التجربة الثالثة والرابعة

برمجة منافذ التحكم ضمن لوحة Raspberry Pi

هدف التجربة:Experiment's objective

تهدف هذه التجربة إلى التعرف على كيفية برمجة منافذ GPIO ضمن لوحة Raspberry Pi باستخدام لغة python بحيث يمكن أن تتحول لوحة Raspberry Pi إلى بديل متطور عن المتحكمات المصغرة Microcontrollers, وهذه التجربة هي سلسلة من عدة تجارب تتمحور حول نفس الهدف, وتتضمن أساسيات اللغة ثم كتابة خوارزميات لتطبيقات مختلفة باستخدام الحساسات والمحركات وملحقات أخرى.

مخارج التحكم GPIO

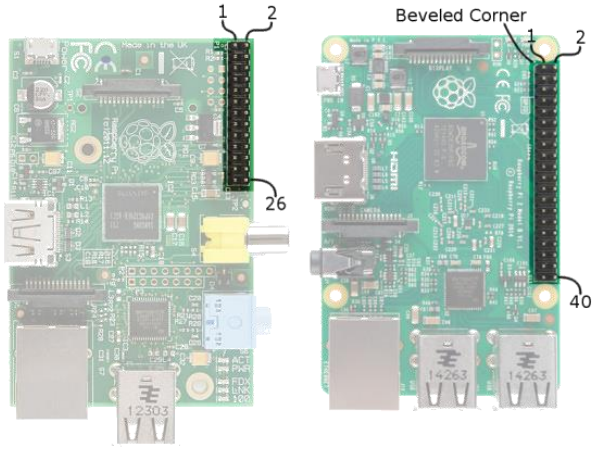
هي اختصار لعبارة General –Purpose input/output، تمتلك لوحة Raspberry Pi 26 قطب أو 40 قطب وذلك حسب نوعها كما رأينا في التجربة الأولى.

القطب(1,17): مخرج للطاقة (3.3V-50mA)، يجب ألا يتم سحب تيار أكبر من 50mA كي لا تتضرر لوحة الراسبيري.

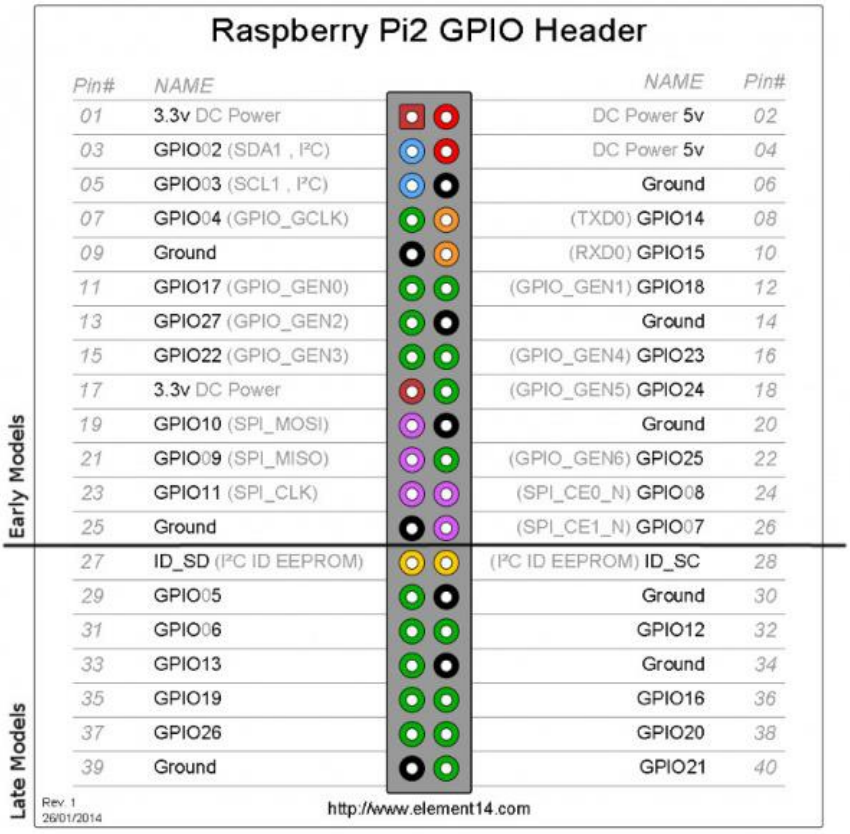
القطب(2,4): مخرج أو مدخل للطاقة (5V)، يتصل هذا المخرج بمنفذ MicroUsb ويمكن استخدامه كمدخل للطاقة عن طريق توصيل بطارية أو منبع جهد (5V) لتشغيل اللوحة.

