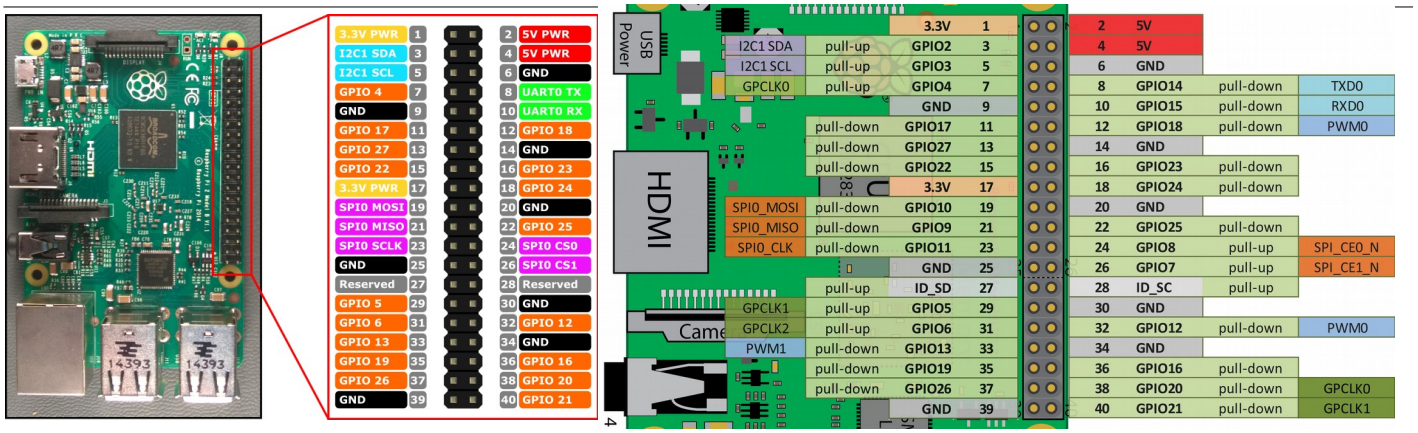
Cette étude porte sur la mise en pratique de développement sur la gestion des entrées-sorties sur la **Raspberry** (avec le système d'exploitation « RASPBIAN »). Ces entrées-sorties sont de type tout-ou-rien et sont gérées par le système **GPIO**.

Les ports GPIO (General Purpose Input/Output, littéralement Entrée/Sortie pour un Usage Général) sont des ports d'entrée/sortie très utilisés dans le monde des microcontrôleurs, en particulier dans le domaine de l'électronique et de l'informatique embarquée.

Un connecteur GPIO offre à une carte électronique la possibilité de communiquer avec d'autres circuits électroniques. Un connecteur GPIO est généralement alimenté en 3.3Vcc et ne peut fournir que des courants de faible intensité, allant de 3mA à 50mA.

CONNECTEUR GPIO DE LA RASPBERRY



Ce serait une erreur de penser que chaque port GPIO est utilisable, car nombre d'entre eux sont réservés à d'autres tâches que de servir d'entrée sorties pour nous. En réalité, seuls quelques ports sur les 40 sont vraiment disponibles. Les broches 4, 17, 27, 22, 5, 6, 13, 19 , 26, 18, 23, 24, 25, 12, 16, 20 et 21 sont les ports que nous pouvons utiliser sans soucis.

SYSTÈME DE FICHIERS ASSOCIÉS AUX CONNECTEURS GPIO

Sous Linux, d'un point de vue général, la communication avec des systèmes physiques se fait systématiquement au travers de fichiers spécifiques. Le **GPIO** n'échappe pas à cette règle de base. Le répertoire de base se situe à l'endroit suivant : « **/sys/class/gpio** ». La gestion complète d'un broche du **GPIO** se fait en trois phases :

- ***L'activation de la broche choisie*** *: se fait au travers du fichier «* ***export*** *».*
- ***Choisir le mode de fonctionnement en entrée ou en sortie*** *: se fait au travers du fichier «* ***direction*** *».*
- ***Utilisation de la broche*** *: se fait au travers du fichier «* ***value*** *».*

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
pi@raspberrypi ~ $ ls /sys/class/gpio
export  gpiochip0  unexport
pi@raspberrypi ~ $ echo 15 > /sys/class/gpio/export
pi@raspberrypi ~ $ ls /sys/class/gpio
export  gpio15  gpiochip0  unexport
pi@raspberrypi ~ $ cd /sys/class/gpio/gpio15
pi@raspberrypi /sys/class/gpio/gpio15 $ ls
active_low  device  direction  edge  power  subsystem  uevent  value
pi@raspberrypi /sys/class/gpio/gpio15 $ cat direction
in          <- entrée par défaut
pi@raspberrypi /sys/class/gpio/gpio15 $ cat value
1           <- l'entrée est validée
pi@raspberrypi /sys/class/gpio/gpio15 $ echo out > direction
pi@raspberrypi /sys/class/gpio/gpio15 $ cat direction
out         <- la broche 15 est maintenant en mode sortie
pi@raspberrypi /sys/class/gpio/gpio15 $ echo 1 > value   <- active la sortie
```

Lorsque nous consultons le contenu du répertoire dédié aux fichiers de gestion du connecteur GPIO, nous remarquons la présence de plusieurs fichiers. D'une part, les fichiers export et unexport dont le but est respectivement d'activer ou de désactiver les broches choisies. D'autre part, les fichiers correspondant aux broches du GPIO déjà activées (ici uniquement « gpiochip0 »).

Depuis le shell du système linux embarqué, nous pouvons activer une broche très simplement grâce à la commande suivante :

