TP-Quick_Pi-Objets_connectes-Parcours_1-Correction

September 12, 2020

1 Parcours 1 : des objets qui réagissent

1.1 Crédits

Auteur du document original : **Eric Madec** de l'académie de Rennes. Le document a été légèrement modifié.

1.2 Thèmes du programme

Contenus	Capacités attendues	Commentaires
Périphériques d'entrée et de sortie Interface HommeMachine (IHM)	Identifier le rôle des capteurs et actionneurs. Réaliser par programmation une IHM répondant à un cahier des charges donné.	Les activités peuvent être développées sur des objets connectés, des systèmes embarqués ou robots.
Interaction avec l'utilisateur dans une page Web	Analyser et modifier les méthodes exécutées lors d'un clic sur un bouton d'une page Web	

Contenus	Capacités attendues	Commentaires
Constructions élémentaires	Mettre en évidence un corpus de constructions élémentaires.	Séquences, affectation, conditionnelles, boucles bornées, boucles non bornées, appels de fonction.

Contenus	Capacités attendues	Commentaires
Utilisation de bibliothèques	Utiliser la documentation d'une bibliothèque.	Aucune connaissance exhaustive d'une bibliothèque particulière n'est exigible.

1.3 Consignes

Rendez-vous à l'adresse : https://amazon.quick-pi.org/ pour apprendre à programmer vos objets connectés en Python en réalisant les activités du Parcours 1.

Copier ci-dessous le code d'accès personnel fourni par l'enseignant :

……

Lisez la présentation pour programmer en python , écrire, tester puis valider votre programme avec interface France IOI.

> Au fur et à mesure que vous expérimentez et validez vos programmes **par simulation** pour chaque étape, recopiez votre script dans les cellules de code vides ci-dessous…

Un tutoriel video de présentation de QuickPi https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

1.4 Mélodie :

Corrigé video : https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation du buzzer et de gestion du temps .

Écrire un programme qui active le buzzer pour jouer un son.

[2]:
```
# Recopier le code ici
from quickpi import *
turnBuzzerOn()
```

Écrire un programme qui joue la note “la”, à la fréquence 440Hz, pendant une seconde.

[ ]:
```
# Recopier le code ici
from quickpi import *
setBuzzerNote("buzzer1", 440)
sleep(1000)
turnBuzzerOff()
```

Écrire un programme qui joue la mélodie “do ré mi ré do”, en jouant chaque note pendant 500ms, l’une après l’autre, puis éteint le buzzer.

Les fréquences des notes sont : Do : 523Hz, Ré : 587Hz, et Mi : 659Hz.

[ ]:
```
from quickpi import *
setBuzzerNote("buzzer1", 523)
sleep(500)
setBuzzerNote("buzzer1", 587)
sleep(500)
setBuzzerNote("buzzer1", 659)
sleep(500)
setBuzzerNote("buzzer1", 587)
sleep(500)
setBuzzerNote("buzzer1", 523)
sleep(500)
turnBuzzerOff()
```

1.5 Alternance

Corrigé video https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation de la boucle de répétition avec l'instruction `for` , des leds et de la gestion du temps

Écrire un programme qui fait clignoter la LED cinq fois : c'est à dire l'allume pendant 1s, puis l'éteint pendant 1s, puis recommence quatre autres fois.

[ ]:
```
# Recopier le code ici
from quickpi import *
for k in range(5):
    turnLedOn()
    sleep(1000)
    turnLedOff()
    sleep(1000)
```

Écrire un programme qui allume en alternance les LED rouge et bleue pendant 500ms chacune : la rouge pendant 500ms, la bleue pendant 500ms, etc.

Chacune des LEDs doit être allumée 5 fois au total.

Tout doit être éteint à la fin.

[ ]:
```
# Recopier le code ici
from quickpi import *
for k in range(5):
    setLedState("red1", True)
    sleep(500)
    setLedState("red1", False)
    setLedState("blue1", True)
    sleep(500)
    setLedState("blue1", False)
```

Écrire un programme qui allume en alternance les LED rouge et bleue 5 fois au total, chaque fois pendant 500ms mais allumant la suivante 100ms avant d'éteindre la précédente : on allume la rouge au tout début, puis allume la bleue au temps 400ms, puis éteint la rouge au temps 500ms, puis allume la rouge au temps 800ms, etc.

