ISN – Informatique et Sciences du Numérique

TP3 PYTHON : TESTS – STRUCTURES ALTERNATIVES

1 – LOGIQUE BOOLEENNE

*En informatique, tous les mécanismes de tests sont basés sur la **logique booléenne** : Une expression est soit **vraie** (**True**) soit **fausse** (**False**). Les opérateurs de comparaison permettent de comparer deux valeurs et d'indiquer si la condition énoncée est vérifiée ou non (a est plus grand que b, a et b ont la même valeur, etc…). **True** correspond au niveau logique **1** et **False** au niveau logique **0**.*

Il existe 6 opérateurs de comparaisons :

>	*Strictement supérieur à*
>=	*Supérieur ou égal à*
<	*Strictement supérieur à*
<=	*Supérieur ou égal à*
==	*Egal à*
!=	*Différent de*

Exercice 1

Taper dans l'interpréteur les lignes ci-dessous. **Vérifier** et **justifier** que les résultats des tests correspondent à ceux attendus.

```
>>>a = 1
>>>b = 2
>>>a < b
>>>a > b
>>>a == 5
>>>a != b
>>>0 == False
```

*Pour obtenir des tests plus complexes il est possible de combiner plusieurs tests à l'aide des trois **opérateurs booléens** : **and** (ET), **or** (OU), **not** (NON).*

Opérateur and

A	B	A and B
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Opérateur or

A	B	A or B
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Opérateur not

A	not A
0	**1**
1	**0**

Exercice 2

Taper dans l'interpréteur les lignes ci-dessous. **Vérifier** et **justifier** que les résultats des tests correspondent à ceux attendus.

```
>>>a = 1
>>>b = 2
>>>a == 1 and b < 3
>>>a == 1 and b == 3
>>>a == 1 or b > 3
>>>a < 1 or b > 3
```

2 – STRUCTURES ALTERNATIVES

*La **structure alternative** « **if** » permet d'exécuter un bloc d'instruction si une condition est vraie, sinon le bloc d'instruction n'est pas exécuté.*

Syntaxe

```
if Condition :
    Bloc d'Instructions
Suite du programme
```

Exercice 3

1. **Taper** dans l'éditeur de scripts les lignes suivantes :

```
a = 1
if a == 1 :
    print("La valeur est bien 1")
```

2. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.

3. **Modifier** le script de la manière suivante :

```
a = 3
if a == 1 :
    print("La valeur est bien 1")
```

4. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.

*La **structure alternative** « **if - else** » permet d'exécuter un bloc d'instruction si une condition est vraie, sinon un autre bloc d'instruction est exécuté.*

Syntaxe

```
if Condition :
    Bloc d'Instructions 1
else :
    Bloc d'Instructions 2
Suite du programme
```

Exercice 4

1. **Taper** dans l'éditeur de scripts les lignes suivantes :

```
a = 1
if a == 1 :
    print("La valeur est bien 1")
else :
    print("La valeur est différente de 1")
```

2. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.

3. **Modifier** le script de la manière suivante :

```
a = 3
if a == 1 :
    print("La valeur est bien 1")
else :
    print("La valeur est différente de 1")
```

4. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.

*Dans le cas de **structures alternatives imbriquées**, il est possible d'utiliser une instruction **elif** (contraction de else if).*

Syntaxe

```
if Condition 1 :
    Bloc d'Instructions 1
elif Condition 2 :
    Bloc d'Instructions 2
else :
    Bloc d'Instructions 3
Suite du programme
```

Exercice 5

1. **Taper** dans l'éditeur de scripts les lignes suivantes :

```
a = 10
if a <= 0 :
    print("La valeur est nulle ou négative")
elif a == 100 :
    print("La valeur est égale à 100")
elif a < 100 :
    print("La valeur est inférieure à 100")
else :
    print("La valeur est supérieure à 100")
```

2. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.
3. **Donner** les valeurs de a permettant de d'exécuter chacun des blocs d'instructions. **Tester**-les.

4 – DECOUPAGE DE CHAINE DE CARACTERES

*Une chaîne de caractère est **indexée**. Le **premier caractère** de la chaîne possède **l'index 0**. Pour connaître la **taille d'une chaine**, il faut utiliser l'instruction « `len` ».*

Exercice 6

1. **Taper** dans l'éditeur de scripts les lignes suivantes :

```
chaine = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
taille = len(chaine)
print("Taille de la chaine :",taille)
ch1 = chaine[0]
print("Chaîne 1 :",ch1)
ch2 = chaine[0:5]
print("Chaîne 2 :",ch2)
ch3 = chaine[0:5:2]
print("Chaîne 3 :",ch3)
ch4 = chaine[:5]
print("Chaîne 4 :",ch4)
ch5 = chaine[6:]
print("Chaîne 5 :",ch5)
ch6 = chaine[::2]
print("Chaîne 6 :",ch6)
```

2. **Enregistrer** puis **exécuter** le script. **Tester-le**. **Justifier** le résultat obtenu.

