Programmation des objets connectés en Python

> **Nom Prénom :**

Lecture de programmes

Script n°1

Au lancement du programme, les trois LEDs sont éteintes.

La fonction `setLedState(led,state)` permet d'allumer ou d'éteindre une LED. Elle prend en paramètre le nom de la LED et l'état à considérer, `True` pour l'allumer, `False` pour l'éteindre.

```
from quickpi import *
setLedState("rouge",True)
sleep(500)
setLedState("rouge",False)
for loop in range(5):
    setLedState("verte",True)
    sleep(100)
    setLedState("verte",False)
    sleep(100)
for loop in range(3):
    setLedState("bleu",True)
    sleep(50)
    setLedState("bleu",False)
    sleep(50)
setLedState("verte",True)
```

- Durant combien de temps la LED rouge reste allumé ? ...
- Quelle est l'état final de la LED verte ? ...
- Représenter par chronogramme l'état des LEDs rouge, verte et bleu en fonction du temps tout au long de l'exécution du programme :

R...

V...

B...

Script n°2

Au lancement du programme, les trois LEDs sont éteintes.

La fonction `toggleLedState(led)` permet d'inverser l'état de la LED entrée en paramètre sous forme de chaine de caractères.

```
from quickpi import *
for i in range(2) :
    toggleLedState('rouge')
    for j in range(3) :
        toggleLedState('verte')
        for k in range(4) :
            toggleLedState('bleu')
            sleep(100)
```

- Combien de fois la LED bleu changera d'état ? ...
- Durant combien de temps au total la LED rouge sera allumée ? ...
- Durant combien de temps au total la LED verte sera éteinte ? ...

Script n°3

La fonction `setServoAngle(servo, angle)` permet de modifier l'angle du servomoteur choisi. L'angle est exprimés en degrés, entre 0 et 180 degrés.
La fonction `getServoAngle(servo)` permet de relire l'angle auquel on a réglé le servomoteur choisi. Ce n'est pas un capteur, mais simplement une mémorisation de la dernière valeur modifiée par une instruction.

La fonction `isButtonPressed("stick.left")` renvoie `True` si le bouton de gauche de la manette est enfoncé, et `False` s'il est relevé. De même pour le bouton de droite avec la fonction `isButtonPressed("stick.right")`.

Au lancement du programme, les trois LEDs sont éteintes.

```
from quickpi import *
while True :
    setServoAngle("servo", 90)
    for t in range(100) :
        if isButtonPressed("stick.left") :
            if 10 < getServoAngle("servo") :
                setServoAngle("servo", getServoAngle("servo") - 5)
        if isButtonPressed("stick.right") :
            if getServoAngle("servo") < 170 :
                setServoAngle("servo", getServoAngle("servo") + 5)
        sleep(20)
    if getServoAngle("servo") == 90 :
        setLedState('verte', True)
    elif getServoAngle("servo") < 90 :
        setLedState('rouge', True)
    else :
        setLedState('bleu', True)
    sleep(2000)
    setLedState('verte', False)
    setLedState('rouge', False)
    setLedState('bleu', False)
```

- Quelle LED va s'allumer si après 2 secondes aucun des boutons de la manette n'a été enfoncé ?

...

- Que se passera-t'il si on appuie 5 fois sur le bouton de droite entre 2 et 4 secondes plus tard ?

...

- Quelle est alors la position angulaire du servo ?

...

- A partir de cette situation, que faut-il faire au moins pour que la LED rouge s'allume ?

...

Comparaison de programmes

Les fonctions `turnLedOn()` et `turnLedOff()` permettent respectivement d'allumer et d'éteindre une LED.

La fonction `isButtonPressed("A")` renvoie `True` si le bouton A est enfoncé, et `False` s'il est relevé. De même pour le bouton B avec la fonction `isButtonPressed("B")`.

Parmi les programmes A, B, C, D, E, F proposés ci-dessous, lesquels sont équivalents entre eux ?

...

...

...

Script A

```
from quickpi import *
while True :
    if isButtonPressed("A") :
        turnLedOn()
    elif isButtonPressed("B") :
        turnLedOn()
    else :
        turnLedOff()
```

Script B

```
from quickpi import *
while True :
    if isButtonPressed("A") and isButtonPressed("B") :
        turnLedOn()
    else :
        turnLedOff()
```

Script C

```
from quickpi import *
while True :
    if isButtonPressed("A") or isButtonPressed("B") :
        turnLedOn()
    else :
        turnLedOff()
```

Script D

```
from quickpi import *
while True :
    if isButtonPressed("A") :
        if isButtonPressed("B") :
            turnLedOn()
    else :
        turnLedOff()
```

Script E

```
from quickpi import *
while True :
    if isButtonPressed("A") :
        turnLedOn()
    if isButtonPressed("B") :
        turnLedOn()
    else :
        turnLedOff()
```

Script F

```
from quickpi import *
while True :
    if not (not isButtonPressed("A") or not isButtonPressed("B")) :
        turnLedOn()
    else :
        turnLedOff()
```

Ecriture de programmes

La fonction `turnBuzzerOn()` permet d'allumer le buzzer, tandis que `turnBuzzerOff()` permet d'éteindre le buzzer.
La fonction `readDistance(range)` renvoie la distance mesurée par le capteur de distance entré en paramètre. Cette distance est exprimée en centimètres.

- Compléter le script suivant pour que le buzzer sonne en continu si la distance mesurée par le capteur est inférieure à 0,3 m et qu'il emmette des bips alternativement toutes les demi seconde quand le

capteur détecte un objet compris entre 0,3 m et 1 m, et arrête le buzzer dès qu'il n'en détecte pas à moins de 1 m.

```
from quickpi import *
while True :
    if readDistance("capteur") ........ :
        turnBuzzerOn()
    else :
        if readDistance("capteur") ........ :









        else :
            turnBuzzerOff()
```

- Réécrire ce programme ci-dessous en le modifiant avec l'aide d'une variable pour que la fréquence d'alternance des bips augmente proportionnellement à mesure que la distance captée diminue entre 1 m et 0,3 m...

```
from quickpi import *
while True :

    distance = ........................

        if ......................... :
        turnBuzzerOn()
    else :
        if ......................... :









        else :
            turnBuzzerOff()
```