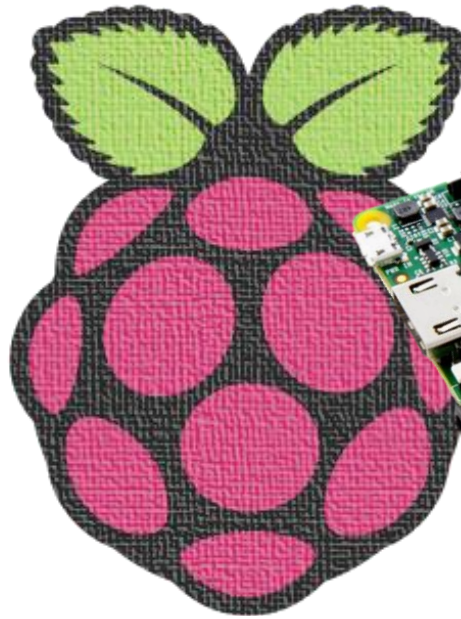


# DÉCOUVERTE DE LA RASPBERRY PI



## TRAVAUX PRATIQUES SÉANCE N°2

## Objectifs de la séance :

- Découvrir un peu plus le langage Python
- Comprendre la notion de classes et d'objets
- Se familiariser avec l'utilisation des broches d'entrée-sortie de la Raspberry

Cette seconde séance va vous laisser davantage d'autonomie. N'oubliez pas de vous référer au mémento ou aux sites proposés dans le TP n°1.

## Récupération des fichiers utiles pour cette séance :

Pour cela, tapez la commande suivante dans le terminal :

```
$ sudo git clone https://github.com/MbottiniIUT/IUT\_IPE\_TP2.git
```

[rem : il n'y a pas d'espace dans ce lien, juste des '\_']

Une fois la tâche effectuée, vous devriez retrouver, en ouvrant un explorateur de fichiers, un dossier 'IUT\_IPE\_TP2' sous '/home/pi'.

## Notion d'objets et de classes :

A travers les exercices en Python du premier TP, vous avez pu aborder :

- L'affichage de texte et de variables
- La saisie d'une donnée utilisateur
- L'utilisation d'une boucle
- Le principe de la gestion des erreurs
- La création de fonctions
- L'utilisation d'une librairie externe (en fait une classe !!)

Toutefois, si ces exercices vous ont permis d'écrire vos premiers programmes, ils ne vous ont pas permis d'imaginer pourquoi Python est un langage si populaire à l'heure actuelle. En effet, ce que vous avez écrit en Python aurait pu être écrit en C mise à part la synthèse vocale.

On va donc essayer, avant d'aller plus loin, de présenter très rapidement ce qui caractérise un langage orienté objet comme le Python : les objets et les classes.

En effet, l'utilisation de classes et d'objets rendent un langage comme Python beaucoup plus puissant que le langage C par exemple.

Encore une fois, ce module n'est qu'une première approche directement par la pratique et ne sera pas suffisant pour bien comprendre la programmation orientée objet. Il faudrait un module beaucoup plus conséquent et un cours associé.

### **Qu'est-ce qu'un objet ?**

Et bien, tout est 'objet' : un signal électrique, une voiture, un compte bancaire, un nombre complexe, un plan de bâtiment, ...

En POO (Programmation Orientée Objet), l'idée est de représenter de manière simplifiée des objets du monde réel (concrets ou abstraits) avec des lignes de code.

Tout objet possède donc des caractéristiques, ce que l'on nomme '**Attributs**' en POO :

- Peugeot 508 : motorisation, boîte de vitesse, couleur, habillage intérieur, ...
- Nombre complexe : partie réelle, partie imaginaire
- Compte bancaire : numéro, détenteur, solde, ...

Dans un programme, toutes les caractéristiques réelles d'un objet ne sont pas forcément utiles. On n'attribuera à l'objet informatique que les caractéristiques utiles.

Mais un objet ne se décrit pas uniquement par ses caractéristiques, il se décrit également par des actions ou des réactions, ce que l'on nomme '**Méthodes**' en POO :

- Peugeot 508 : démarrer, s'arrêter, rouler, ouvrir hayon, régler rétroviseur, ...
- Nombre complexe : déterminer son module, déterminer sa phase, ...
- Compte bancaire : Afficher solde, effectuer un retrait, effectuer un dépôt, virement vers un autre compte, ...

### Et qu'est-ce qu'une classe ?

C'est un abus de langage de dire qu'une classe et un objet sont identiques. La classe est un peu le 'moule' permettant de créer des objets similaires. On dit qu'un objet est une **instance** d'une classe.

L'objet instancié disposera de tous les attributs et les méthodes de la classe, mais chaque objet instancié peut avoir des valeurs d'attributs différents.

Prenons l'exemple de la classe 'compte bancaire', on peut créer autant d'objets 'comptes' que l'on veut :

- Compte numéro 1078654491 – détenteur : Armand Dupont – nature : courant – solde : 1025,87€
- Compte numéro 1079635122 – détenteur : Cécile Nivers – nature : Livret A – solde : 41.023,75€

Quel que soit le compte, il hérite des différentes méthodes fournies par la classe 'compte bancaire' (effectuer un retrait, afficher le solde, ...) mais chaque objet créé depuis cette classe peut avoir des attributs différents.

Généralement, on représente graphiquement une classe de la manière suivante :

Nom de la classe
Attributs
Méthodes()

Nombre complexe
partie_réelle
partie_imaginaire
module()
phase()

Compte bancaire
numéro
détenteur
solde
afficher_solde()
retrait()
dépôt()

Remarque : toute classe d'objet dispose forcément soit d'attributs, soit de méthodes, soit des deux. Mais l'un des deux peut être absent.

Remarque : les méthodes sont associées à des parenthèses '()', car elles s'apparentent à des fonctions. Elles peuvent donc avoir des paramètres si nécessaire.