Solution 1 (correcte mais utilise plus de blocs que le nombre autorisé)

[ ]:
```
# Recopier le code ici
from quickpi import *
```

```
setLedState("red1",True)
sleep(400)
for k in range(4):
    setLedState("blue1", True)
    sleep(100)
    setLedState("red1", False)
    sleep(300)
    setLedState("red1", True)
    sleep(100)
    setLedState("blue1", False)
    sleep(300)
sleep(100)
setLedState("red1", False)
sleep(400)
setLedState("blue1", False)
```

Solution 2

[ ]:
```
# Recopier le code ici
from quickpi import *
for k in range(5):
    setLedState("red1", True)
    sleep(100)
    setLedState("blue1", False)
    sleep(300)
    setLedState("blue1", True)
    sleep(100)
    setLedState("red1", False)
    sleep(300)
sleep(100)
setLedState("blue1", False)
```

1.6 Show lumineux 1

Corrigé video https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation de boucle for, des leds et de gestion du temps.

Écrire un programme qui dans l'ordre :

- Allume la LED rouge pendant 1s
- Fait clignoter la LED bleue 5 fois : 500ms allumée puis 500ms éteinte.
- Allume la LED verte et la laisse allumée.

[ ]:
```
# Recopier le code ici
from quickpi import *
setLedState("red1", True)
sleep(1000)
setLedState("red1", False)
for k in range(5):
    setLedState("blue1", True)
    sleep(500)
    setLedState("blue1", False)
    sleep(500)
setLedState("green1", True)
```

Écrire un programme qui dans l'ordre :

- Allume la LED rouge pendant 1s
- Fait clignoter la LED verte 5 fois : 500ms allumée puis 500ms éteinte.
- Fait clignoter la LED bleue 3 fois : 500ms allumée puis 500ms éteinte.
- Rallume la LED verte et la laisse allumée.

[ ]:
```
 # Recopier le code ici
from quickpi import *
setLedState("red1", True)
sleep(1000)
setLedState("red1", False)
for k in range(5):
    setLedState("green1", True)
    sleep(500)
    setLedState("green1", False)
    sleep(500)
for k in range(3):
    setLedState("blue1", True)
    sleep(500)
    setLedState("blue1", False)
    sleep(500)
setLedState("green1", True)
```

Écrire un programme qui joue la séquence suivante, où chaque tiret représente un état allumé de 500ms, et chaque point un état éteint de 500ms.

Notez que les deux lignes se jouent en même temps :

```
- LED rouge : -.-.-.--.-.-.
- LED verte : .......------
```

Les LEDs doivent être éteintes à la fin du programme.

[ ]:
```
# Recopier le code ici
from quickpi import *
setLedState("green1", False)
for k in range(3):
    setLedState("red1", True)
    sleep(500)
    setLedState("red1", False)
    sleep(500)
setLedState("red1", True)
sleep(500)
setLedState("green1", True)
for k in range(3):
    setLedState("red1", True)
    sleep(500)
    setLedState("red1", False)
    sleep(500)
setLedState("green1", False)
```

1.7 Quelle direction ?

Corrigé video https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953.

Lisez la présentation de : boucle while, l'instruction if, l'écran, le bouton poussoir et la manette.

Écrire un programme qui laisse l'écran vide au début, puis affiche le texte "Bonjour" dès que l'on appuie sur le bouton, et le laisse affiché.

Votre programme devra boucler indéfiniment, et tester en permanence si le bouton est enfoncé. Notez qu'un texte affiché à l'écran y reste jusqu'à ce que l'on affiche autre chose.

[ ]:
```
 # Recopier le code ici
from quickpi import *
while True:
    if isButtonPressed():
        displayText("Bonjour")
```

Écrire un programme qui :

- affiche le texte “Appuyez”
- affiche le texte “Merci” dès que l’on appuie sur le bouton, et le laisse affiché.

[ ]:
```
 # Recopier le code ici
from quickpi import *
displayText("Appuyez")
while True:
    if isButtonPressed():
        displayText("Merci")
```

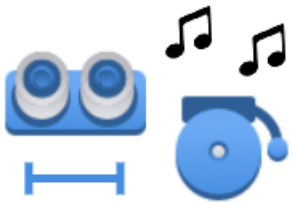
Écrire un programme qui :

- affiche le texte “Direction ?” dès le début
- puis lorsque l’on appuie sur une direction, affiche le texte “Haut”, “Droite”, “Bas” ou “Gauche” selon la direction appuyée.

