

Devoir surveillé n°1

NOM, Prénom :

Classe :

Quelques consignes de présentation :

- compléter les réponses directement sur le sujet,
- écrire lisiblement et sans ratures,
- encadrer les réponses aux questions.

Les codes demandés doivent être écrits en *Python*, comme le sont les codes proposés dans l'énoncé.

Exercice 1 QCM Architecture/Web

Chaque question comporte au moins une bonne réponse et parfois plusieurs.

Questions	Réponses
1. Dans l'architecture de Von Neumann, le microprocesseur contient	<input type="checkbox"/> la mémoire <input type="checkbox"/> l'unité de contrôle <input type="checkbox"/> l'unité arithmétique et logique
2. Dans l'architecture de Von Neumann, la mémoire vive (RAM)	<input type="checkbox"/> contient le programme et les données <input type="checkbox"/> contient seulement des données <input type="checkbox"/> contient seulement des programmes <input type="checkbox"/> est persistante sans alimentation électrique <input type="checkbox"/> est adressable en lecture/écriture par des câbles appelés bus
3. Dans le fichier source d'une page Web, le code qui permet de créer un lien hypertexte vers la page d'adresse <code>http://site.fr</code> est :	<input type="checkbox"/> <code>lien</code> <input type="checkbox"/> <code>http://site.fr</code> <input type="checkbox"/> <code>http://site.fr</code> <input type="checkbox"/> <code><href a="http://site.fr">lien</href></code>
4. URL est un acronyme qui désigne :	<input type="checkbox"/> le protocole d'échange entre client et serveur Web <input type="checkbox"/> le langage des pages Web <input type="checkbox"/> le format d'adressage des pages Web

Exercice 2 : Comprendre et modifier un code ... points

L'Indice de Masse Corporelle se calcule par la formule $IMC = \frac{\text{masse}}{\text{taille}^2}$ où la masse est en kilogrammes et la taille en mètres. Un IMC est considéré comme normal s'il est compris entre 18,5 et 25. En dessous de 18,5, la personne est en sous-poids et au-dessus de 25 elle est en sur-poids.

1. Compléter le programme ci-dessous pour qu'il calcule l'IMC en fonction de la masse et de la taille saisis par l'utilisateur et qu'il affiche sa classification.

```
masse = float(input("Saisir la masse en kilogrammes : "))
taille = float(input("Saisir la taille en mètres : "))
imc = .....
if imc < 18.5:
    print("Sous-poids")
else:
    if imc ..... :
        print("Surpoids")
    else:
        print("Poids normal")
```

2. Proposer une autre écriture de la structure conditionnelle utilisant la syntaxe `if ...elif ...else`.

Exercice 3 : Comprendre un code ... points

On considère quatre fonctions écrites en Python :

```
def f1(n):
    if n < 740:
        if n >= 735:
            n = n + 5
    if n >= 740:
        n = n + 1
    return n
```

```
def f2(n):
    if n < 740:
        if n >= 735:
            n = n + 5
    else:
        n = n + 1
    return n
```

```
def f3(n):
    if n < 740:
        if n >= 735:
            n = n + 5
        else:
            n = n + 1
    return n
```

```
def f4(n):
    if n >= 740:
        n = n + 1
    elif n >= 735:
        n = n + 5
    return n
```

1. Compléter les valeurs retournées par les appels de fonctions :

- f1(736) retourne, f1(741) retourneet f1(730) retourne
- f2(736) retourne, f2(741) retourneet f2(730) retourne
- f3(736) retourne, f3(741) retourneet f3(730) retourne
- f4(736) retourne, f4(741) retourneet f4(730) retourne

2. Sans justifier, donner les fonctions qui retournent la même valeur pour tout entier n passé en argument.

Exercice 4 : Compléter un code ... points

Au jeu de molki, chaque joueur marque à son tour de jeu entre 0 et 12 points qui viennent s'ajouter à son score précédent. Le premier à atteindre le score de 51 gagne. Un cas d'égalité se produit si plusieurs joueurs atteignent 51 lors du même tour.

Mais quiconque dépasse le score de 51 revient immédiatement à 25 points!

1. Compléter le code de la fonction `nouveauscore(score, points)` ci-dessous pour qu'elle retourne le nouveau score lorsqu'on lui passe en argument le score précédent et le nombre de points obtenu lors du tour de jeu.

```
def nouveauscore(score, points):
    nouveau = score + points
    if ..... :
        return .....
    else:
        return .....
```

2. Compléter le code du programme ci-dessous pour qu'il simule une partie de molki entre deux joueurs. On considère que la fonction précédente `nouveauscore` est disponible car elle a été définie dans l'en-tête du programme.

```
score_joueur1 = 0
score_joueur2 = 0
while ..... and ..... :
    print("Nouveau tour de jeu")
    points1 = int(input("Points du joueur 1 ? "))
```

```

    points2 = int(input("Points du joueur 2 ? "))
    .....
    .....
if .....:
    print("Le joueur 1 a gagné !")
elif .....:
    print("Le joueur 2 a gagné !")
else:
    print("Égalité !")