### Concrètement, comment accède-t-on aux attributs et aux méthodes d'un objet ?

Supposons que l'on crée deux objets 'nombre\_complexe1' et 'nombre\_complexe2' depuis la classe 'nombre\_complexe'. On utilise le '.' Pour accéder à leurs attributs et/ou méthodes.

Exemples :

- nombre\_complexe1.partie\_reelle = 5
- nombre\_complexe1.partie\_imaginaire = 9
- resultat = nombre\_complexe1.module()

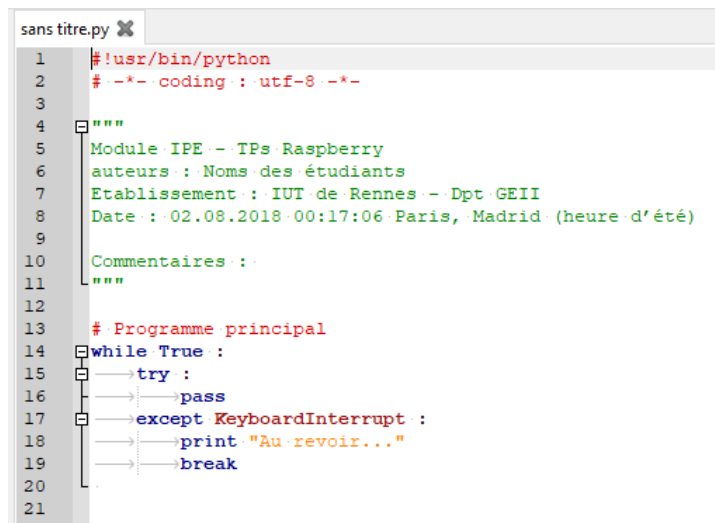
Maintenant que ces notions de base ont été posées, on va pouvoir y regarder de plus près à travers l'écriture de plusieurs programmes.

## EXERCICE 1 : 'Ex01.py'

On va donc maintenant s'intéresser à programmer un jeu de dés en Python et découvrir dans la version finale du jeu comment utiliser les classes en Python.

A partir de maintenant et pour la plupart des programmes que l'on va écrire durant ce module, on va utiliser un autre 'template' dans Geany.

1. Dans 'Geany', faire 'Fichier > Nouveau (selon un modèle) > main\_IPE.py' :



```
1 #!/usr/bin/python
2 # -*- coding : utf-8 -*-
3
4 """
5 Module IPE -- TPs Raspberry
6 auteurs : Noms des étudiants
7 Etablissement : IUT de Rennes -- Dpt GEII
8 Date : 02.08.2018 00:17:06 Paris, Madrid (heure d'été)
9
10 Commentaires :
11 """
12
13 # Programme principal
14 while True :
15     try :
16         pass
17     except KeyboardInterrupt :
18         print "Au revoir..."
19         break
20
21
```

Prenons le temps de décrire les nouveautés apparues dans ce modèle :


- **'while True' :** il s'agit d'une boucle 'while' infinie qui permettra d'exécuter en continu notre programme
- **'try .... Except KeyboardInterrupt' :** Tout comme avant, on utilise ici une structure qui s'exécute tant qu'il n'y a pas d'erreur. Toutefois ici, la seule exception gérée est celle de l'interruption du programme par la combinaison '**CTRL+C**'.
- **'pass' et 'break' :** le 'pass' signifie que l'on ne fait rien de spécial. Il est nécessaire ici car aucun code n'est présent dans le 'try'. Il sera remplacé par votre propre code dans la suite. Le 'break' permet de sortir de la boucle infinie lorsque l'on utilise la combinaison de touches '**CTRL+C**'.

N'oubliez pas de renommer votre fichier '**Ex01.py**' par la commande '**Enregistrez sous...**'

2. Cahier des charges de ce programme de ce premier exercice :
  - a. On génère un nombre aléatoire entre 1 et 6 qui simule le lancer de dé du joueur et on l'affiche (**'votre lancer : x'** où x est le nombre aléatoire)
  - b. On insère une temporisation d'une à deux secondes pour ralentir un peu l'exécution
  - c. On génère un nombre aléatoire entre 1 et 6 qui simule le lancer de dé de la Raspberry et on l'affiche (**'le lancer de la raspberry : y'** où y est le nombre aléatoire)

- d. On compare les deux lancers et l'on affiche qui a gagné (celui qui a la valeur la plus élevée) ou s'il y a match nul
  - e. On demande au joueur s'il veut refaire une manche (réponse par 'o' ou 'n')
3. Comme précisé précédemment, la quasi-totalité de votre code se tiendra dans le **'try'**, à la place du **'pass'**. Tout d'abord, vous allez utiliser plusieurs fois l'instruction **'print'** que vous avez déjà découvert dans le précédent TP. Mais des nouveautés doivent également être intégrées :
- a. Vous allez avoir besoin de générer des nombres aléatoires. Pour cela, vous allez utiliser la classe **'random'** qu'il faut importer dans votre programme (juste après le bloc de commentaires) par la commande **'import random'**
  - b. Cette classe **'random'** dispose de plusieurs méthodes. Celle qui nous intéresse s'appelle **'randint'** et s'utilise ainsi : **'nombre = random.randint(a,b)'** . Consultez la page suivante pour savoir quoi mettre en a et b : <https://docs.python.org/2/library/random.html>
  - c. La temporisation utilise également une classe dédiée, la classe **'time'** que l'on intègre dans le programme via la commande **'import time'** comme toute classe externe. Vous utiliserez alors la méthode **'sleep()'** en vous référant à la page web suivante : <https://docs.python.org/2/library/time.html>
  - d. Vous aurez besoin d'effectuer des tests avec une structure **'if'** : documentation Mémento page 8 ou en ligne <https://docs.python.org/2/tutorial/controlflow.html> ou <https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/231174-les-structures-conditionnelles>
  - e. Pour demander au joueur s'il veut faire une nouvelle manche, il sera plus facile ici d'utiliser la méthode **'raw\_input()'** qui renvoie déjà une chaîne de caractères. La description vous est donnée sur la page suivante : <https://docs.python.org/2/library/functions.html>