5 – LABYRINTHE

Le programme lab3.py est une nouvelle version du labyrinthe. Il est constitué du programme principal et de 4 fonctions.

5.1 – Programme principal

*Le **personnage** devant évoluer dans le labyrinthe sera repérer par un « **X** ». La **position du personnage** est stockée dans la **liste à deux éléments** « **pos_perso** ». Le premier élément de la liste **pos_perso[0]** donne le **numéro de la colonne** et second élément **pos_perso[1]** donne le numéro de la **ligne**. Les lignes et les colonnes sont numérotées à partir de la valeur 0.*

Exercice 7 : Labyrinthe – Programme principal

1. **Compléter**, dans le programme **lab3.py**, la ligne 103 permettant de fixer le caractère représentant le personnage :

```
perso = ..................
```

2. **Compléter**, dans le programme **lab3.py**, la ligne 104 permettant de placer le personnage sur la première case libre en haut et à gauche :

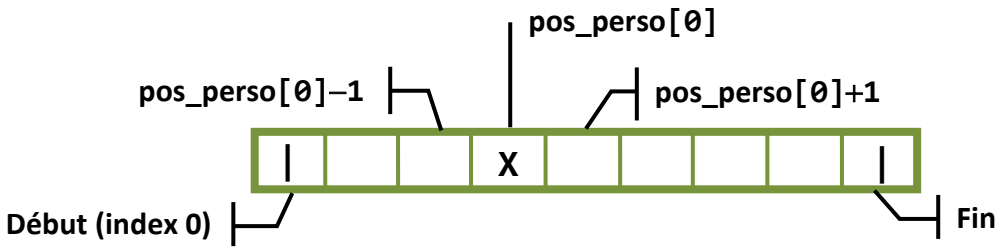
```
pos_perso = [..... , ......]
```

5.2 – Fonction affiche_labyrinthe

La fonction `affiche_labyrinthe(lab,perso,pos_perso)` *permet d'afficher le labyrinthe et le personnage. Elle reçoit comme paramètre, le labyrinthe à afficher, le caractère représentant le personnage et la position du personnage.*

L'affichage du labyrinthe est réalisé ligne à ligne.

Si la ligne à afficher correspond à la ligne sur laquelle se trouve le personnage, il faut remplacer le caractère en position **pos_perso[0]** *par le caractère « **X** » représentant le personnage. La solution consiste à réaliser l'affichage du personnage en découpant la chaîne. La ligne commence par le contenu de la chaine jusqu'à la colonne du personnage, le personnage est ensuite ajouté, puis les caractères suivants de la chaine sont placés.*




```
print(ligne[0:pos_perso[0]-1] + perso + ligne[pos_perso[0]+1:])
```

Si la ligne à afficher ne correspond pas à la ligne sur laquelle se trouve le personnage, elle est affichée entièrement comme dans la version précédente du labyrinthe.

Exercice 8 : Labyrinthe – Fonction affiche_labyrinthe

1. **Compléter**, dans le programme **lab3.py**, la ligne 11 permettant de tester si la ligne à afficher correspond à la ligne sur laquelle se trouve le personnage :

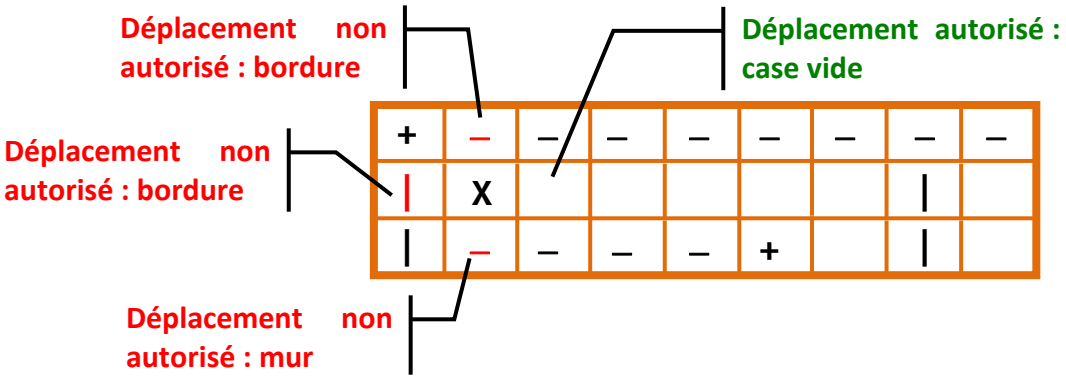
```
if n_ligne ..... :
```

2. **Compléter**, dans le programme **lab3.py**, la ligne 14 permettant d'afficher la ligne entière si elle ne correspond pas à la ligne sur laquelle se trouve le personnage.
3. **Compléter** la ligne 15 permettant d'incrémenter le numéro de la ligne qui à afficher :

```
n_ligne = .....
```

5.3 – Fonction verification_deplacement

Le personnage doit pouvoir évoluer vers toute case adjacente (horizontale ou verticale). Le déplacement ne sera toutefois pas autorisé si la case adjacente est une bordure ou un mur (c'est-à-dire une case non vide).