[ ]:
```
 # Recopier le code ici
from quickpi import *
displayText("Direction ?")
while True:
    if isButtonPressed("stick1.up"):
        displayText("Haut")
    elif isButtonPressed("stick1.down"):
        displayText("Bas")
    elif isButtonPressed("stick1.left"):
        displayText("Gauche")
    elif isButtonPressed("stick1.right"):
        displayText("Droite")
```

1.8 Instrument

Corrigé video : https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953



Lisez la présentation de boucle while, de l’instruction if, du capteur de distance et du buzzer.

Écrire un programme qui allume le buzzer quand le capteur détecte un objet à moins de 100cm, et arrête le buzzer dès qu’il n’en détecte plus.

[ ]:
```
# Recopier le code ici
from quickpi import *
while True:
```

```
    if readDistance("distance1") < 100:
        turnBuzzerOn()
    else:
        turnBuzzerOff()
```

Écrire un programme qui allume le buzzer quand le capteur détecte un objet à moins de 500cm.

Le buzzer doit jouer un son à une fréquence égale à la distance de l'objet en centimètres. Il doit être éteint s'il n'y a pas d'objet à moins de 500cm.

[ ]:
```
 # Recopier le code ici
from quickpi import *
while True:
    if readDistance("distance1") < 500:
        setBuzzerNote("buzzer1", readDistance("distance1"))
    else:
        turnBuzzerOff()
```

Modifier le programme de la version précédente, avec la différence suivante : lorsque le bouton est enfoncé, la fréquence du buzzer doit être doublée.

[ ]:
```
 # Recopier le code ici
from quickpi import *
while True:
    if readDistance("distance1") < 500:
        if isButtonPressed():
            setBuzzerNote("buzzer1", 2 * readDistance("distance1"))
        else:
            setBuzzerNote("buzzer1", readDistance("distance1"))
    else:
        turnBuzzerOff()
```

1.9 Show Lumineux 2 :

corrigé video :https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation boucles imbriquées avec l'instruction for, des leds et de gestion du temps.

Écrire un programme qui fait 3 fois ces deux étapes :

- Faire clignoter la LED rouge 3 fois (200ms allumée puis 200ms éteinte)
- Attendre 1 seconde.

[ ]:
```
# Recopier le code ici
from quickpi import *
for i in range(3):
    for j in range(6):
        toggleLedState("red1")
        sleep(200)
    sleep(1000)
```

Écrire un programme qui fait 3 fois ces deux étapes :

- Clignoter la LED rouge 3 fois (200ms allumée puis 200ms éteinte)
- Clignoter la LED verte 3 fois (200ms allumée puis 200ms éteinte)
- Clignoter la LED bleue 3 fois (200ms allumée puis 200ms éteinte)
- Attendre 1 seconde.

[ ]:
```
 # Recopier le code ici
from quickpi import *
for i in range(3):
    for j in range(6):
        toggleLedState("red1")
        sleep(200)
    for j in range(6):
        toggleLedState("green1")
        sleep(200)
    for j in range(6):
        toggleLedState("blue1")
        sleep(200)
    sleep(1000)
```

Écrire un programme qui fait en même temps, et pendant 4 secondes au total :

- Clignoter la LED rouge : 1s allumée puis 1s éteinte
- Clignoter la LED verte : 500ms allumée puis 500ms éteinte
- Clignoter la LED bleue : 250ms allumée puis 250ms éteinte

[ ]:
```
 # Recopier le code ici
from quickpi import *
for i in range(4):
    toggleLedState("red1")
    for j in range(2):
        toggleLedState("green1")
        for j in range(2):
            toggleLedState("blue1")
            sleep(250)
```

1.10 Avertisseur :

Corrigé video : https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation de boucle while, de l'instruction if, de boucle for, du capteur de distance, du buzzer, et de gestion du temps.