ATTENTION, dès lors que vous avez une boucle infinie dans votre programme, celui-ci ne s'arrêtera pas tant que vous ne l'interrompez pas vous-même. Cela peut se faire de deux façons :

- En faisant au clavier la combinaison de touches '**CTRL + C**'
- Ou dans 'Geany' en cliquant sur l'icône 

**Faites contrôler votre programme par un enseignant.**

## EXERCICE 2 : 'Ex02.py'

On va maintenant améliorer quelque peu l'interface graphique de notre jeu de dé en affichant une représentation des dés comme ceci :

```
Jeu de des  
Votre lancer :  
-----  
|   o   |  
|       |  
|       |  
|       |  
-----  
Au tour de la Raspberry...  
Le lancer de la Raspberry :  
-----  
| o  o  |  
|    o  |  
| o  o  |  
|       |  
-----  
Vous perdez !!  
Voulez-vous rejouer (o/n) ?
```

Le cœur de votre programme ne change pas. Les modifications à apporter sont les suivantes :

1. Avant la boucle principale, déclarer ce que l'on appelle en Python un 'dictionnaire' qui contiendra tous les dessins des dés. Pour gagner du temps et parce que cela ne présente aucune difficulté de programmation, un bout de code avec la déclaration d'un tableau de type '**dictionnaire**' avec le dessin des dés vous est fourni via le fichier '**dessins\_de.txt**' dans le dossier '**IUT\_IPE\_TP2**'. Copiez-le.

Remarque : En Python, il y a plusieurs formes de structuration de données de type 'tableaux' :

- Les **listes** (repérables par '**[]**') : elle se rapproche de la notion de 'structure' en C. Une liste peut mixer des éléments de type hétérogène (entier, chaîne de caractères, objet complexe, ...) mais en Python sa taille n'est pas figée. Elle peut s'agrandir ou se réduire au cours du programme. La position (l'index) est importante et permet d'accéder à un élément particulier.
  - Les **tuples** (repérables par '**()**') : ce sont en fait des listes non modifiables (ni leur taille, ni leur contenu). Exemple : si une fonction renvoie plusieurs éléments, elle le fera sous forme d'un tuple. Ici aussi, c'est l'index qui donne accès à un élément.
  - Les **dictionnaires** (repérables par '**{}**') : Ici, chaque élément peut être aussi de type différent, mais il est toujours précédé d'une '**clef**' qui permettra d'y accéder. L'ordre des éléments n'a donc aucune importance puisqu'ils sont identifiés par leurs clefs. La clef est toujours entre guillemets " " suivie de ':' et de la valeur stockée. Pour afficher la valeur associée à une clef, il faut écrire '**print dico[clef]**'.
2. Après chaque lancer (utilisateur et Raspberry), vous n'afficherez plus la valeur du dé, mais son dessin via le dictionnaire ainsi défini. Attention ici, la valeur associée à chaque clef est une liste, il faut donc une boucle pour afficher chaque chaîne de caractères de la liste.

Au besoin, vous pouvez vous aider des deux pages web suivantes :

<https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python/232273-les-dictionnaires>

<http://apprendre-python.com/page-apprendre-dictionnaire-python>

**Faites contrôler votre programme par un enseignant.**

## EXERCICE 3 : 'Ex03.py'

On a commencé dans le TP précédent à structurer nos programmes en utilisant des fonctions. On l'a évoqué, au cours de ce TP, on va aller plus loin en élaborant des classes d'objet.

Toutefois, cette étape de création de classe passe naturellement par l'utilisation de fonctions.

Modifier donc le programme précédent en mettant une partie de votre code dans deux fonctions :

- La fonction '**Lancer\_de()**' : elle ne reçoit aucun paramètre et se contente de renvoyer un chiffre aléatoire entre 1 et 6.
- La fonction '**Affichage\_de(valeur)**' : elle reçoit un paramètre qui correspond à la valeur du dé (obtenu par la fonction précédente) à la suite du lancer et se contente d'afficher graphiquement la face du dé à l'aide du dictionnaire vu dans l'exercice précédent.



## EXERCICE 4 : 'Ex04.py'

On va donc maintenant créer une classe pour notre objet 'dé à jouer'. Cet objet a comme attribut sa valeur et comme méthodes associées dans notre cas : son lancer et son affichage (d'où la création des deux fonctions de l'exercice précédent).

La création de classe sans explication (sous forme de cours ou via le web) n'est pas chose aisée à comprendre. Aussi, étant donné que ces séances de TP sont une initiation à la POO, le code de la définition de la classe de cet exercice vous est fourni via le fichier 'Ex04\_incomplet.py' :

```
class De_a_jouer :# Rem : un nom de classe commence par une majuscule
                  # contrairement a une variable

    # La methode __init__ permet de definir
    # les attributs de l'objet lors de son instantiation
    def __init__(self) :
        self.valeur = '1'

    # La methode 'lancer_de' simule le lancer de de
    def lancer_de(self) :
        self.valeur = random.randint(1,6)
        return self.valeur

    # La methode 'affichage_de' affiche dans le terminal la face du de
    def affichage_de(self, valeur='1') :
        for ligne in range(len(Valeurs_de['1'])):
            print(Valeurs_de[str(self.valeur)][ligne])
```

On retrouve les deux fonctions 'lancer\_de()' et 'affichage\_de(valeur)' demandées précédemment. Pour le reste quelques explications s'imposent :

- 'class' : c'est le mot clé dédié pour la création d'une classe d'objets. Ce mot est toujours suivi du nom de la classe
- '\_\_init\_\_' : cette méthode est obligatoire. Elle n'a généralement pas de paramètres et sert simplement à définir les attributs de l'objet créé. Dès qu'un objet est instancié, cette méthode est automatiquement exécutée.
- **Mais à quoi sert ce 'self' que l'on voit partout ?** Il fait justement référence à l'objet créé depuis cette classe.