الأقطاب(6,9,14.20,25): نقاط التوصيل بالأرضي GND.

الأقطاب(3,5,7,8,10,11,12,13,15,16,18,19,21,22,23,24,26): تمتلك هذه اللوحة سبعة عشر منفذ خاص، يمكن برمجة هذه المنافذ إما على صورة دخل أو خرج أو منفذ اتصال (I2C, SPI, UART,1Wire), ويتم ضبط تشغيلها عن طريق البرمجة بلغات (C++, JAVA, C#, Python, Perl Ruby, Pascal) التي يدعمها نظام لينكس.



والشكل التالي يوضح وظيفة كل قطب:



لغة بايثون Python:

- **نشأت لغة بايثون في مركز العلوم والحاسب الآلي في امستردام عام 1991 على يد Guido van Rossum حيث تمت كتابة نواة اللغة بلغة C.**
- **غالبا ما تحصل هذه اللغة على الترتيب الأول في قائمة أشهر لغات البرمجة في العالم تبعا للاحصائيات العالمية.**
- **تعمل على جميع أنظمة التشغيل وإصداراتها المختلفة وأنظمة الهواتف المحمولة.**

ولها مميزات عديدة أخرى سنكتشفها معاً في التجارب القادمة إن شاء الله.

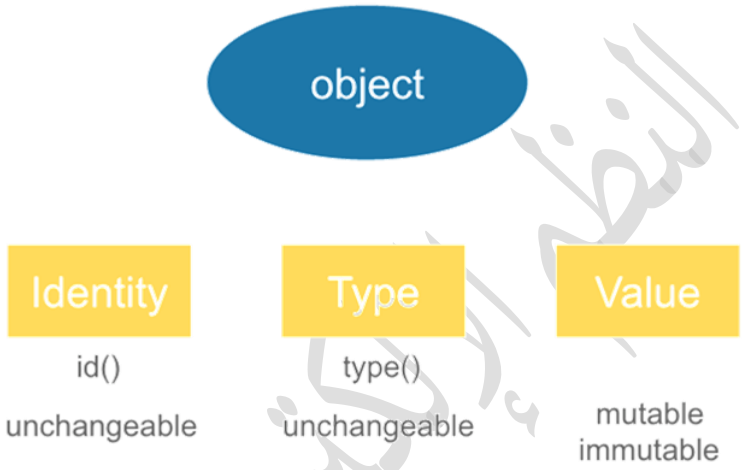
ملحق 1:

التعرف على أساسيات python ضمن البيئة البرمجية الشاملة Spyder

أولا-الكائنات: Objects

- تحتوي لغة python العديد من أنماط المعطيات (Data types)، والتي تعد جزء من نواة اللغة، ومن المهم فهم طبيعة المعطيات ضـــمن البرنامج الذي نقوم بكتابته، والتي يتم تمثيلها باســـتخدام الكائنات، كذلك فهم العلاقات بين الكائنات.

- تقسم الكائنات ضمن python إلى نوعين: الأول (mutable objects)، وهي التي تملك قيمة يمكن أن تتغير، والنوع الثاني (immutable objects)، وهي الكائنات التي لاتتغير قيمتها المسندة إليها بعد الإنشاء.
- لكل كائن ثلاث خصائص أساسية: (Type–Value–Identity)، إن النمط Type يحدد نوع الكائن (string,int,float,list,……)، والقيمة value تمثل محتوى الكائن مثل رقم معين أو عبارة محددة حسب نوعه, أما هوية الكائن Identity فهي عبارة عن رقم الكائن الذي يأخذه عندما يتم تخزينه ضمن الذاكرة، وكل كائن يملك رقم هوية خاص به ولايتغير.