Écrire un programme qui, dès qu'un objet passe à moins de 30cm du capteur, fait dans l'ordre : - Jouer 3 bips de 100ms espacés de 100ms - Attendre 500ms

[ ]:
```
 # Recopier le code ici
from quickpi import *
while True:
    if readDistance("distance1") < 30:
        for i in range(3):
            turnBuzzerOn()
            sleep(100)
            turnBuzzerOff()
            sleep(100)
        sleep(500)
```

Modifiez le programme de la version précédente, pour qu'il arrête de jouer des bips dès qu'il n'y a plus d'objet.

S'il y a un bip en cours, il se termine, mais on ne joue pas le suivant.

[ ]:
```
# Recopier le code ici
from quickpi import *
while True:
    if readDistance("distance1") < 30:
        for i in range(3):
            if readDistance("distance1") < 30:
                turnBuzzerOn()
                sleep(100)
                turnBuzzerOff()
                sleep(100)
        sleep(500)
```

Modifiez le programme de la version précédente, pour que lorsqu'un objet passe à moins de 10cm du capteur alors qu'une série de bips n'est pas en cours, alors il joue un bip en continu.

Au dessus de cette distance, il doit faire comme dans la version précédente.

[ ]:
```python
# Recopier le code ici
from quickpi import *
while True:
    if readDistance("distance1") < 10:
        turnBuzzerOn()
    else:
        turnBuzzerOff()
        if readDistance("distance1") < 30:
            for i in range(3):
                if readDistance("distance1") < 30:
                    turnBuzzerOn()
                    sleep(100)
                    turnBuzzerOff()
                    sleep(100)
            sleep(500)
```

1.11 Servo chronométré

Corrigé : https://tube.ac-lyon.fr/videos/watch/playlist/33c4984a-b64e-46cb-b0eb-3118f9728953

Lisez la présentation du servomoteur, de boucle while, de l'instruction if, de la boucle for, de Section 4.6, du buzzer, des leds, et de gestion du temps.

Écrire un programme qui met l'angle du servomoteur à 0°, puis lorsque l'on appuie sur le bouton :

- Augmente l'angle 18 fois de 10°, toutes les 50ms
- Joue un bip pendant 500ms
- Remet l'angle à 0°

[ ]:
```python
# Recopier le code ici
from quickpi import *
setServoAngle("servo1", 0)
while True:
    if isButtonPressed():
        for k in range(1, 19):
            setServoAngle("servo1", k * 10)
            sleep(50)
        turnBuzzerOn()
        sleep(500)
        turnBuzzerOff()
        setServoAngle("servo1", 0)
```

Écrire un programme qui met le servomoteur à 10°, puis lorsque que le stick est appuyé :

- À gauche : diminue de 2° puis attend 50ms
- À droite : augmente de 2° puis attend 50ms

Le programme doit ignorer les actions qui ne gardent pas l'angle entre 10° et 170°.

[ ]:
```
# Recopier le code ici
from quickpi import *
setServoAngle("servo1", 10)
while True:
    if isButtonPressed("stick1.left"):
        if 10 <= getServoAngle("servo1") - 2 <= 170:
            setServoAngle("servo1", getServoAngle("servo1") - 2)
            sleep(50)
    elif isButtonPressed("stick1.right"):
        if 10 <= getServoAngle("servo1") + 2 <= 170:
            setServoAngle("servo1", getServoAngle("servo1") + 2)
            sleep(50)
```

Modifier le programme de la version précédente, pour qu'il arrête au bout de 5s, puis selon l'angle du servo :

- S'il vaut 90°, allume la LED verte
- S'il est inférieur à 90°, allume la LED rouge
- S'il est supérieur à 90°, allume la LED bleue

Le programme doit ensuite attendre 2s, puis éteindre les LEDs et tout recommencer.

[ ]:
```
# Recopier le code ici
from quickpi import *
while True:
    setServoAngle("servo1", 10)
    for k in range(100):
        if isButtonPressed("stick1.left"):
            if 10 <= getServoAngle("servo1") - 2 <= 170:
                setServoAngle("servo1", getServoAngle("servo1") - 2)
        elif isButtonPressed("stick1.right"):
            if 10 <= getServoAngle("servo1") + 2 <= 170:
                setServoAngle("servo1", getServoAngle("servo1") + 2)
        sleep(50)
    if getServoAngle("servo1") < 90:
        setLedState("red1", True)
    elif getServoAngle("servo1") > 90:
        setLedState("blue1", True)
    else:
        setLedState("green1", True)
    sleep(2000)
    setLedState("red1", False)
    setLedState("blue1", False)
```

```
    setLedState("green1", False)
```

2 Les instructions utiles en Python :

2.1 Programmation :

Python permet de créer des programmes à partir d'instructions.