```
$ echo 15 > export
```

Cette instruction provoque la création d'un nouveau répertoire nommé « gpio15 » et qui représente la broche choisie du connecteur.

Dans ce répertoire un certain nombre de fichiers ont également été générés automatiquement, dont deux sont particulièrement importants pour l'utilisation de la broche concernée, « direction » et « value ».

Le premier permet de spécifier si nous désirons avoir une entrée (mode par défaut) ou une sortie. Si vous désirez que la broche choisie devienne une sortie, vous devez placez la valeur « out » dans ce fichier « direction » : $ echo out > direction

Pour consulter le contenu ou pour imposer une valeur spécifique (en mode sortie) à cette broche, vous devez utiliser le fichier « value » avec une valeur « 0 » ou « 1 ».

```
$ cat value // consultation de l'état actuel de la broche
$ echo 1 > value // proposer une nouvelle valeur (0 désactivation ou 1 activation)
```

GESTION DES RÉPERTOIRES ET DES FICHIERS DANS PYTHON

Vu que la gestion du **GPIO** peut se faire au travers de la manipulation de fichiers spécifiques, nous allons travailler de la même façon à l'aide du langage Python afin de pouvoir créer des classes de haut niveau qui encapsulent automatiquement toutes ces fonctionnalités là. Il existe le module **« os.path »** (chemin) qui possède un ensemble de fonctions qui implémentent toutes les différentes notions de répertoires. Plutôt qu'un long discourt, voici un exemple évocateur :

principal.py

```
from os import path

dossier = "/home/manu/Documents"
fichier = "/home/manu/Documents/bienvenue.txt"

print('Chemin complet : ', path.abspath(fichier))
print('Fichier séparé : ', path.basename(fichier))
print('Dossier parent : ', path.dirname(fichier))
print('Teste si le fichier existe : ', path.exists(fichier))
print('Taille du fichier : ', path.getsize(fichier), 'octets')
print("Vérifie s'il s'agit d'un dossier : ", path.isdir(dossier))
print("Vérifie s'il s'agit d'un fichier : ", path.isfile(fichier))
print('Associe un fichier à son dossier : ', path.join(dossier, "bienvenue.txt"))
print('Sépare le fichier de son dossier (tuple) : ', path.split(fichier))
```

Shell Python : [évaluer principal.py]

```
Chemin complet :  /home/manu/Documents/bienvenue.txt
Fichier séparé :  bienvenue.txt
Dossier parent :  /home/manu/Documents
Teste si le fichier existe :  True
Taille du fichier :  30 octets
Vérifie s'il s'agit d'un dossier :  True
Vérifie s'il s'agit d'un fichier :  True
Associe un fichier à son dossier :  /home/manu/Documents/bienvenue.txt
Sépare le fichier de son dossier (tuple) :  ('/home/manu/Documents', 'bienvenue.txt')
```

*Comme son nom l'indique, **« path »** prévoit des manipulations propres à la notion de chemin. Il est possible de manipuler uniquement les fichiers. Dans ce cadre là, il existe beaucoup de fonctions toutes faites. Je vous en propose quelques unes bien utiles pour la plupart des situations. Elles sont toutes intégrées dans le module **« os »**.*

principal.py

```
import os

os.chdir('/home/manu/Documents/Python')      # change le répertoire courant (par défaut)
dossier = os.getcwd()                        # renvoie le nom du répertoire courant
print('Dossier en cours : ', dossier)
fichiers = os.listdir()                      # donne la liste des fichiers dans le répertoire courant
print('Liste des fichiers : ', fichiers)
os.mkdir('Stockage')                         # création d'un nouveau répertoire (dans le rep. courant)
print('Liste :', os.listdir())
os.rename('Stockage', 'stockage')            # renomme un répertoire ou un fichier
print('Liste :', os.listdir())
os.removedirs('stockage')                    # supprime un répertoire
print('Liste :', os.listdir())
os.remove('complexe.py')                     # supprime un fichier
print('Liste :', os.listdir())
```

Shell Python : [évaluer principal.py]