Complétez ce programme 'Ex04\_incomplet.py' :

- Créer deux objets 'dé' héritant de la classe 'De\_a\_jouer' :
  - `de_joueur = De_a_jouer()`
  - `de_raspberry = De_a_jouer()`
- Reprenez la trame de votre programme 'Ex03.py' en utilisant maintenant la notation des attributs et des méthodes de chaque objet. Exemple : au lieu d'écrire '`valeur_de=Lancer_de()`' comme précédemment, on écrira '`de_joueur.lancer_de()`' ou '`de_raspberry.lancer_de()`' suivant le cas.

**Faites contrôler votre programme par un enseignant.**

## EXERCICE 5 : 'Ex05.py'

Lorsque l'on crée une classe d'objets, c'est un peu comme une librairie en C, l'intérêt est de l'externaliser afin de pouvoir l'utiliser à loisir dans différents programmes simplement en l'important.

Voyons comment externaliser la classe d'objets 'De\_a\_jouer' :

- Créez sous 'Geany' un fichier vide. Enregistrez-le avec le nom '**De.py**'
- Insérez dans ce fichier :
  - Les lignes habituelles de commentaires de début de fichier
  - L'importation de la classe '**random**'
  - La déclaration du dictionnaire '**Valeurs\_de**'
  - Votre classe '**De\_a\_jouer**'
- Vous pouvez alors enregistrer et fermer ce fichier (assurez-vous tout de même de le mettre dans le dossier '**IUT\_IPE\_TP2**' comme vos autres programmes)

On va maintenant créer le fichier '**Ex05.py**' en repartant du fichier '**Ex04.py**' :

- Effacez bien sûr tout ce que vous avez copié dans le fichier **'De.py'**
- Commencez par importer votre classe **'De\_a\_jouer'** depuis le fichier **'De.py'**. Pour cela il vous suffit d'écrire : **from De import De\_a\_jouer**  
Pourquoi l'on ne peut pas se contenter d'écrire **'import De'** ? On pourrait le faire, mais certains fichiers comportent beaucoup de classes différentes (parfois imbriquées) et importer tout le fichier importerait toutes ces classes. Il est donc de bonne habitude d'importer que la ou les classes utiles pour notre programme.
- Une fois la classe importée, on crée/instancie des objets de cette classe simplement par les instructions suivantes (comme on a déjà pu le faire dans l'exercice n°4) :

```
Objet1 = De_a_jouer()
```

Il n'y a rien dans les parenthèses car la méthode '\_\_init\_\_()' de la classe '**De\_a\_jouer**' n'a pas de paramètre.

- Tout l'intérêt d'une classe est de pouvoir créer autant d'objet que l'on veut à partir de cette classe. On va donc modifier le programme précédent en jouant maintenant avec deux dés. La combinaison de dés la plus élevée reste la combinaison gagnante. L'écran de la console doit ressembler à ceci :

```

Jeu de des
Votre lancer :
-----
| o |
|   |
|   o |
|   |
-----
| o o |
|   o |
| o o |
|   |
-----
Au tour de la Raspberry...
Le lancer de la Raspberry :
-----
| o o |
|   o |
| o o |
|   |
-----
| o o |
|   o |
| o o |
|   |
-----
Vous perdez !!
Voulez-vous rejouer (o/n) ?

```

Remarque : Il va donc falloir créer ici 4 objets 'dé' à partir de votre classe.

**Faites contrôler votre programme par un enseignant.**



## EXERCICE 6 : 'Ex06.py'

Cet exercice n'en est pas vraiment un. On va en fait tester une ultime version de notre jeu de dés mais en ayant une interface graphique.

C'est un des grands intérêts de la Raspberry. Elle dispose d'une sortie vidéo, elle peut donc afficher des IHMs et interagir avec l'utilisateur.

Il existe un grand nombre de bibliothèques et d'outils plus ou moins complets pour créer des interfaces graphiques sur Raspberry : gtk, Qt, Tkinter, WxPython, pyglet, pygubu, Remi, Easygui, guizero, pygame zero,...

Ce large choix est plutôt déconcertant lorsque l'on part de zéro !! Lequel choisir ?

Même si certains d'entre eux se veulent plus simples, les aborder dans ce module est un peu ambitieux si l'on veut pouvoir s'intéresser également aux GPIO de la Raspberry, à l'aspect Client-serveur, ...

J'ai donc préparé un programme qui se base sur la classe que vous avez précédemment créée mais qui ajoute une interface graphique. Le choix s'est porté ici sur '**guizero**' qui est une version simplifiée de Tkinter et qui est également régulièrement mis à jour (ce qui n'est pas toujours le cas selon le package).

On a donc besoin d'installer cette bibliothèque '**guizero**' via un terminal :

```
$ sudo pip3 install guizero
```

[remarque : c'est bien '**pip3**' et non '**pip**' ici car cette bibliothèque n'existe que pour Python 3 !!]