Par exemple, l'instruction `droite()` peut faire déplacer un robot d'une case vers la droite.

Un programme formé d'instructions les unes en dessous des autres, exécute ces instructions l'une après l'autre.

```
from robot import *

droite()
haut()
droite()
```

Le programme ci-dessus fait déplacer le robot vers la droite, puis vers le haut, puis de nouveau vers la droite.

2.2 Structure de test

2.2.1 L'instruction if :

Avec l'instruction if, on peut exécuter une instruction uniquement dans certaines conditions.

```
if caseMarquee():
    peindre()
```

Par exemple, le programme ci-dessus teste le contenu de la case du robot, et ne la peint que si elle est marquée.

On peut aussi placer plusieurs instructions dans une instruction if, comme illustré ci-dessous :

```
if caseMarquee():
    peindre()
    droite()
```

2.2.2 L'instruction if/else

On peut utiliser une instruction if/else, pour effectuer des opérations différentes selon la situation. Par exemple :

```
if caseMarquee():
    peindre()
else:
    haut()
```

Dans le programme ci-dessus, si la case du robot est marquée, le robot la peint, sinon il ne la peint pas mais se déplace vers le haut.

2.3 Les structures de boucles

2.3.1 Boucle inconditionnelle avec l'instruction `for`

Une boucle inconditionnelle permet de répéter un bloc d'instructions lorsqu'on connaît à l'avance le nombre de répétitions. On peut utiliser l'instruction `for k in range(...)`.

Par exemple, plutôt que de mettre 5 fois la même instruction :

```
droite()
droite()
droite()
droite()
droite()
```

On peut écrire la boucle suivante :

```
for k in range(5):
    droite()
```

On peut aussi mettre plusieurs instructions dans une boucle :

```
for k in range(5):
    droite()
    haut()
```

La situation la plus courante est :

```
Pour k allant de 1 à 10 répéter
    Bloc d'instructions
FinPour
```

Sa traduction en `Python` est :

```
# flux parent
for k in range(1, 11):
    # bloc d'instructions
# retour au flux parent
```

On remarque l'utilisation `range(1, 11)` alors qu'on attendrait `range(1, 10)`.

En fait `range(1, 11)` retourne un *itérateur*, qui va parcourir avec un incrément de 1 tous les entiers n tels que $1 \leqslant n < 11$.

Il faut bien se souvenir que la borne supérieure de `range(n, m)` est exclue mais il est facile de retenir que le nombre de tours de boucles est $m - n$.

Par exemple, pour calculer la somme des tous les entiers consécutifs de 100 à 200, on peut écrire :

```
somme = 0
for k in range(100, 201):
    somme = somme + k
print("La somme est ", somme)
```

S'il s'agit juste de répéter `n` fois un bloc d'instructions on utilise le raccourci `range(n)` au lieu de `range(0, n)` ou de `range(1, n + 1)`.

Par exemple pour dire 100 fois “Merci”, on peut écrire :

```
for k in range(100):
    print("Merci !")
print("Ouf!")
```

2.3.2 La boucle conditionnelle avec l’instruction `while`

Généralement, on souhaite répéter un bloc d’instructions tant qu’une condition est réalisée mais on ne sait pas à l’avance combien de répétitions seront nécéssaires.

La situation la plus courante est :

```
Tant Que Condition répéter
    Bloc d'instructions
Fin Tant Que
```

Elle se traduit en `Python` par :

```
# flux parent
while condition:
    # bloc d'instructions indenté
# retour au flux parent
```

Par exemple, pour demander la saisie d’un login jusqu’à une saisie correcte, on peut écrire :

```
login = 'moi'
while input('Login ?') != login:        # test d'entrée de boucle
    print("Essaie encore !")
print('Bienvenue ', login)    # sortie de boucle
```

Évidemment, on peut aboutir à une *boucle infinie* si la condition d’entrée de boucle n’est jamais vérifiée.

Parfois une boucle infinie est le comportement souhaité, par exemple pour un programme qui doit surveiller des événéments par l’intermédiaire d’écouteurs logiciels (Javascript dans le navigateur) ou de capteurs physiques (de lumière, de température …).