```
Dossier en cours :  /home/manu/Documents/Python
Liste des fichiers :  ['principal.py', 'complexe.py', 'personne.py', 'eleve.py']
Liste : ['principal.py', 'Stockage', 'complexe.py', 'personne.py', 'eleve.py']
Liste : ['stockage', 'principal.py', 'complexe.py', 'personne.py', 'eleve.py']
Liste : ['principal.py', 'complexe.py', 'personne.py', 'eleve.py']
Liste : ['principal.py', 'personne.py', 'eleve.py']
```

LECTURE D'UN FICHIER TEXTE

Maintenant que nous connaissons comment faire pour manipuler des dossiers et des fichiers, je vous propose maintenant de pouvoir consulter le contenu d'un fichier texte. Il existe une fonction spécifique **« open() »** qui permet d'ouvrir un fichier et de pouvoir ensuite travailler avec. Cette fonction retourne un objet qui possède alors un certain nombre de méthodes bien utiles pour lire ou écrire dans un fichier.

Comme pour dans un système d'exploitation, nous pouvons ouvrir un fichier avec différents modes qui vont nous permettre de spécifier ce que nous désirons faire par la suite avec ce fichier :

Mode d'ouverture	Caractéristiques associées
'r'	*Ouverture en lecture seule (mode par défaut)*
'w'	*Ouverture en écriture. À chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas, python le crée automatiquement.*
'x'	*Crée impérativement un nouveau fichier et l'ouvre pour l'écriture.*
'a'	*Ouverture en mode ajout à la fin du fichier (append). Si le fichier n'existe pas, python le crée automatiquement.*
'b'	*Le contenu du fichier n'est pas interprété. Il est considéré comme une suite d'information binaire.*

't'	*Le contenu du fichier est un texte (mode par défaut)*
'+'	*Ouvre un fichier pour une mise à jour (lecture et écriture)*

Ci-dessous un exemple très simple qui permet de lire le fichier de notre programme et qui l'affiche à l'écran :

principal.py

```
fichier = open('principal.py')      # ouvre le fichier en lecture
programme = fichier.read()          # lit le contenu du fichier en entier
print(programme)
fichier.close()                     # ferme le fichier
```

Shell Python : [évaluer principal.py]

```
fichier = open('principal.py')      # ouvre le fichier en lecture
programme = fichier.read()          # lit le contenu du fichier en entier
print(programme)
fichier.close()                     # ferme le fichier
```

Dans cet exemple, nous ouvrons le fichier texte ***« principal.py »*** *en lecture seule. Nous utilisons les modes par défaut (lecture et texte). La méthode* ***« read() »*** *de l'objet* ***« fichier »*** *lit l'ensemble du fichier texte et le place dans la variable* ***« programme »****. Attention aux fichiers textes trop grands. Dès que vous avez fini avec votre fichier, pensez à le fermer grâce à la méthode* ***« close() »****.*

Cette dernière remarque est importante. Ne laisser jamais un fichier ouvert. Fermer le dès que possible. À ce sujet, nous pouvons utiliser une autre syntaxe qui nous permet de ne pas oublier cette fermeture et d'avoir un bloc d'instructions associés au traitement souhaité avec le fichier, grâce au mots clés « with … as … ». Voici le même exemple traité avec cette nouvelle syntaxe.

principal.py

```
with open('principal.py') as fichier:  # ouvre le fichier en lecture
    programme = fichier.read()         # lit le contenu du fichier en entier
print(programme)                       # à la sortie du bloc, le fichier est automatiquement fermé
```

Shell Python : [évaluer principal.py]

```
with open('principal.py') as fichier:  # ouvre le fichier en lecture
    programme = fichier.read()         # lit le contenu du fichier en entier
print(programme)                       # à la sortie du bloc, le fichier est automatiquement fermé
```

Plutôt que de lire le fichier en une seule fois avec la méthode ***« read() »****, il est possible de récupérer chaque ligne séparément grâce à la méthode* ***« readline() »****. Malheureusement, cette méthode récupère également le caractère de saut de ligne* ***'\n'****.*

Notez qu'il existe également la méthode ***« readlines() »*** *qui lit le fichier texte en entier et récupère la totalité sous la forme d'une liste de chaînes de caractères.*

principal.py

```
with open('principal.py') as fichier:
    ligne = 'pas la fin'
    while ligne != '':                 # tant que nous ne sommes pas à la fin du fichier
        ligne = fichier.readline()     # lecture d'une ligne du fichier
        print(ligne)
```

Shell Python : [évaluer principal.py]

```
with open('principal.py') as fichier:      # remarquez bien les sauts de lignes

    ligne = 'pas la fin'

    while ligne != '':                     # tant que nous ne sommes pas à la fin du fichier

        ligne = fichier.readline()         # lecture d'une ligne du fichier

        print(ligne)
```

Nous pouvons faire beaucoup plus concis en prenant directement l'objet ***« fichier »*** *qui, associé à la boucle* ***« for … in … »*** *nous fournit l'ensemble des lignes du fichier séparément. Nous obtenons alors le même résultat (toujours les retours à la ligne intégrés à la ligne en cours).*

principal.py

```
with open('principal.py') as fichier:
    for ligne in fichier:
        print(ligne)
```

Une des solutions consiste à ne pas afficher le dernier caractère de la ligne : ***'\n'*** *:*

principal.py

```
with open('principal.py') as fichier:
    for ligne in fichier:
        print(ligne[ :-1])
```

Shell Python : [évaluer principal.py]