```

Exercice 5 : Maîtriser les boucles ... points

1. Écrire un code Python qui affiche les entiers de 1 à 10 dans l'ordre croissant.

.....

.....

.....

2. Écrire un code Python qui affiche les entiers de 10 à 1 dans l'ordre décroissant.

.....

.....

.....

3. Corriger ce code Python pour qu'il affiche le plus petit entier n tel que $n^2 + n \geq 100000$:

```

n = 0
while n ** 2 + n >= 100000:
    n = n + 1
print(n)

```

.....

.....

.....

.....

4. Écrire un code Python qui affiche le plus grand entier n tel que $n^2 + n < 100000$:

.....

.....

.....

Exercice 6 : Exécuter un code ... points

```
x = int(input("x :"))
y = int(input("y :"))
z = int(input("z :"))
x = x - y - z
y = x + y + z
z = y - z - x
x = y - z - x
```

On considère le programme ci-dessus qu'on exécute lors de deux tests :

- dans le test 1 les variables **x**, **y** et **z** sont initialisées respectivement avec les valeurs 4, 2 et 1 ;
- dans le test 2 les variables **x**, **y** et **z** sont initialisées respectivement avec les valeurs formelles **a**, **b** et **c** qui représentent des nombres quelconques ;

Compléter le tableau ci-dessous pour chaque test avec l'état de la mémoire après l'exécution de chaque instruction.

	État de la mémoire					
Instruction	x (test 1)	y (test 1)	z (test 1)	x (test 2)	y (test 2)	z (test 2)
<code>x = int(input("x :"))</code>	4	×	×	a	×	×
<code>y = int(input("y :"))</code>	4	2	×	a	b	×
<code>z = int(input("z :"))</code>	4	2	1	a	b	c
<code>x = x - y - z</code>						
<code>y = x + y + z</code>						
<code>z = y - z - x</code>						
<code>x = y - z - x</code>						

Exercice 7 : Comprendre puis écrire du code ... points

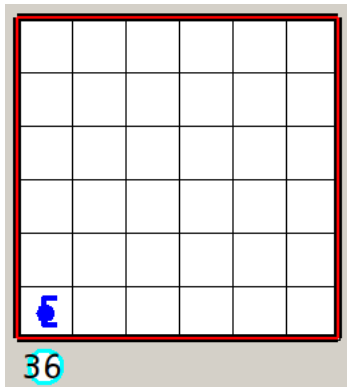
Dans le module **Rurple** on peut déplacer un robot dans une grille avec le langage Python enrichi avec les instructions suivantes :

- `avance()` fait avancer le robot d'une case dans l'orientation fixée par ses bras ;
- `gauche()` fait pivoter sur lui-même le robot vers la gauche (il reste sur la même case) ;

- `depose()` lui fait déposer une bille au sol ;
- `mur_devant()` retourne `True` si un mur se trouve devant lui ou `False` sinon ;
- `bille_devant()` peut être utilisée si une case se trouve devant le robot, elle retourne `True` si la case contient une bille ou `False` sinon ;
- `bille_en_poche()` retourne `True` si le robot possède encore une bille en poche et `False` sinon.

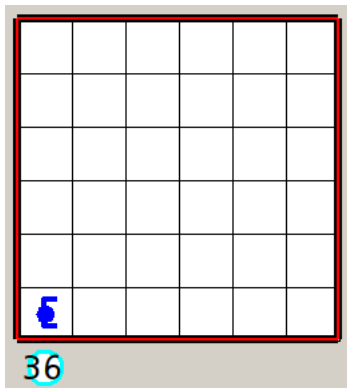
Dans chaque question, le robot est positionné initialement sur la case située dans le coin inférieur gauche d'une grille de 6×6 cases et il dispose de 36 billes en poche.

1. Compléter la grille avec les billes déposées par le robot après exécution du code ci-dessous :



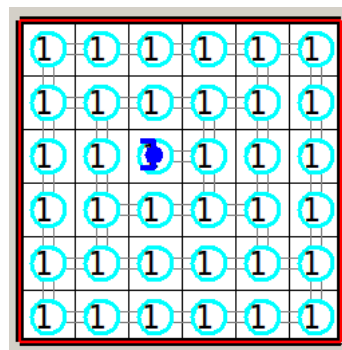
```
for i in range(4):
    for j in range(5):
        avance()
        depose()
        gauche()
```

2. Compléter la grille avec les billes déposées par le robot après exécution du code ci-dessous :



```
for i in range(4):
    while not mur_devant():
        avance()
        depose()
        gauche()
```

3. Écrire un programme qui fait déposer par le robot une bille sur chaque case de la grille.



.....

.....

.....

.....

.....

.....

.....

.....

.....