On peut utiliser l’instruction `while True :` pour répéter sans fin une séquence d’instructions, comme dans l’exemple ci-dessous, qui allume et éteint une LED toutes les 500 millisecondes :

```
while True :
    allumerLED()
    attendre(1000)
    eteindreLED()
    attendre(1000)
```

L’instruction `while True :` exécute en boucle les instructions placées en dessous, indentées vers la droite, sans s’arrêter, jusqu’à ce que l’on interrompe l’exécution du programme.

Ce type de boucle est utile pour les programmes qui ne doivent jamais s’arrêter, par exemple le programme d’un système d’alarme, qui doit être actif en permanence.

2.3.3 Boucles imbriquées :

Il est possible d'utiliser des boucles imbriquées, c'est-à-dire que l'on peut mettre des boucles for, à l'intérieur d'autres boucles for.

Par exemple :

```
for i in range(5):
    droite()
    for j in range(3):
        haut()
    droite()
```

Ce programme répétera 5 fois un déplacement d'1 case vers la droite, 3 cases vers le haut et 1 case vers la droite.

3 L'interface FranceIOI :

L'interface comporte deux onglets EXPÉRIMENTER et VALIDER, qui correspondent à deux modes d'utilisation différents.

3.1 Ecrire un programme :

C'est à droite dans l'éditeur que vous allez écrire vos programmes en langage python.

Votre programme doit toujours commencer par la ligne `from quickpi import *`.

Cette ligne de code permet de pouvoir utiliser les fonctions spécialement définies pour le module QuickPi.

Les fonctions et mots clés disponibles sont répertoriés dans des listes déroulantes.

Ensuite, vous écrivez les instructions de votre programme les unes

3.2 Tester un programme :

L'onglet EXPÉRIMENTER est en quelque sotre un "bac à sable". Il vous permet de mener des expérimentations avec les composants et les fonctions disponibles dans l'interface.

Un ou des composants sont mis à votre disposition sur l'interface.

Vous pouvez tester ces composants en cliquant dessus pour changer leur état.

Exemple : cliquer sur une LED pour l'allumer.

Pour exécuter votre programme, cliquez sur le bouton en bas à gauche de l'écran.

N'importe quel programme que vous concevez est exécuté entièrement, avec une simulation visuelle ou auditive du fonctionnement des composants.

3.3 Valider un programme :

Votre mission consiste à concevoir un programme qui permet d'effectuer la tâche demandée dans l'énoncé.

L'onglet VALIDER permet une validation automatique des programmes soumis : grâce à un test, le programme présent dans l'éditeur est automatiquement évalué.

Voici comment fonctionne ce test :

Une ligne du temps (en secondes) est présente pour chaque composant disponible.

Cette ligne du temps apparaît en gris lorsque le composant est activé pendant la période, sinon elle n'apparaît pas.

Au fur et à mesure de l'exécution du programme, un curseur se déplace horizontalement et vérifie que l'état des composants est conforme à ce qui est attendu.

Dans l'exemple, la LED doit être initialement éteinte (état OFF), puis allumée (état ON, ligne grise) entre les instants 0 et 3, puis à nouveau éteinte à l'instant 3.

4 Les fonctions utiles des composants :

4.1 Gestion du temps :

- `sleep(milliseconds)`

  Cette fonction permet de stopper l'exécution du programme pendant une durée entrée en paramètre.

  Cette durée est exprimée en millisecondes.

  Exemple :

  ```
  sleep(1000)
  ```

  Pour stopper l'exécution du programme pendant une durée de 1 seconde.

4.2 Buzzer :

Un buzzer est un composant qui produit un son lorsqu'il est soumis à un tension électrique.

Un buzzer possède deux états :

- ON : le buzzer est soumis à une tension électrique, il sonne.

- OFF : sans tension électrique, le buzzer reste silencieux.

Le son peut être toujours le même ou être paramétrable.

Fonctions disponibles :

- `turnBuzzerOn()`

  Cette fonction permet d'allumer le buzzer.

- `turnBuzzerOff()`

  Cette fonction permet d'éteindre le buzzer.

- `setBuzzerNote(buzzer, frequency)`

  Pour le buzzer entré en paramètre, cette fonction permet de produire un son à une fréquence donnée.

  La fréquence est exprimée en Hertz.