```
with open('principal.py') as fichier:
    for ligne in fichier:
        print(ligne)
```

*Une autre solution, si le fichier texte n'est pas trop important, est de reprendre la méthode **« read() »** et de choisir comme délimiteur le caractère **'\n'**.*

principal.py

```
with open('principal.py') as fichier:
    for ligne in fichier.read().split('\n'):
        print(ligne)
```

*Une dernière alternative consiste à supprimer les espaces et les séparateurs (tabulations comprises) présent dans une chaîne de caractères avec la méthode **« strip() »**.*

principal.py

```
with open('principal.py') as fichier:
    for ligne in fichier:
        print(ligne.strip())
```

Shell Python : [évaluer principal.py]

```
with open('principal.py') as fichier:
for ligne in fichier:
print(ligne.strip())
```

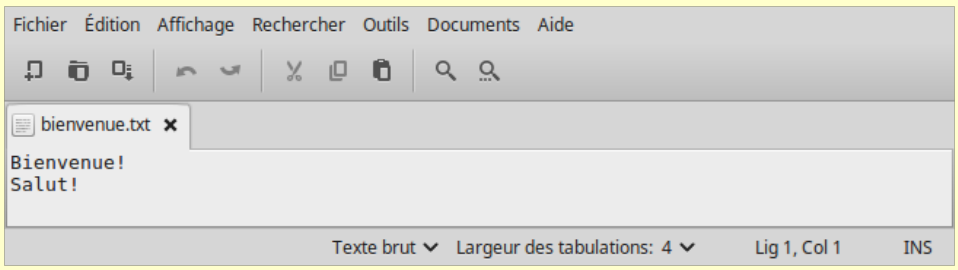
ÉCRITURE DANS UN FICHIER TEXTE

Maintenant que nous savons comment lire un fichier texte, je vous propose de réaliser l'opération inverse, c'est-à-dire d'écrire dans un fichier texte. Deux modes existent. Soit nous créons un nouveau fichier ou nous écrasons celui déjà existant (mode **'w'**). Soit nous rajoutons des informations à un fichier déjà connu (mode **'a'**).

*Je vous propose de créer un nouveau fichier **« bienvenue.txt »** dans lequel nous proposerons deux lignes d'écritures. Remarquez au passage que vous devez spécifier dans votre texte, le retour à la ligne explicitement, sinon le deuxième texte s'écrit à la suite du précédent.*

principal.py

```
with open('bienvenue.txt', 'w') as fichier:     # ouverture en écriture (écrase l'ancien fichier)
    fichier.write('Bienvenue!\n')               # création du fichier si non existant
    fichier.write('Salut!\n')
```



*C'est vraiment très simple à produire. Remarquez que si vous disposez d'une liste de texte, vous pouvez utiliser directement la méthode **« writelines() »** en lieu et place de la méthode **« write() »**.*

ÉCRIRE ET LIRE DES VALEURS NUMÉRIQUES DANS UN FICHIER TEXTE

Il est bien entendu possible d'enregistrer n'importe quel type de valeurs dans un fichier texte. Toutefois, comme il s'agit d'un fichier texte, vous êtes obligé de transformer vos nombres sous forme de chaîne de caractères, et vice versa. Pensez également à proposer un séparateur entre vos différentes valeurs numériques pour qu'il soit facile de les discriminer par la suite lors de la lecture du fichier.

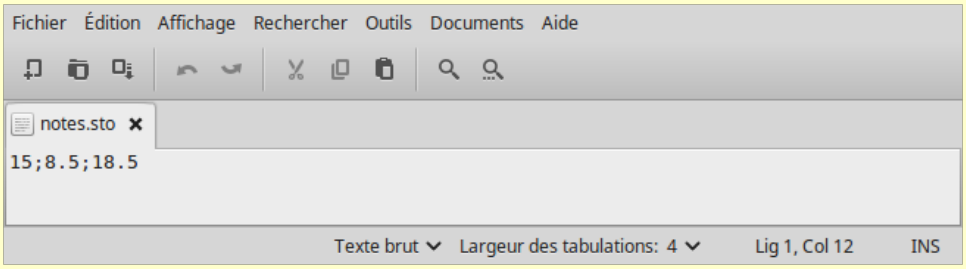
*À titre d'exemple, je vous propose de réaliser un script **« enregistrer.py »** qui permet de stocker dans un fichier, un ensemble de notes saisies par l'utilisateur. Un autre script **« ouvrir.py »** va lire ce fichier et afficher à l'écran la moyenne des notes stockées dans le fichier.*

enregistrer.py

```
texte = ''
while True:
    note = input('Votre note (-1 pour quitter) : ')
    if note == '-1': break
    texte += note + ';'
with open('notes.sto', 'w') as fichier: fichier.write(texte[:-1])
```

Shell Python : [évaluer enregistrer.py]

```
Votre note (-1 pour quitter) : 15
Votre note (-1 pour quitter) : 8.5
Votre note (-1 pour quitter) : 18.5
Votre note (-1 pour quitter) : -1
```



ouvrir.py