La fonction `verification_deplacement(lab,pos_col,pos_ligne)` *permet de vérifier que la case (définie par les coordonnées pos_col et pos_ligne) vers laquelle doit évoluer le personnage n'est pas une bordure ou un mur. S'il s'agit d'une bordure la fonction renvoie « none » sinon elle renvoie les coordonnées* ***pos_col*** *et* ***pos_ligne****.*

Exercice 9 : Labyrinthe – Fonction verification_deplacement

1. **Compléter** la ligne 29 permettant de tester si la case n'est pas vide :

```
if lab[pos_ligne][pos_col] ..... :
```

2. **Compléter** la ligne 32 permettant de retourner les variables **pos_col** et **pos_ligne** dans le cas où la case est vide.

```
return .....
```

5.4 – Fonction choix joueur

*Pour le déplacement du personnage, le joueur doit choisir la case adjacente vers laquelle doit se déplacer le personnage par l'intermédiaire des caractères « **h** » (haut), « **b** » (bas), « **g** » (gauche) ou « **d** » (droite). La fonction* `choix_joueur(lab,pos_perso)` *doit retourner, si le déplacement est autorisé (appel de la fonction* `verification_deplacement`*) les nouvelles coordonnées de la position du personnage. Si le déplacement n'est pas autorisé, la position du personnage ne doit pas changer et un message doit être affiché. La fonction* `choix_joueur` *doit permettre également de quitter le jeu si l'utilisateur rentre le caractère « **q** » (quitter). Dans ce cas là, l'instruction «* `exit(0)` *» permet d'interrompre l'exécution du programme et doit retourner la valeur 0, ce qui permet d'indiquer que le programme s'est terminé correctement. Si une autre valeur est retournée, le programme ne s'est pas arrêter correctement et à provoquer une erreur.*

Exercice 10 : Labyrinthe – Fonction choix_joueur

1. **Compléter** la ligne 46 permettant de demander au joueur le choix de déplacement :

```
choix = ..........................................................
```

2. **Compléter** les lignes 47 et 48 permettant de vérifier si la case au-dessus est vide dans le cas où joueur entre le caractère « **H** » ou « **h** ».
3. **Indiquer** à quoi va correspondre la variable **dep** dans le cas où si la case au-dessus n'est vide puis dans le cas où elle est vide.
4. **Compléter** les lignes 49 et 50 permettant de vérifier si la case au-dessous est vide dans le cas où joueur entre le caractère « **B** » ou « **b** ».
5. **Compléter** les lignes 51 et 52 permettant de vérifier si la case à gauche est vide dans le cas où joueur entre le caractère « **G** » ou « **g** ».
7. **Compléter** les lignes 53 et 54 permettant de vérifier si la case à droite est vide dans le cas où joueur entre le caractère « **D** » ou « **d** ».
8. **Compléter** les lignes 55 et 56 permettant de quitter le programme dans le cas où joueur entre le caractère « **Q** » ou « **q** ».
9. **Compléter** la ligne 58 permettant d'afficher « Choix non valide » dans le cas où joueur entre un caractère ne correspondant pas aux cas précédents.
10. **Compléter** les lignes 61 et 62 permettant d'afficher « Déplacement non autorisé » dans le cas où la case à atteindre n'est pas vide.

5.4 – Fonction jeu

La fonction `jeu(level,perso,pos_perso)` *permet de gérer le jeu : afficher la labyrinthe, demander au joueur où il souhaite déplacer son personnage et recommencer indéfiniment.*

Algorithme de la fonction jeu

Début
    Tant que vraie
        Afficher le labyrinthe et le personnage
        Demander et définir le déplacement du personnage
    Fin Tant que
Fin

La boucle « Tant que » est répétée tant que la condition est vérifiée. Ici la condition est : vraie == vraie. Elle est donc toujours vérifiée. Le bloc est donc répété indéfiniment. Une boucle « Tant que » est réalisée par l'intermédiaire de l'instruction « while *» :*

Syntaxe

```
while Condition :
    Bloc d'Instructions
Suite du programme
```

Exercice 11 : Labyrinthe

1. **Compléter** les lignes 75 et 76 permettant d'afficher le labyrinthe par l'appel de la fonction affiche_labyrinthe et de définir le déplacement du personnage par l'appel de la fonction choix_joueur.
2. **Tester** le fonctionnement du programme et des différentes fonctions.