  Exemple :

  ```
  setBuzzerNote("buzzer1", 264)
  ```

  permet de jouer la note DO.

4.3 LEDs ou diodes électroluminescentes :

Une LED est un composant qui émet de la lumière quand il est parcouru par un courant électrique.

Une LED possède deux états :

- ON : le courant traverse la LED, elle est allumée :

- OFF : il n'y a pas de courant, la LED est éteinte : 

Une LED ne laisse passer le courant électrique que dans un seul sens.

On trouve des LEDs qui émettent de la lumière rouge ou de la lumière verte, ou d'autres couleurs encore.

Fonctions disponibles :

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

  Elles ne peuvent servir que lorsqu'il n'y a qu'une seule LED utilisée.

- `setLedState(led,state)` :

  Cette fonction permet d'allumer ou 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.

  Exemple :

  ```
  setLedState("led1",True)
  ```

- `toggleLedState(led)` :

  Cette fonction permet d'inverser l'état de la LED entrée en paramètre sous forme de chaine de caractères.

  Exemple :

  ```
  toggleLedState("led1")
  ```

4.4 Bouton poussoir

Un bouton poussoir est un élément qui possède deux états, relevé et enfoncé. - ON : le bouton est enfoncé :  - OFF : le bouton est relevé : 

Fonctions disponibles :

- `isButtonPressed()`

  Cette fonction renvoie `True` si le bouton est enfoncé, et `False` s'il est relevé.

  Cette fonction est utilisée seulement lorsqu'il n'y a qu'un seul bouton pressoir sur le montage.

- `isButtonPressed(button)`

  Pour le bouton entré en paramètre sous forme de chaine de caractères, cette fonction renvoie `True` si le bouton est enfoncé, et `False` s'il est relevé.

  Exemple :

  ```
  isButtonPressed("button1")
  ```

4.5 Écran :

L'écran de ce module est un petit écran qui permet d'afficher deux lignes de 16 caractères.

Fonctions disponibles

- `displayText(screen, line1, line2)` Cette fonction permet d'afficher deux lignes de texte sur un écran.

  Elle prend en paramètre l'écran à considérer, ainsi qu'une ou deux lignes à afficher, sous forme de chaines de caractères.

  Exemple : avec le code

  ```
  displayText("screen1", "Hello", "World !")
  ```

  on affiche Hello sur la première ligne, et World ! sur la deuxième ligne de l'écran screen1.

4.6 Manette :

Une manette (stick en anglais) est un ensemble de 5 boutons, chacun correspondant à une direction : haut, droite, bas, gauche ou bien centre.

Il s'agit d'un seul composant, mais qui se programme comme 5 boutons différents :

- "stick1.up" pour la direction haut de la manette "stick1"
- "stick1.right" pour la direction droite
- "stick1.down" pour la direction bas

- “stick1.left” pour la direction gauche
- “stick1.center” pour la direction centre

Par exemple pour tester si la direction haut est enfoncée, on utilise :

```
buttonState("stick1.up")
```

4.7 Capteur de distance :

Ce capteur permet de mesurer la distance sans contact grâce à un capteur à ultrasons ou à laser.
Il a une portée de 3 centimètres à 5 mètres.

Fonctions disponibles : - `readDistance(range)`

`Cette fonction renvoie la distance captée par le capteur de distance entré en paramètre. Cette distance est exprimée en centimètres.`

`Exemple :`

```
readDistance("range1")
```

4.8 Servomoteur :

Le Servomoteur est un petit moteur qui peut tourner précisément jusqu’à un angle donné, entre 0 et 180 degrés. On peut l’utiliser pour contrôler la direction des roues d’un petit véhicule, ou pour ouvrir ou fermer une barrière, etc.

Fonctions disponibles :

- `setServoAngle(servo, angle)`

  Cette fonction permet de modifier l’angle du servomoteur choisi. L’angle est exprimés en degrés, entre 0 et 180 degrés.

  Exemple :

  ```
  setServoAngle("servo1", 90)`
  ```

- `getServoAngle(servo)`

  Ce bloc 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.

On peut par exemple l'utiliser pour augmenter l'angle de 1 degré.

Exemple :

```
setServoAngle("servo1", getServoAngle("servo1") + 1)
```

4.9 Crédits

Auteur du document original : **Eric Madec** de l'académie de Rennes.