```
with open('notes.sto') as fichier: notes = fichier.read().split(';')
print(notes, end=' :  ')
somme = 0
for note in notes: somme += float(note)
print('Moyenne =', somme/len(notes))
```

Shell Python : [évaluer ouvrir.py]

```
['15.0', '8.5', '18.5'] :  Moyenne = 14.0
```

*N'importe quel séparateur peut faire l'affaire. Il suffit de s'entendre. Il faut être cohérent entre l'écriture et la lecture du fichier. Ici, nous avons choisi le séparateur le **« ; »** utilisé dans le format **« CSV »** qui intervient dans les feuilles de calcul.*

ÉCRIRE ET LIRE DES VALEURS NUMÉRIQUES DANS UN FICHIER BINAIRE

Travailler avec des fichiers textes est très facile à manipuler. C'est très polyvalent. Nous pouvons travailler avec des langages de programmation différents et obtenir le même résultat. Toutefois, il peut être intéressant de cacher les données enregistrées et de produire un fichier binaire non accessible par un simple éditeur de texte.

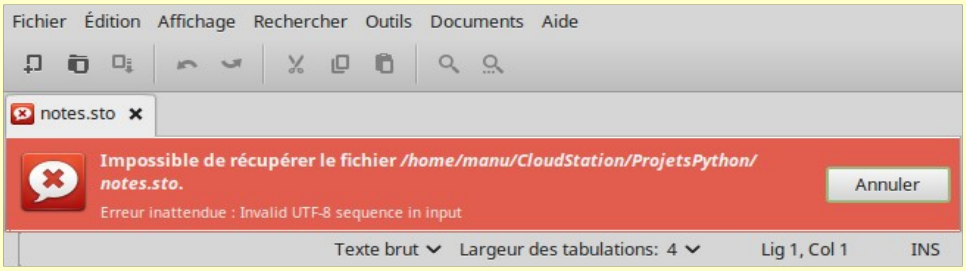
*Pour cela, il faut utiliser le mode **« b »** à la fois en écriture et en lecture du fichier. Ensuite, il existe un module spécialisé **« pickle »** capable de prendre n'importe quelle variable (avec une valeur numérique ou même un objet de n'importe quelle nature) et de la transformer en données binaires enregistrable.*

enregistrer.py

```
import pickle
notes = []
while True:
    note = input('Votre note (-1 pour quitter) : ')
    if note == '-1': break
    notes.append(note)
with open('notes.sto', 'wb') as fichier:    # enregistrement en binaire
    pickle.dump(notes, fichier)
```

Shell Python : [évaluer enregistrer.py]

```
Votre note (-1 pour quitter) : 15
Votre note (-1 pour quitter) : 8.5
Votre note (-1 pour quitter) : 18.5
Votre note (-1 pour quitter) : -1
```



ouvrir.py

```
import pickle
with open('notes.sto', 'rb') as fichier:       # lecture en mode binaire
    notes = pickle.load(fichier)
print(notes, end=' :  ')
somme = 0
for note in notes: somme += float(note)
print('Moyenne =', somme/len(notes))
```

Shell Python : [évaluer ouvrir.py]

```
['15.0', '8.5', '18.5'] :  Moyenne = 14.0
```

*La fonction **« dump() »** du module **« pickle »** attend deux arguments : le premier est la variable à enregistrer, le second est l'objet fichier dans lequel nous travaillons. La fonction **« load() »** effectue le travail inverse, c'est-à-dire la restitution de chaque variable avec son type.*

*Voici ci-dessous le fichier **« notes.sto »** en format binaire. Vous avez la suite des octets représentant la liste des notes :*

notes.sto ×