Toujours dans le terminal, placez vous dans le répertoire '**IUT\_IPE\_TP2**' et tapez la commande suivante pour lancer le programme :

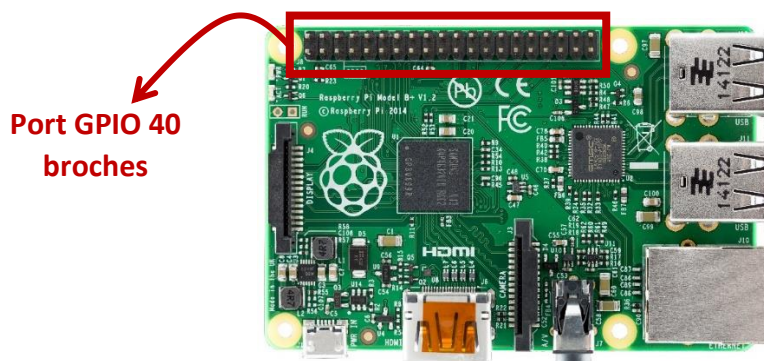
```
$ sudo python3 Ex06.py
```

[remarque : là aussi, c'est bien '**python3**' et non '**python**' pour les mêmes raisons que précédemment]

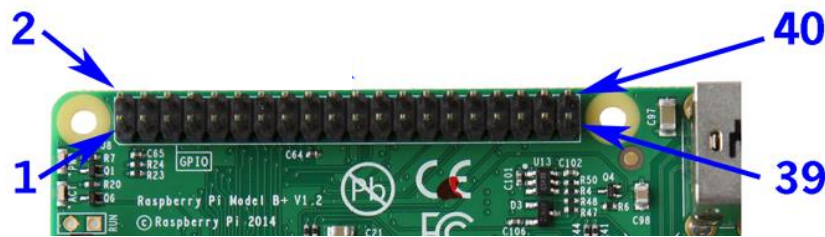
Quel est l'intérêt de cet exercice ? Et bien simplement que vous consultiez le fichier '**Ex06.py**' afin d'en voir rapidement la structure et de constater le faible nombre de lignes de code nécessaires pour la création d'une telle interface !!

## Présentation du connecteur GPIO

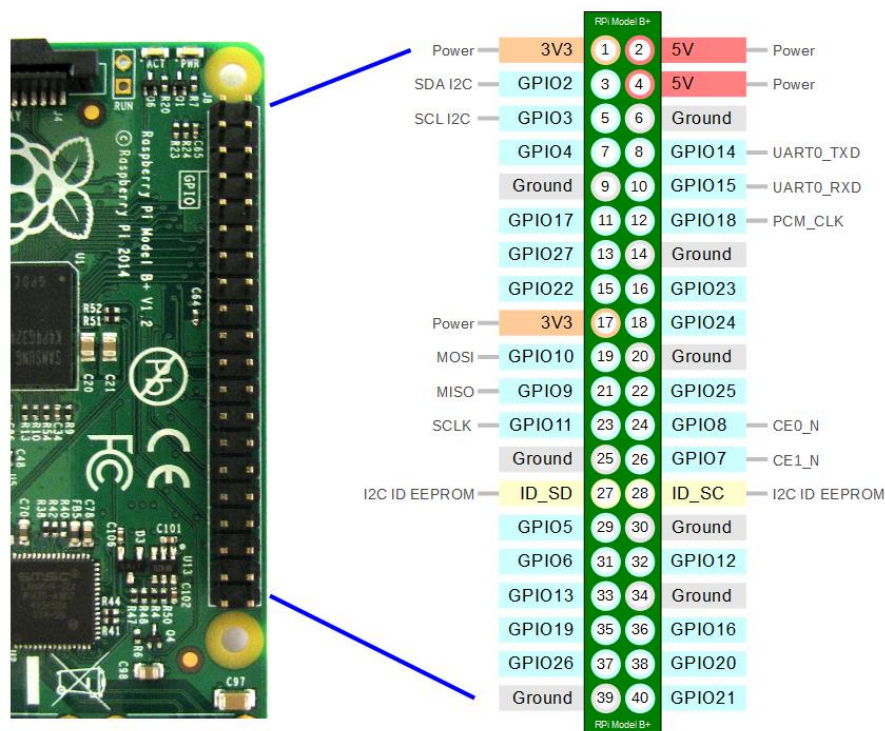
La carte Raspberry Pi (modèle B+) dispose d'un connecteur mâle de deux rangées de 20 broches appelé communément 'port GPIO' (General Purpose Input Output) :



Ces broches sont numérotées de la manière suivante :



Bien évidemment, toutes ces broches n'ont pas la même fonction. Le graphique ci-dessous vous donne leurs dénominations :



On voit donc bien qu'il n'y a pas que des broches d'entrée-sortie :

- 2 broches d'alimentation **3,3V** : ce sont des sorties permettant de fournir une tension régulée de 3,3V vers l'extérieur, mais attention, le courant est limité à 50mA.
- 2 broches d'alimentation **5V** : ce sont également des sorties mais sous 5V. Elles sont directement reliées au connecteur micro-USB de la Raspberry utilisé pour son alimentation. La Raspberry consommant au maximum 600-700mA, si vous utilisez un bloc secteur 5V/2A, vous disposerez de 1300 à 1400mA sur ces deux broches.
- 8 broches de **masse** (Ground).
- 2 broches **ID\_SD** (27) et **ID\_SC** (28) réservées uniquement à la communication avec une EEPROM I<sup>2</sup>C (type 24Cxx) installée sur une carte d'extension et qui contiendrait la configuration des I/O propre à cette carte.
- Il reste donc 26 broches I/O. Toutes ces broches peuvent être utilisées indifféremment en entrée numérique ou en sortie numérique. Certaines d'entre elles peuvent être configurées avec un usage particulier :
  - Les 2 broches n°8 et 10 peuvent être configurées en liaison série, respectivement en **TX** et **RX**.
  - Les 2 broches n°3 et 5 peuvent être configurées en liaison I<sup>2</sup>C, respectivement **SDA** et **SCL**.

- Les 5 broches 19, 21, 23, 24 et 26 peuvent être configurées en liaison SPI (**SPI0**), respectivement **MOSI**, **MISO**, **SCLK**, **CE0** et **CE1**, permettant ainsi de piloter deux esclaves SPI sur un même bus SPI.
- Il y a une seconde interface SPI (**SPI1**) mais qui nécessite un peu de configuration pour pouvoir être utilisée.

Chaque broche I/O dispose de deux numéros pour l'identifier : un numéro correspondant à la broche sur le connecteur (**BOARD**) et un numéro correspondant au numéro de broche du microprocesseur embarqué (**BCM**).

