Introduction à micropython

Pour suivre cette introduction, il est fortement recommandé de lancer l'émulation d'une carte ESP32 tournant sous micropython à l'aide de Wokwi, donc rendez-vous sur wokwi.com et d'ouvrez un nouveau projet (Start from scratch->micropython on ESP32 un peu plus bas sur la page) et lancez la simulation.

Vous devriez voir apparaître en bas à droite une console micropython avec cet affichage :

```
Hello, ESP32!

MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32

Type "help()" for more information.

>>>
```

On peut bien sûr réaliser des calculs numériques avec des entiers, des nombres en virgule flottante, et même des complexes, de manière analogue à python mais avec quelques nuances :

```
>>> from math import sqrt; sqrt(2)**2 - 2
```

et comparez avec python. Commentaire ?

Essayez `3**1000`, `sqrt(-1)`, `(-1)**0.5` (interdit en mathématiques !)

La liste des modules présents s'obtient avec `help('modules')`. Vous verrez que nombre de modules ont leur nom préfixé de u (qu'il faut comprendre comme la lettre μ qui symbolise 'micro'. Assez curieusement, on peut importer ces modules avec ou non la première lettre u, et par exemple, les deux commandes suivantes sont acceptées `import random` comme `import urandom` (mais bien sûr, si vous avez exécuté `import random`, vous accéderez aux fonctions de ce module en les préfixant de `random`, et vous les préfixerez de `urandom` dans le second cas. En un sens, la directive `import random` correspond à ce qu'on aurait pu écrire : `import urandom as random`). A noter aussi la complétion automatique : introduisez `random.` puis utilisez la touche de tabulation et vous verrez les fonctions du module `random` listées.

1 Commandes spécifiques au matériel

Le module `machine` contient des fonctions et classes essentielles au fonctionnement d'un microcontrolleur. Celle qu'on utilisera le plus est la classe `Pin` qui permet d'interfacer le microcontrolleur à des capteurs et autres dispositifs externes, sur les broches du microcontrolleur.

Le « Hello World ! » des microcontrolleurs consiste à faire clignoter une led (pas d'écran a priori !), ce qui est un peu plus investi qu'un `print("Hello World !")` mais pas bien compliqué. Aucun montage n'est en général nécessaire car le plus souvent, un microcontrolleur monté sur carte dispose d'une led. Dans le cas de la carte ESP32 proposée par Wokwi (comme c'est souvent le cas sur ce type de cartes), il y a une led de couleur bleue et qui est connectée entre la masse et la broche portant le numéro 2 (laquelle est également accessible pour des montages, ce qui indique que si on utilise cette broche, elle aura une double action !)

Essayez donc :

```
>>> from machine import Pin
>>> led = Pin(2, Pin.OUT)
>>> led.on() # jetez un oeil !
>>> led.off()
```

Puis :

```
>>> from time import sleep
>>> for i in range(40): # ou toute autre valeur, voire while True: si on préfère (ctrl-c pour
arrêter)
        led.on()
        sleep(0.1)
        led.off()
        sleep(0.5)
```

