span style="text-color:red ;">incorrectes<span> :
       balises
10
       <l
11
       <a text="La Madone">https://www.lamadone.fr/</a>
12
     </body>
```

Corriger ce code HTML en indiquant sur votre copie le numéro de la ligne et les corrections apportées.

2. Expliquer la différence entre HTML et CSS.

Exercice 4

L'objectif de ce problème est de mettre en place une modélisation d'un jeu de labyrinthe avec le langage Python.

10 p.

On décide de représenter un labyrinthe par un tableau carré de taille n = 10, dans le quel les cases contiendront 0 si on peut s'y déplacer et 1 s'il s'agit d'un mur. Voici un exemple de représentation d'un labyrinthe.



- 1. Proposer un dessin de labyrinthe simple de taille n = 6 et sa représentation associée sous forme de liste de liste.
- 2. Compléter la fonction mur qui prend comme paramètres un tableau représentant un labyrinthe et deux entiers i et j compris entre 0 et n-1 et qui renvoie un booléen indiquant la présence ou non d'un mur.

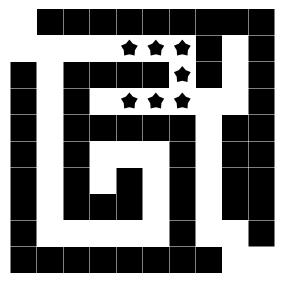
Exemple:

```
>>> mur(laby, 2, 3)
True
>>> mur(laby, 1, 8)
False

def mur(laby, i, j)
    if laby[i][j] ...:
        return ...

else:
        return ...
```

Un parcours de labyrinthe va être représenté par une liste de cases. Il s'agit de couples (i,j) où i et j correspondent respectivement aux numéros de ligne et colonne des cases successivements visitées au long du parcours. Ainsi, la liste suivante [(1,4), (1,5), (1,6), (2, 6), (3,6), (3,5), (3,4)] correspond au parcours repéré par des étoiles.



La liste [(0,0), (1,0), (1,1), (5,1), (6,1)] ne peut correspondre au parcours d'un labyrinthe car toutes les cases successives ne sont pas adjacentes.

- 3. a) Donner une liste qui permet de parcourir tout le labyrinthe.
 - b) Expliquer pourquoi la liste [(0,0), (1,0), (1,1), (2,1), (3,1), (3,2)] n'est pas valide.
- 4. On considère la fonction voisine ci dessous, écrite en langage Python, qui prend en paramètre deux cases données sous forme de couple.

```
def voisine(case1, case2):
    ligne1 = case1[0]
    col1 = case1[0]
    ligne2 = case2[0]
    col2 = case2[0]
    d = (ligne1 - ligne2)**2 + (col1 - col2)**2
    return (d == 1)
```

- a) Quel est le résultat d'une expression comme d == 1? En déduire ce que renvoie la fonction voisine.
- b) En détaillant, avec un tableau de variables, donner le résultat de la fonction voisine dans les deux cas suivant :

```
>>> voisine((1, 1), (1, 0))
...
>>> voisine((1, 1), (3, 1))
```

c) Compléter la fonction adjacentes qui reçoit une liste de case et renvoie un booléen indiquant si la liste des cases forme une chaîne de cases adjacentes.

```
def adjacentes(liste_cases):
    for i in ... ... ... :
        if voisine(liste_cases[i], liste_cases[i + 1]) == ...:
            return ...
    return ...
```