Exemple : **GPIO9** → broche '**BOARD**' = **21** / broche '**BCM**' = **9**

Il est important aussi d'avoir à l'esprit que la carte Raspberry Pi ne dispose pas d'entrées analogiques (ce qui est bien regrettable...) !! Il existe bien entendu sur le marché des convertisseurs analogique-numérique pilotables par une liaison I<sup>2</sup>C ou par une liaison SPI.

**REMARQUE TRES IMPORTANTE** : *Les broches GPIO sont prévues pour une utilisation avec des niveaux de tension 0V-3,3V. Elles ne sont pas protégées !! Appliquer des tensions plus élevées détériorerait de manière irréversible la Raspberry !!*

*Il n'y a pas non plus de protection contre les surintensités (économie oblige !!). Or au reset, les broches d'entrée-sortie sont configurées pour délivrer un courant max de 8mA. Conclusion : sous 3,3V, une résistance de charge inférieure à 400Ω créerait une surintensité !!*

## EXERCICE 7 : 'Ex07.py'

On va au cours des prochains exercices utiliser des LEDs et des boutons poussoirs afin de découvrir quelques exemples simples d'utilisation du connecteur GPIO.

Pour cela, on va également utiliser une librairie Python permettant la gestion de ces ports d'entrée-sortie : **GPIO zero**

Le but ici est de découvrir quelques fonctionnalités de cette librairie.

Vous pourrez utiliser le document '**Memento**' mis à votre disposition en salle (à partir de la page 14) ou au format numérique dans le dossier '**IPE\_IUT\_TP1/Documentation**'. Vous trouverez également dans ce dossier le PDF de la documentation complète de cette librairie.

Mais vous pouvez aussi vous aider de pages web avec des exemples dont il vous suffira de vous inspirer pour écrire votre propre code. ATTENTION toutefois... Vous allez certainement tomber sur des pages utilisant la librairie '**RPi.GPIO**'.

Certes, elle a été très utilisée et ce ne serait pas dramatique de l'utiliser encore aujourd'hui. Mais elle est plutôt ancienne et globalement obsolète. A sa place est donc apparue la librairie '**GPIO zero**' qui est une classe pour la gestion du port GPIO de niveau beaucoup plus élevé. Veillez donc à faire vos recherches web en précisant '**raspberrypi python gpio zero**' !!

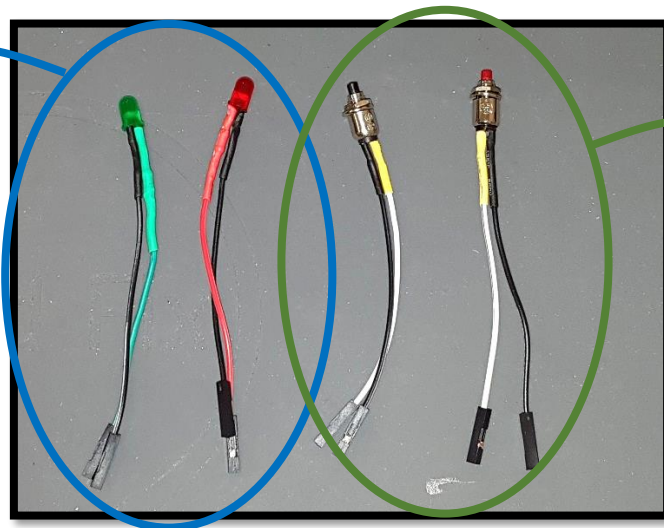
Ces librairies sont normalement déjà installées avec l'OS, mais il est toujours bon de les tenir à jour au cas où. Commencez donc par taper la commande suivante dans un terminal :

**\$ sudo apt install python-gpiozero**

On va maintenant découvrir le matériel associé. Vous disposez de deux LEDs et de deux boutons poussoirs tels que ceux sur la photo suivante :

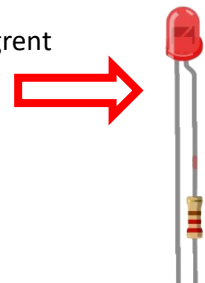
Les deux LEDs :

- Une rouge
- Une verte

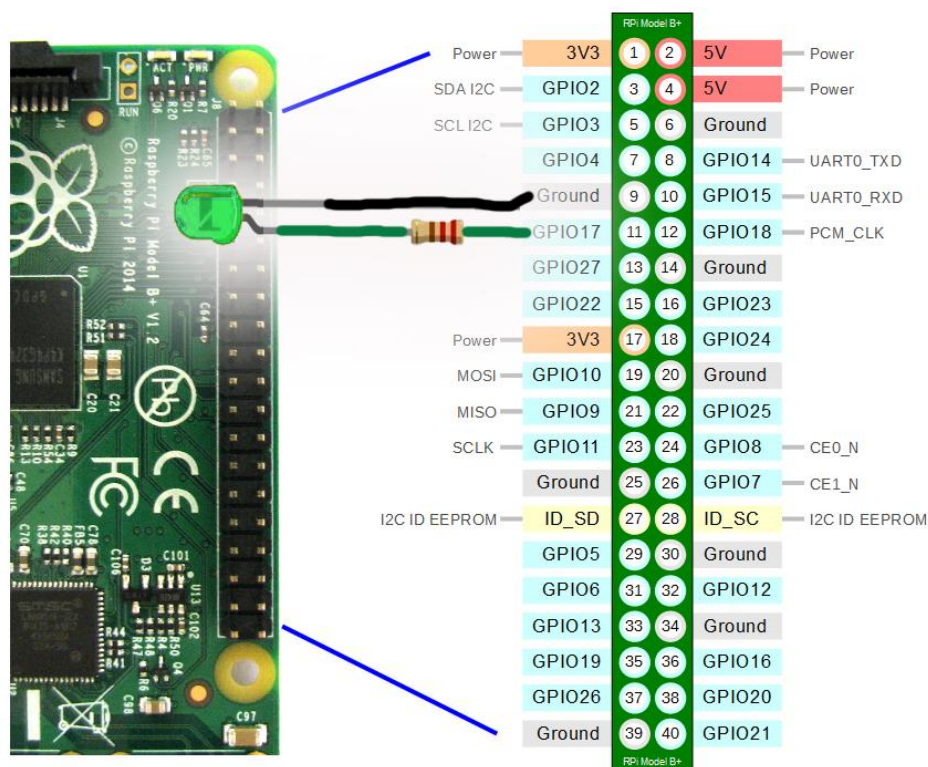


Les deux boutons poussoirs

Remarque : **INFORMATION IMPORTANTE !!** Les LEDs **fournies** intègrent déjà une résistance de protection. Elles peuvent donc être directement branchées à une broche 3.3V de la Raspberry sans risquer d'être détruite.