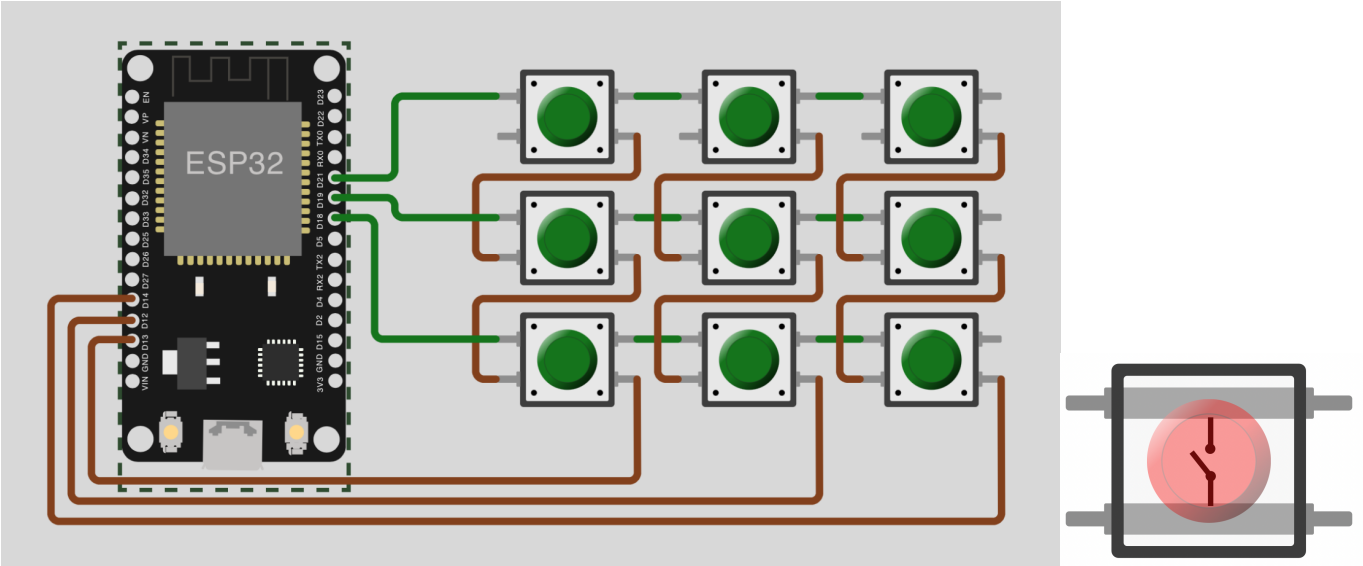
Vous le devinez sans doute ici vu la syntaxe précédente : une broche peut, le plus souvent, servir de point d'entrée ou de sortie du microcontrolleur (même si, on le devine quant aux marquages de la carte, toutes les broches n'ont pas la même fonction. L'une, marquée EN (enable) permet en la reliant à la masse d'éteindre le microcontrolleur (vous aurez remarqué qu'il n'y a pas de bouton on/off, donc relier la carte à une alimentation lance aussitôt la séquence de boot et exécute le programme principal (s'il y en a un). Les broches marquées GND correspondent à la masse (et permet en reliant deux broches de deux dispositifs ainsi nommés de s'assurer qu'ils partagent une même référence). Les broches nommées VP et VN sont là pour des raisons historiques et auraient dû permettre des mesures précises pour des petites tensions, mais cette spécification de l'ESP32 a été annulée. Elles mériteraient donc de s'appeler plutôt D36 et D39.

On a souvent besoin d'avoir sous la main les spécifications et particulièrement le brochage de la carte, et les possibilités de chaque broche. Certaines ne fonctionnent qu'en lecture (34, 35, 36, 39), certaines permettent la lecture d'un signal analogique, (ADC : analog to digital converter) et d'autres n'acceptent qu'un signal digital. Micropython peut certes nous aider de temps en temps :

```
>>> vp = Pin(36, Pin.OUT)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: pin can only be input
```

Mais rien ne remplace une bonne documentation. Il est facile d'en trouver pas mal en ligne. Voici un exemple en français : https://www.upesy.fr/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide

Exemple d'un montage qui fera l'objet d'un exercice :



Les fils verts (en lignes) sont reliés aux broches nommées D21, D19 et D18 et les fils marrons (en colonnes) sont reliés aux broches D13,D12 et D14, et les boutons fonctionnent selon le schéma reproduit ci-dessus. En appuyant sur le bouton en haut à gauche, on ferme le circuit reliant D21 et D13, et en appuyant en bas à droite, on ferme le circuit entre D18 et D14.

Voici un (mauvais) exemple d'utilisation : la fonction suivante est bloquante, jusqu'à ce que soit pressé le bouton du milieu (du moins est-ce le but…) :

```
from machine import Pin
def wait():
    col = [Pin(13, Pin.IN), Pin(12, Pin.IN), Pin(14, Pin.IN)]
    row = [Pin(21, Pin.OUT), Pin(19, Pin.OUT), Pin(18, Pin.OUT)]
    for r in row:
        r.off()
    row[1].on()
    while col[1].value()==0:
        pass
```

Testez ! (Le projet est à l'adresse : https://wokwi.com/projects/347328106194797140) Le but est-il atteint ? Si non, avez-vous une idée de la raison ? (Elle est dans votre cours, cherchez bien !)