- تمتلك لغة python أنواع معطيات متعددة مثل (Integer–Floating point numbers– complex numbers), وبمجال غير محدود، ويمكن الدمج بين أنماط الأعداد المختلفة، وهي جميعها تدعم كل العمليات الحسابية (*,–,+,/).

```
In [55]: 12/4
Out[55]: 3.0

In [56]: _
Out[56]: 3.0

In [57]: _**12
Out[57]: 531441.0
```

```
In [48]: 123**144
Out[48]:
883763472157120985653569148980438269910635527541336730666023752
231531701798857602343291165555029820128380659438121081332830002
421669122701382578143002700068787659798187493020798885571014758
983355337860977625393089483178983462821640772806494378399186063
627121529339874644351088500643927702316112612704 1

In [49]: 9/2
Out[49]: 4.5

In [50]: 9//2
Out[50]: 4

In [51]: 15/2.3
Out[51]: 6.521739130434783
```

ثانيا-الطرق Attributes:

تمتلك الكائنات في لغة python معطيات وتوابع خاصة بها تسمى Attributes، اسم Attributes: يتبع اسم الكائن، ويتم الفصل بينهما بنقطة.

تقسم Attributes: إلى نوعين:(الأول يدعى Attributes، قيمتها تتعلق بكائن محدد، الثاني يدعى methods، تحتوي على تابع أو أكثر، تعطى القيم للكائنات، أي تنجز التوابع والعمليات على الكائنات التي تحتويها)، إنّ نمط الكائن يحدد نوع العمليات التي تدعمها الطريقة.

ملاحظة: مصطلح (instance) يطلق على object الذي نستدعي به الطريقة أو يتم تكوينه باستخدامها.

```
In [1]: import numpy as np

In [2]: x= np.array([1,3,5])

In [3]: y= np.array([1,9,5])

In [4]: x.mean()
Out[4]: 3.0

In [5]: y.mean()
Out[5]: 5.0
```

```
In [6]: x.shape
Out[6]: (3,)

In [7]: y.shape
Out[7]: (3,)
```

ملاحظة:

Meanهي method، بينما shape هي Attribute.

ثالثا-السلاسل المتتالية Sequences:

تقسم إلى ثلاثة أنواع (list،tuple، range):

النوع الأول list: هي عبارة عن سلسلة متغيرة من الكائنات وفق أي نوع من المعطيات، وتستخدم لتخزين المعطيات المتجانسة.

```
In [22]: num = [2,6, 4,8,10]

In [23]: num[0]
Out[23]: 2

In [24]: num[-1]
Out[24]: 10

In [25]: num.append(12)

In [26]: num
Out[26]: [2, 6, 4, 8, 10, 12]

In [27]: f=[14,16]

In [28]: num =num+f

In [29]: num
Out[29]: [2, 6, 4, 8, 10, 12, 14, 16]

In [30]: type(f)
Out[30]: list
```

```
In [36]: names=["aa","cc","dd","bb"]
```

```
In [43]: names.sort()

In [44]: names
Out[44]: ['aa', 'bb', 'cc', 'dd']
```

```
In [46]: names.reverse()

In [47]: names
Out[47]: ['dd', 'cc', 'bb', 'aa']
```

مقارنة بين list & String:

List	String
متغيرة	ثابتة
تقبل أي نوع معطيات	تحتوي كائنات من نوع واحد فقط(محرفي)
لا تمتلك طرق خاصة بها	لها طرق خاصة بها

النوع الثاني Tuple: يستخدم لتخزين بيانات ثابتة ضمن كائن واحد يحتوي أجزاء مختلفة، تفيد في التوابع التي تعيد قيم متعددة حيث تخزن ضمن single tuple object، وهي تقبل الإضافة كما سنرى في الأمثلة التالية.

```
In [1]: a =(2,3)

In [2]: type (a)
Out[2]: tuple

In [3]: b= (2)

In [4]: type (b)
Out[4]: int

In [5]: b= (2,)

In [6]: type (b)
Out[6]: tuple
```

```
In [7]: len(a)
Out[7]: 2

In [8]: a+(4,5,6,7)
Out[8]: (2, 3, 4, 5, 6, 7)

In [9]: a
Out[9]: (2, 3)

In [10]: a[1]
Out[10]: 3
```