Le but de ce premier exercice est faire clignoter une LED. On va ici brancher la LED entre la broche GPIO17 (pour le câble de couleur) et une masse (pour le câble noir) :

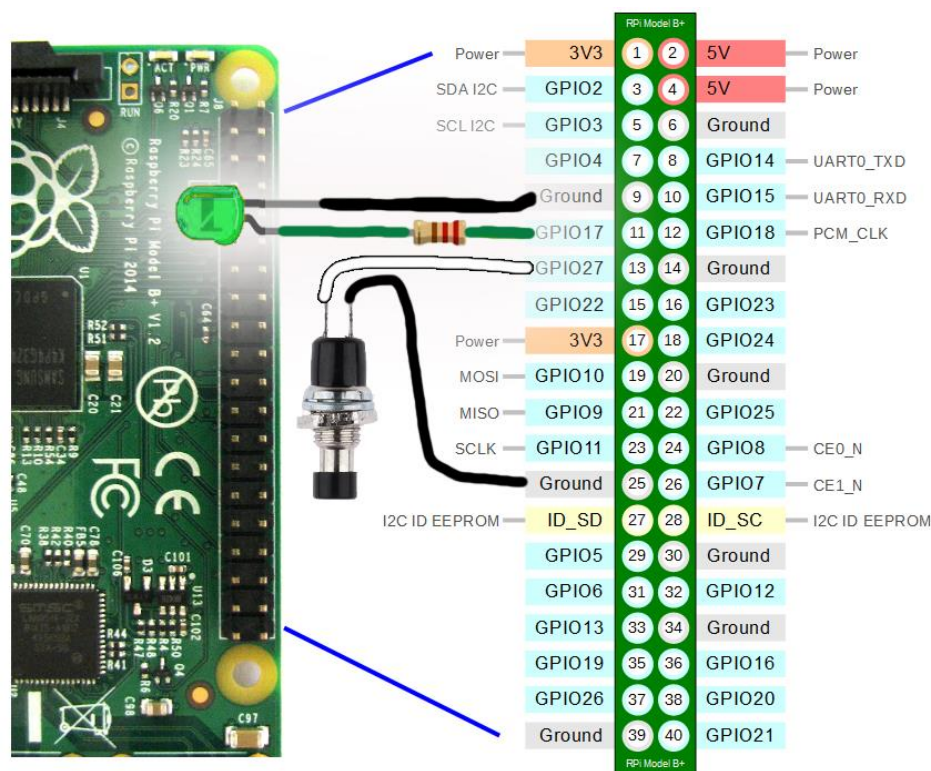




- Créez un programme vide à partir du template **'main\_IPE.py'** sous Geany.
- Utilisez les ressources proposées pour compléter votre programme de manière à faire clignoter la LED chaque seconde (méthode **'sleep'** de la librairie **'time'**)
- Pensez à éteindre la LED dans l'exception avant le **'break'**.

## EXERCICE 8 : 'Ex08.py'

On va maintenant ajouter un bouton dans cet exercice. Celui-ci sera câblé entre la broche GPIO27 et une masse :



On veut maintenant que la LED change d'état à chaque appui sur le bouton poussoir :

- Utilisez les ressources disponibles (mémento/documentation gpio zero/ web) pour compléter le programme précédent avec la LED en y intégrant maintenant la gestion par le bouton poussoir via la classe **'Button'**.
- Attention toutefois, le bouton étant relié à la masse, il est actif sur un niveau '0' et il lui faut une résistance de pull-up pour qu'il fournisse un niveau '1' au repos.

**Faites contrôler votre programme par un enseignant.**

## EXERCICE 09 : 'Ex09.py'

Dans cet exercice, on va réaliser un jeu qui teste les réflexes de chacun.

En termes de câblage, vous allez devoir ajouter un autre bouton et la seconde LED. Je vous laisse choisir les broches correspondantes pour les câbler.

Le cahier des charges de ce jeu de réflexes à deux joueurs est le suivant :

- Au début, la LED rouge s'allume.
- Au bout d'un temps aléatoire compris entre 2 et 12 secondes, elle s'éteint et instantanément, c'est la LED verte qui s'allume.
- A partir de cet instant, le premier joueur à appuyer sur son bouton gagne la partie.
- Le jeu se déroule en 5 manches.
- Des informations doivent être affichées dans le terminal, au moins le nom du joueur gagnant la manche.

Je ne vous fournis pas beaucoup d'information afin de vous laisser un peu plus d'autonomie et que vous trouviez votre propre solution sans être expert en python !!

