Corrigé du devoir surveillé n°1

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	☐ la mémoire ■ l'unité de contrôle ■ l'unité arithmétique et logique
2. Dans l'architecture de Von Neumann, la mémoire vive (RAM)	■ contient le programme et les données ☐ contient seulement des données ☐ contient seulement des programmes ☐ est persistante sans alimentation électrique ■ 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 `http://site.fr` est :	■ `<a href="http://site.fr">lien</a>` ■ `<a href="http://site.fr">http://site.fr</a>` ☐ `<a href="lien hypertexte">http://site.fr</a>` ☐ `<href a="http://site.fr">lien</href>`
4. URL est un acronyme qui désigne :	☐ le protocole d'échange entre client et serveur Web ☐ le langage des pages Web ■ 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 kilogrammes : "))
imc = masse / taille ** 2
if imc < 18.5:
    print("Sous-poids")
else:
    if imc > 25 :
        print("Surpoids")
    else:
        print("Poids normal")
```

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

```
masse = float(input("Saisir la masse en kilogrammes : "))
taille = float(input("Saisir la taille en kilogrammes : "))
imc = masse / taille ** 2
if imc < 18.5:
    print("Sous-poids")
elif imc > 25:
    print("Surpoids")
else:
    print("Poids normal")
```

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 742, `f1(741)` retourne 742 et `f1(730)` retourne 735.
   - `f2(736)` retourne 741, `f2(741)` retourne 742 et `f2(730)` retourne 730.
   - `f3(736)` retourne 741 , `f3(741)` retourne 741 et `f3(730)` retourne 731.
   - `f4(736)` retourne 741 , `f4(741)` retourne 742 et `f4(730)` retourne 730.
2. Sans justifier, donner les fonctions qui retournent la même valeur pour tout entier `n` passé en argument.

   Les fonctions `f2` et `f4` 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.

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 nouveau > 51:
        return 25
    else:
        return nouveau
```

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 score_joueur1 < 51 and score_joueur2 < 51:
    print("Nouveau tour de jeu")
    points1 = int(input("Points du joueur 1 ? "))
    points2 = int(input("Points du joueur 2 ? "))
    score_joueur1 = nouveauscore(score_joueur1, points1)
    score_joueur2 = nouveauscore(score_joueur1, points1)
```

```
if score_joueur1 == 51:
    print("Le joueur 1 a gagné !")
elif  score_joueur2 == 51:
    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.

```
for n in range(1, 11):
    print(n)
```

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

```
for n in range(10, -1, -1):
    print(n)
```

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

Code faux

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

Code correct

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

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

```
n = 0
while n ** 2 + n < 100000:
    n = n + 1
print(n - 1)
```

Exercice 6 : Éxécuter 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)
`x = int(input("x :"))`	4	×	×	a	×	×
`y = int(input("y :"))`	4	2	×	a	b	×
`z = int(input("z :"))`	4	2	1	a	b	c
`x = x - y - z`	$4-2-1=1$	2	1	$a-b-c$	b	c
`y = x + y + z`	1	$1+2+1=4$	1	$a-b-c$	$a-b-c+b+c=a$	c
`z = y - z - x`	1	4	$4-1-1=2$	$a-b-c$	a	$a-(a-b-c)-c=b$
`x = y - z - x`	$4-2-1=1$	4	2	$a-b-(a-b-c)=c$	a	b

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

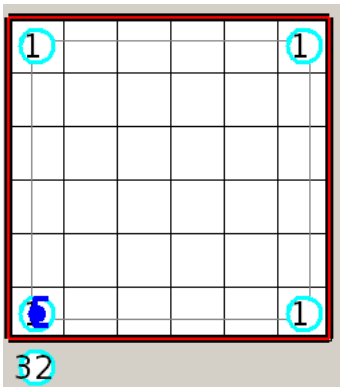
Dans le module `Rurple` on peut déplacer un robot dans une grille avec le langage `Python` enrichi avec les instructions suivantes :

- `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 e ;
- `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.

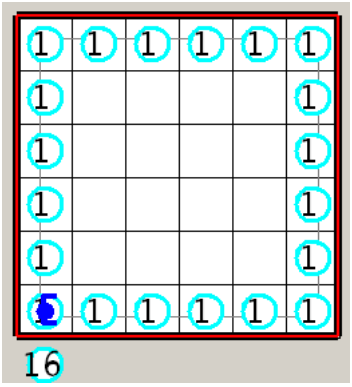
Dans chaque question, le robot est positionné initialement sur la case située dans le coin inférieur gauche d'une grille de 6×6 cases et il dispose de 36 billes en poche.

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



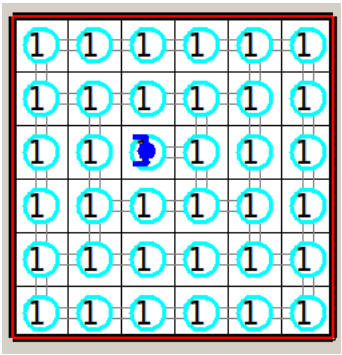

```
for i in range(4):
    for j in range(5):
        avance()
    depose()
    gauche()
```

2. Voici 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.



```
def bille_devant():
    '''Fonction non définie par défaut
        dans rurple'''
    avance()
    reponse = bille_au_sol()
    #demi-tour sur place
    for k in range(2):
        gauche()
    avance()
    #demi-tour sur place
    for k in range(2):
        gauche()
    return reponse

for k in range(36):
    depose()
    if mur_devant() or bille_devant():
        gauche()
    avance()
```