```
00000000 80 03 5D 71 00 28 58 02 00 00 00 31 35 71 01 |..]q.(X....15q.|
0000000f 58 03 00 00 00 38 2E 35 71 02 58 04 00 00 00 |X....8.5q.X....|
0000001e 31 38 2E 35 71 03 65 2E                      |18.5q.e.|
```

CRÉATION DE NOUVELLES CLASSES POUR LA GESTION DES ENTRÉES-SORTIES GPIO

Comme nous l'avons montré en début d'étude la gestion du **GPIO** peut se faire simplement au travers de la mise en œuvre de fichiers spécifiques, et vu que nous venons de voir comment faire avec Python, je vous propose maintenant de créer des classes qui vont encapsuler toutes ces fonctionnalités là afin que cette partie là soit cachée à l'utilisateur et que cela soit très simple à utiliser.

*D'après nos différentes expériences effectuées, dès que nous activons une broche du **GPIO**, elle est automatiquement considérée comme une entrée. Il suffit de consulter son entrée (fichier **« value »**) pour connaître son état. Pour avoir une sortie, vous devez donner une information supplémentaire (**« out »** dans le fichier **« direction »**), et vous pouvez dès lors activer cette sortie (**« 1 »** dans le fichier **« value »**) .*

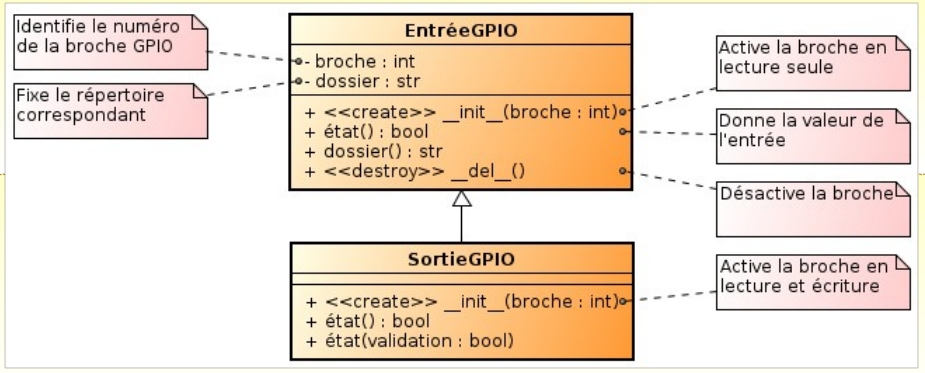
La sortie possède exactement les mêmes fonctionnalités que pour une entrée avec en plus la possibilité d'activer la broche souhaitée. Dans ce contexte, je prévoie donc de créer une classe spécifique pour l'entrée « EntréeGPIO » et une autre classe pour la sortie « SortieGPIO » qui hérite de la première.

La programmation objet est vraiment très intéressante pour ce genre de projet. Il suffit ici de créer uniquement deux classes qui implémentent toutes cette gestion de fichier qui sera automatiquement gérée (et même cachée) sans que l'utilisateur ne s'en rende compte. Ensuite, ces fonctionnalités s'appliqueront à toutes les broches (du coup tous les objets associés à l'une de ces classes). Pour l'utilisateur, je le répète, ce sera très simple à manipuler. Cela vaut le coup de créer ces deux classes.

gpio.py

```
from os import path

class EntréeGPIO:
    def __init__(self, broche):
        self.__broche = broche
        self.__dossier = '/sys/class/gpio/gpio{}/'.format(broche)
        if not path.exists(self.__dossier):
            with open('/sys/class/gpio/export', 'w') as fichier: fichier.write(str(broche))
        with open(self.__dossier + 'direction', 'w') as fichier: fichier.write('in')

    @property
    def état(self):
        with open(self.__dossier + 'value') as fichier:
            caractère = fichier.read()[0]
            return caractère == '1'

    @property
    def dossier(self): return self.__dossier

    def __del__(self):
        with open('/sys/class/gpio/unexport', 'w') as fichier: fichier.write(str(self.__broche))


class SortieGPIO(EntréeGPIO):
    def __init__(self, broche):
        super().__init__(broche)
        with open(self.dossier + 'direction', 'w') as fichier: fichier.write('out')

    @property
    def état(self): return super().état

    @état.setter
    def état(self, validation):
        with open(self.dossier + 'value', 'w') as fichier:
            if validation: fichier.write('1')
            else:          fichier.write('0')
```

Le script de la page précédente, relativement court, nous montre la constitution des deux classes qui, comme nous l'avons évoqué plus haut, ne fait pratiquement que la gestion de fichiers spécifiques.



*Le constructeur de la première classe **« EntréeGPIO »** active, si cela n'est pas déjà fait, la broche du **GPIO** demandée dans l'argument du constructeur. Cette broche est fixée en entrée (mode par défaut normalement) dans le cas où cette broche avait déjà été activée.*

Deux attributs privées sont générés à cette occasion. Nous conservons le numéro de broche qui sera utile pour le destructeur par la suite, et nous gardons également en mémoire le chemin du dossier correspondant à cette broche qui sera intéressante au moment de la consultation de l'entrée.

*L'état de l'entrée est donné par la méthode du même nom **« état() »** qui est implémentée sous forme de propriété, qui sera plus facile à manipuler pour le programme principal (cela ne coûte pas grand-chose de rajouter le décorateur **« @property »**).*

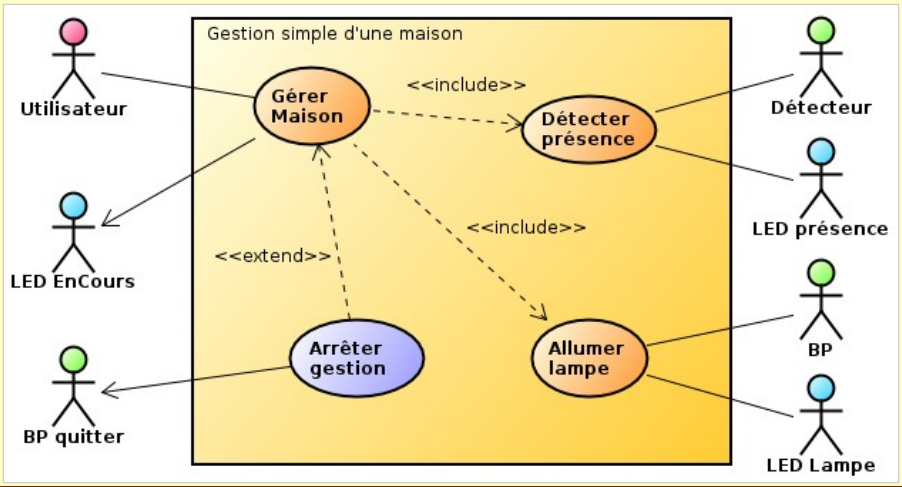
Un destructeur est prévu pour cette classe qui sera automatiquement appelé par tous les objets de cette classe (ou apparentée) lors de la fin du programme principal. Son objectif est de désactiver la broche correspondante qui d'un point de vue électronique est beaucoup plus sécurisant.

*Le constructeur de la deuxième classe **« SortieGPIO »** reprend les bases du constructeur parent en activant cette fois-ci la broche en sortie.*

*La faculté d'une sortie par rapport à une entrée est qu'il possible de changer l'état de la broche afin de pouvoir éventuellement l'activer. Là aussi, j'ai prévu une propriété en écriture **« état(validation) »**. Par contre pour que cela fonctionne correctement, il est nécessaire que la propriété en lecture existe également dans la même classe. La méthode de lecture **« état() »** écrite dans la classe **« EntréeGPIO »** est donc redéfinie dans la classe **« SortieGPIO »**.*

raspberry.py