- طريقة (Tuple packing):

```
In [11]: x=1.2

In [12]: y=3.2

In [13]: coordinate=(x,y)

In [14]: coordinate
Out[14]: (1.2, 3.2)

In [15]: type(coordinate)
Out[15]: tuple
```

- طريقة(unpacking):

```
In [16]: (c1,c2)=coordinate

In [17]: c1
Out[17]: 1.2

In [18]: c2
Out[18]: 3.2
```

- ما هو list of tuple، مع ذكر مثال.

النوع الثالث Range: سلسلة ثابتة من الأعداد الصحيحة تستخدم مع الحلقات.

```
In [1]: range(5)
Out[1]: range(0, 5)

In [2]: a = list(range(5))

In [3]: a
Out[3]: [0, 1, 2, 3, 4]

In [4]: a = list(range(1,6))

In [5]: a
Out[5]: [1, 2, 3, 4, 5]

In [6]: a = list(range(1,16,2))

In [7]: a
Out[7]: [1, 3, 5, 7, 9, 11, 13, 15]
```

رابعاً-السلاسل المحرفية:Strings

هي عبارة عن كائن ثابت يمكن كتابته ضـــمن quotes مفردة أو مزدوجة " أو ثلاثية, يمكن تطبيق توابع Lists عليها.

```
In [1]: s = "python"

In [2]: len(s)
Out[2]: 6

In [3]: s[0]
Out[3]: 'p'

In [4]: s[-1]
Out[4]: 'n'

In [5]: s[0:3]
Out[5]: 'pyt'
```

```
In [13]: "p" in s
Out[13]: True

In [14]: "Y" in s
Out[14]: False
```

كذلك يمكن تطبيق العمليات الحسابية على السلاسل المحرفية.

```
In [10]: 12+12
Out[10]: 24

In [11]: "hello" + " world"
Out[11]: 'hello world'
```

```
In [13]: s="python"

In [14]: 3*s
Out[14]: 'pythonpythonpython'
```

```
In [16]:  name ="Aula"

In [17]: name2= name.replace("A","O")

In [18]: name2
Out[18]: 'Oula'
```

```
In [15]: "eight equals" +str(8)
Out[15]: 'eight equals8'
```

ملاحظة:

يمكننا تابع dir( ) من معرفة جميع الطرق والتوابع والثوابت وغيرها والموجودة ضمن موديول معين أو التي يمكن تطبيقها على عنصر من نوع معين كما نلاحظ في الأمثلة:

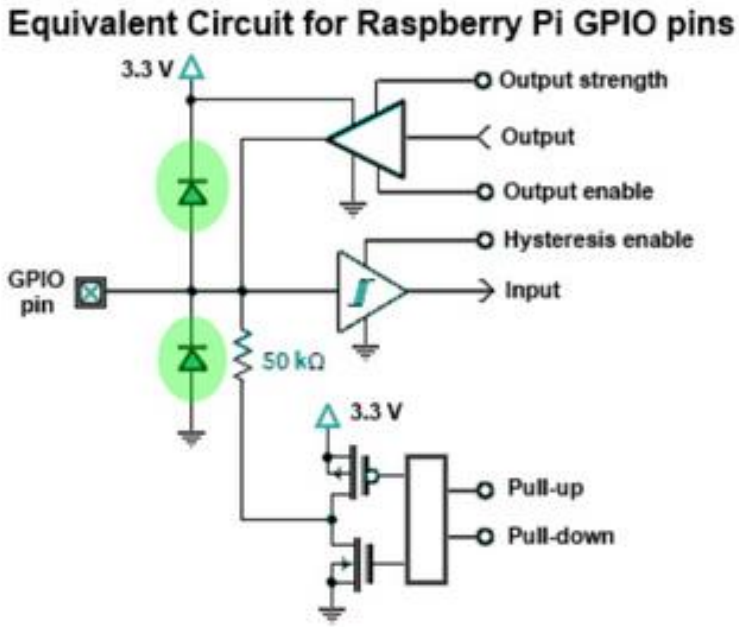
```
In [31]: import math

In [32]: dir(math)
Out[32]:
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf'
```

```
In [28]: name ="Aula"

In [29]: type(name)
Out[29]: str

In [30]: dir(name)
Out[30]:
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
```