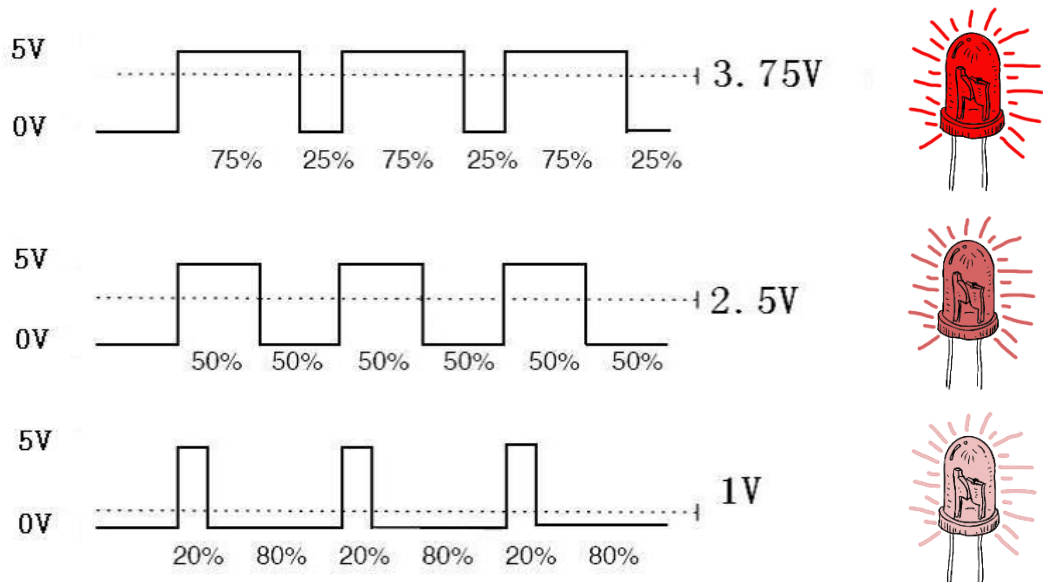
**Faites contrôler votre programme par un enseignant.**

## EXERCICE 10 : 'Ex10.py'

Allumer ou éteindre un éclairage (ici une simple LED) est intéressant, mais pouvoir en faire varier sa luminosité est beaucoup plus intéressant encore. On va ici utiliser les deux boutons disponibles, l'un pour augmenter la luminosité de la LED, l'autre pour la diminuer.

Pour cela, c'est comme pour faire varier la vitesse d'un moteur, on utilise un signal PWM. La librairie '**GPIO zero**' permet de générer un signal PWM sur n'importe quelle broche disponible du connecteur de la Raspberry.

**Rappel du principe de fonctionnement d'un signal PWM (sous 5V, mais le principe est le même sous 3,3V) :**



- Ne modifiez pas votre câblage, c'est inutile.
- Vous allez devoir utiliser ici la classe d'objet '**PWMLED**'. Vous trouverez comme d'habitude des informations et des exemples dans le '**Mémento**' ou sur le web.
- En instanciant un objet '**PWMLED**', vous fixerez son niveau initial à '0' et sa fréquence à 150Hz.
- Une variable '**luminosite**' va correspondre finalement à la valeur du rapport cyclique du signal PWM (comprise entre 0.0 et 1.0). Vous pourrez le régler via l'attribut '**value**' de la classe '**PWMLED**'.



- Vous utiliserez l'attribut **'is\_pressed'** de la classe **'BUTTON'** pour tester les deux boutons. La variable luminosité sera alors incrémentée ou décrémentée d'un pas à condition de ne pas dépasser la plage accessible du rapport cyclique (0.0-1.0).

**Faites contrôler votre programme par un enseignant.**

Profitez-en pour lui expliquer pourquoi, selon vous, malgré un pas linéaire dans votre programme, la luminosité réelle de la LED ne semble pas varier de manière linéaire.

## EXERCICE 11 : 'Ex11.py'

Pour aller un peu plus loin avec cette librairie, on va utiliser une fonctionnalité essentielle de la Raspberry : sa connexion à un réseau (local ou internet).

Vous allez donc ici créer un programme python qui vous permet à partir d'un de vos boutons de piloter une des LEDs de la Raspberry du binôme voisin.

Avec la puissance du langage python associée à cette librairie de haut niveau, vous verrez que peu de lignes de code sont nécessaires...

- Il vous faudra bien sûr utiliser l'adresse IP de la Raspberry du binôme voisin.
- Je vous conseille simplement de lire et d'exploiter le contenu de la page suivante pour écrire votre programme : [https://gpiozero.readthedocs.io/en/stable/recipes\\_remote\\_gpio.html](https://gpiozero.readthedocs.io/en/stable/recipes_remote_gpio.html)

*Remarque :* il aurait été prématuré de vous en parler dans le premier TP, mais lors de la configuration de la Raspberry, vous avez validé une interface appelée 'Remote GPIO'. Sans cette activation, vous ne pourriez faire fonctionner cet exercice.

**Faites contrôler votre programme par un enseignant.**

## EXERCICE 12 : 'Ex12.py' (optionnel)

S'il vous reste un peu de temps, essayez de mettre en œuvre ce petit exercice. La Raspberry dispose de deux connexions sans fil : le Wifi (que l'on n'utilise pas ici puisque l'on est en réseau filaire) et le Bluetooth.

On va utiliser le module Bluetooth de la Raspberry. Vous allez vous connecter à la Raspberry avec votre Smartphone et piloter une des deux LEDs via une application dédiée.

Là encore, vous allez mesurer la puissance d'un langage de haut niveau car le code ne comportera que quelques lignes...

- Vous devez télécharger et installer l'application **'BlueDot'** (de Martin O'Hanlon) sur votre téléphone depuis le PlayStore.
- Vous aurez également besoin d'installer deux librairies python via un terminal :  
`$ sudo apt-get install python-dbus`  
`$ sudo pip install bluedot`

- Pour appairer la Raspberry avec votre téléphone, suivez les indications du site suivant : <https://therobotacademy.com/feed/bluedot-bluetooth-remote-raspberry-pi> ou encore <https://bluedot.readthedocs.io/en/latest/pairpiandroid.html>
- Pour le programme, inspirez vous fortement de la page web suivante : [https://gpiozero.readthedocs.io/en/stable/recipes\\_advanced.html#bluedot-led](https://gpiozero.readthedocs.io/en/stable/recipes_advanced.html#bluedot-led)