```
from gpio import EntréeGPIO, SortieGPIO
from time import sleep

quitter = EntréeGPIO(20)
détecteur = EntréeGPIO(23)
bouton = EntréeGPIO(21)
enCours = SortieGPIO(4)
présence = SortieGPIO(17)
lampe = SortieGPIO(22)

enCours.état = True

while not quitter.état:
   présence.état = détecteur.état
   lampe.état = bouton.état
   sleep(.1)

enCours.état = False
présence.état = False
lampe.état = False

print('fin du programme')
```



*Cet exemple de programme principal est assez modeste. Malgré tout, il montre l'utilisation de six broches du **GPIO** de la **Raspberry**. La première partie consiste à créer justement tous les objets nécessaires à votre application. Ensuite, votre programme ne fait qu'utiliser la propriété **« état »**,de l'ensemble des objets créés, soit en lecture pour une entrée soit en écriture pour une sortie.*

Toute la partie compliqué de la gestion du GPIO est bien totalement cachée au concepteur du programme principal. Pour lui, c'est d'une extrême facilité de développer des applications spécifiques à la Raspberry. Son seul soucis est de concevoir un algorithme qui va bien correspondre à ces attentes au niveau des différents capteurs et actionneurs.
Par ailleurs, l'activation et la désactivation des broches ne se voient pas puisqu'elles sont décrites respectivement dans le constructeur et le destructeur (méthodes non appelées explicitement). La programmation objet est vraiment prépondérante dans ce cadre là.

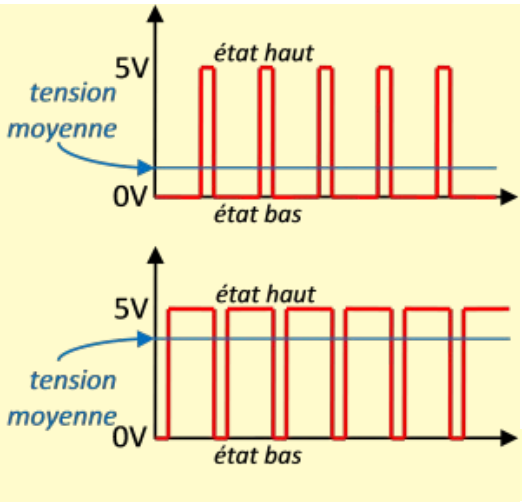
ÉDITEUR DÉPORTÉ

Les deux codes sources précédents peuvent être édités directement sur la raspberry avec l'éditeur **« nano »**, ce qui n'est pas toujours confortable. L'idéal est de rester sur l'éditeur sur lequel vous travaillez habituellement et de déployer ensuite vos sources au moyen de la commande **« scp »** :

```
$ scp fichier login@serveur:chemin // copie un fichier particulier sur une machine distante dans le dossier spécifié par chemin
$ scp gpio.py raspberry.py pi@192.168.1.22:/home/pi/logiciels // copie des deux fichiers précédents dans la bonne
                                                              // raspberry et dans le bon répertoire
```

RPi.GPIO

Plutôt que de passer par le système de fichiers pour gérer les broches du **GPIO**, il est préférable d'installer une bibliothèque qui gère le **GPIO** en passant par l'électronique interne et qui propose un accès direct aux broches.



*Nous pourrons piloter ces broches par **PWM**, ce qui permettra de proposer un rapport cyclique variable sur une sortie avec une fréquence prédéterminée, nous permettant de générer une tension moyenne variable de 0 à 5v.*

*En principe, un port numérique ne peut délivrer qu'une information « tout ou rien » : allumé/éteint. La tension sur la sortie numérique ne peut être que **0V** ou **5V**, jamais entre les deux. Mais, si nous la faisons varier très rapidement (**0V-5V-0V-5V-…**) la valeur moyenne est différente. En jouant sur la durée des états hauts (5V) et états bas (**0V**), nous modifions la valeur de cette tension moyenne, à volonté. Cette technique s'appelle la Modulation de Largeur d'Impulsion (**MLI**) – ou **PWM** (Pulse Width Modulation) en anglais.*

La première démarche consiste à installer cette bibliothèque sur votre raspberry, à l'aide de la commande suivante : sudo apt-get install rpi.gpio

Commnades utilies pour la gestion du GPIO