التطبيق العملي:

تعرفنا سابقا على أقطاب التحكم ضمن لوحة راسبيري باي والتي تسمى:

GPIO (general-purpose input/output) ، وعند برمجتها يمكن اختيار طريقة الترقيم الفيزيائي أو طريقة الترقيم وفق أقطاب المعالج BCM.



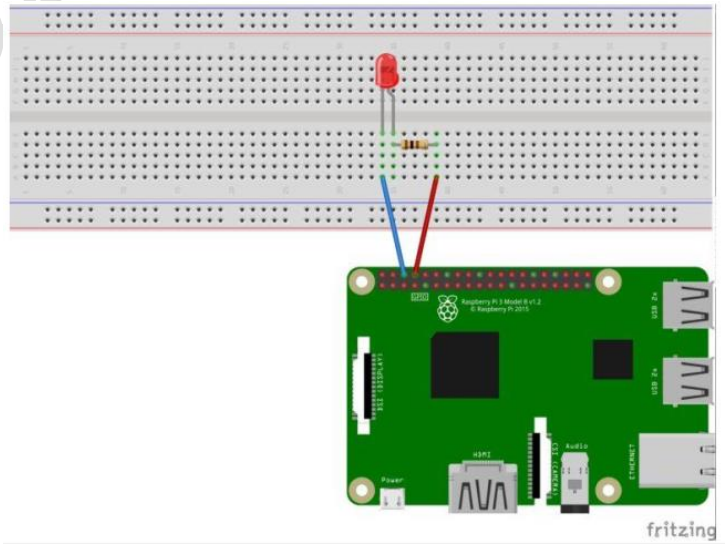

```
#set up GPIO using BCM numbering
GPIO.setmode(GPIO.BCM)
#setup GPIO using Board numbering
GPIO.setmode(GPIO.BOARD
```

ولتهيئة الأقطاب كمداخل أو مخارج نستخدم الطريقة التالية:

```
GPIO.setup(12, GPIO.OUT)
GPIO.setup(13, GPIO.IN)
GPIO.setup(23, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(24, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

التطبيق الأول :

الهدف: تهيئة قطب من لوحة Raspberry Pi واستخدامه كخرج.



```
1.  import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
2.  from time import sleep # Import the sleep function from the time module
3.
4.  GPIO.setwarnings(False) # Ignore warning for now
5.  GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
6.  GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial
    value to low (off)
7.
8.  while True: # Run forever
9.   GPIO.output(8, GPIO.HIGH) # Turn on
10.  sleep(1) # Sleep for 1 second
11.  GPIO.output(8, GPIO.LOW) # Turn off
12.  sleep(1) # Sleep for 1 second
```

- اكتب البرنامج التالي وخزنه باسم blinking_led.py ,وصل الدارة الموضحة في الشكل سجل ملاحظاتك, ثم قم بتعديل البرنامج بطباعة عبارة توضح حالة المتصل الثنائي الضوئي.
- قم بتعديل التطبيق بإضافة كلمة سر للبرنامج عندما تكون صحيحة يعمل المتصل الضوئي.

ملاحظة: يمكن تشغيل البرنامج باستخدام سطر الأوامر بالتعليمة التالية :

```
$ python blinking_led.py
```

أو أحد برامج لغة python (Geany--IDLE- Mu- Thonny)

طريقة التصريح عن التوابع في Python:

Defining a function a Python

```
def function(parameter):
    return parameter
```

Create a working function in Python

```
def absolute(number):
    if number < 0:
        return number * -1
    else:
        return number
```