```
import RPi.GPIO as GPIO  # De façon classique, vous devez importer le module correspondant
                         # Chaque utilisation du GPIO sera précédé par ce même item, pour éviter tout ambiguïté.

GPIO.setmode(GPIO.BCM)  # Avant toute utilisation, vous devez spécifier le mode de broche (toujours le même)

GPIO.setup(12, GPIO.IN)          # Vous devez ensuite spécifier les broches qui sont en sortie ou en entrée
GPIO.setup(12, GPIO.OUT)                                    # Il existe les constantes IN et OUT pour cela
GPIO.setup(12, GPIO.OUT, initial=GPIO.HIGH)

GPIO.input(12)                       # Méthode qui permet de connaître l'état d'une entrée (ou d'une sortie)

GPIO.output(12, GPIO.LOW)               # Méthode qui permet de spécifier une valeur en sortie LOW ou HIGH
GPIO.output(12, not GPIO.input(12))

p = GPIO.PWM(broche, fréquence)            # Mise en place du mode PWM sur une broche particulière
p.start(rapport_cyclique)                  # ici, rapport_cyclique vaut entre 0.0 et 100.0
p.ChangeFrequency(nouvelle_fréquence)
p.ChangeDutyCycle(nouveau_rapport_cyclique)
p.stop()

GPIO.wait_for_edge(broche,  GPIO.RISING)  #  Méthode  qui  interrompt  le  programme  principal  jusqu'à  que  la
                                          # broche change d'état, ici passage à l'état haut. Il existe aussi
                                          # GPIO.FALLING (vers l'état bas), GPIO.BOTH (les deux fronts).

def my_callback(broche):                 # fonction automatiquement appelée (méthode de rappel)
    print("un evenement s'est produit")

GPIO.add_event_detect(broche, GPIO.BOTH, callback=my_callback) # Méthode qui permet d'appeler
                                             # automatiquement une fonction lorsqu'un changement d'état sur
                                             # une broche se produit (RISING, FALLING ou BOTH)
                                             # (Mécanisme d'interruption).
```

À titre d'exemple, je vous propose de tester toutes ces fonctionnalités en ayant une led qui s'éclaire progressivement, une autre qui clignote et une autre qui s'allume lorsque nous appuyons sur un bouton et qui s'éteint dès que nous le relâchons. Enfin, un dernier bouton permet de débuter la séquence d'éclairage progressif et le clignotement.

rpi-gpio.py

```
import RPi.GPIO as GPIO
from time import sleep

def détection(broche):      # fonction automatiquement appelée (méthode de rappel)
   global témoin            # la led « témoin » reste allumée tant que le bouton « bp » reste enfoncé
   GPIO.output(temoin, GPIO.input(broche))

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)  # désactivation des messages d'alerte lorsque le programme est en cours d'exécution

ledcli=4      # led clignotante
ledvar=17     # led à éclairement variable
témoin=22     # led témoin qui s'allume tant que le bouton « bp » est enfoncé
démarrer=24   # bouton qui permet de démarrer le cycle d'éclairage progressif ainsi que le clignotement
bp=25         # bouton poussoir

GPIO.setup(ledcli, GPIO.OUT)
GPIO.setup(ledvar, GPIO.OUT)
```

```
GPIO.setup(temoin, GPIO.OUT)
GPIO.setup(démarrer, GPIO.IN)
GPIO.setup(bp, GPIO.IN)
ledvar = GPIO.PWM(ledvar, 100)

GPIO.add_event_detect(bp, GPIO.BOTH, callback=détection)   # fonction de rappel lorsque nous appuyons
                                                           # sur le bouton « démarrer »

GPIO.output(ledcli, GPIO.HIGH)      # led clignotante allumée en permanence pour confirmer le démarrage
ledvar.start(0)                     # de l'application

GPIO.wait_for_edge(démarrer, GPIO.RISING)   # Attente de l'appui sur la bouton « démarrer »

for i in range(300):                # éclairage progressif pendant 3 secondes
   ledvar.ChangeDutyCycle(i/3)
   if not i%10: GPIO.output(ledcli, not GPIO.input(ledcli))  # clignotement avec une période de 2/10 s
   sleep(0.01)                                               # attente de 1/100 s

GPIO.output(ledcli, GPIO.LOW)       # désactivation de toutes les sorties avant de quitter le programme
GPIO.output(temoin, GPIO.LOW)
ledvar.stop()
```