```
absolute(10)
absolute(-10)
```

التطبيق الثاني:

استخدام طريقة إنشاء التوابع لتشغيل إشارة مرور

```
import time
```

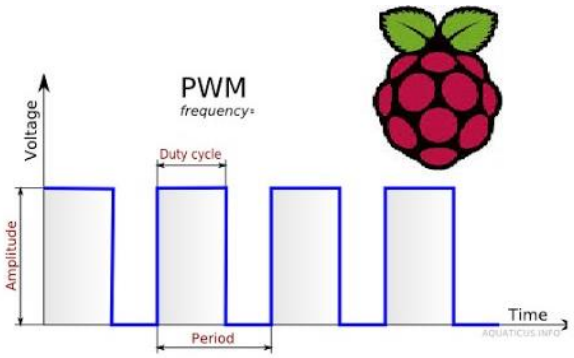
```
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
LED_RED = 7
LED_YELLOW = 11
LED_GREEN = 13
LEDOFF = 0
LEDON = 1
def setupgpio():
  GPIO.setup(LED_RED, GPIO.OUT)
  GPIO.setup(LED_YELLOW, GPIO.OUT)
  GPIO.setup(LED_GREEN, GPIO.OUT)
def alloff (val):
    GPIO.output (LED_RED, LEDOFF)
    GPIO.output (LED_YELLOW, LEDOFF)
    GPIO.output (LED_GREEN, LEDOFF)
    time.sleep(val)
def allon (val):
    GPIO.output (LED_RED, LEDON)
    GPIO.output (LED_YELLOW, LEDON)
    GPIO.output (LED_GREEN, LEDON)
    time.sleep(val)
def red (val):
    GPIO.output (LED_RED, LEDON)
    GPIO.output (LED_YELLOW, LEDOFF)
    GPIO.output (LED_GREEN, LEDOFF)
    time.sleep(val)
def yellow (val):
    GPIO.output (LED_RED, LEDOFF)
    GPIO.output (LED_YELLOW, LEDON)
    GPIO.output (LED_GREEN, LEDOFF)
    time.sleep(val)
def green (val):
    GPIO.output (LED_RED, LEDOFF)
    GPIO.output (LED_YELLOW, LEDOFF)
    GPIO.output (LED_GREEN, LEDON)
    time.sleep(val)
def flashall (val):
    count = 0
    while count < val:
```

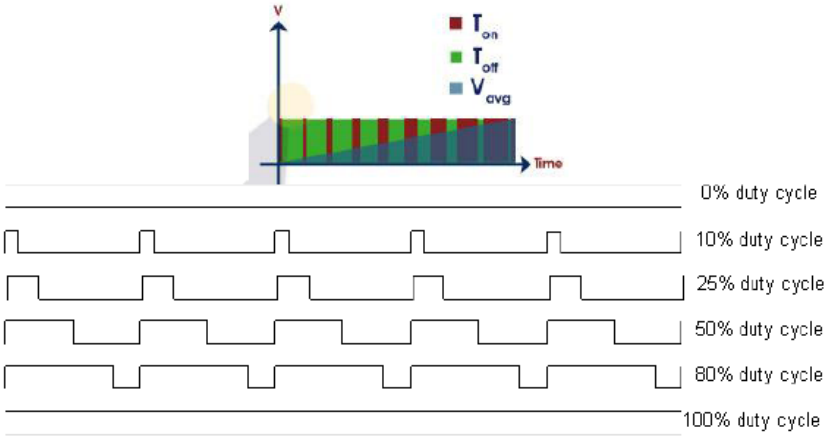
```
        allon(0.1)
        alloff(0.1)
        count += 1
def work (val):
    count = 0
    while count < val:
        red (3)
        green (3)
        yellow (1)
        count += 1
try:
  while True:
    setupgpio()
    alloff(0)
    flashall(5)
    work (5)
except KeyboardInterrupt:
  GPIO.cleanup()
  exit()
```

تقنية تعديل عرض النبضة PWM ضمن مكتبة RPi.GPIO

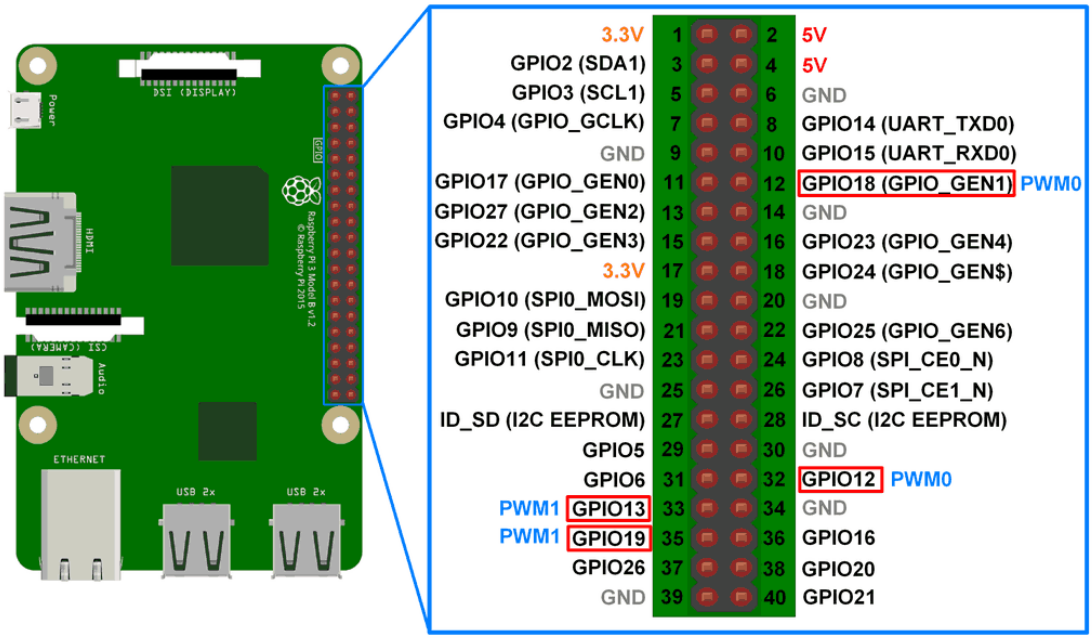
إن أقطاب التحكم ضمن لوحة راسبيري باي تعطي نبضات رقمية إما 3.3 فولط أو صفر فولط, لكن بعض المكونات الالكترونية تحتاج إلى قيم تشابهية للجهود, لذلك يمكن استخدام تقنية تعديل عرض النبضة التي تعتمد على مبدأ توليد نبضات بتردد معين, وبعرض متغير حسب قيمة الجهد المطلوب, حيث أنّ العنصر الالكتروني الذي نطبق على دخله إشارة pwm لا يتأثر بالنبضات وإنما بمتوسط قيمة التغير في هذه النبضات تبعا للزمن فيصبح فرق الجهد الناتج هو تكامل تغير زمن هذه النبضات والذي تتغير قيمته, وبالتالي يتغير فرق الجهد الناتج.

مثلا عندما تكون قيمة duty cycle عبارة عن 75% يكون الجهد 2.475.





تدعم لوحة راســـبيري باي نوعين من تقنية عرض النبضـــة Pulse-Width Modulation أحدهما PWM Software برمجي حيث يمكن تحويل أي قطب إلى قطب PWM واســـتخدام الطرق والتوابع الخاصـــة بذلك، والنوع الثاني PWM Hardware هو قطب خاص قطب رقم 12 وله طريقة خاصة للتحكم به.



طريقة توليد نبضات PWM ضمن RPi.GPIO

To create a PWM instance:

```
p = GPIO.PWM(channel, frequency)
```

To start PWM:

```
p.start(dc)   # where dc is the duty cycle (0.0 <= dc <= 100.0)
```

To change the frequency:

```
p.ChangeFrequency(freq)   # where freq is the new frequency in Hz
```

To change the duty cycle:

```
p.ChangeDutyCycle(dc)  # where 0.0 <= dc <= 100.0
```

To stop PWM:

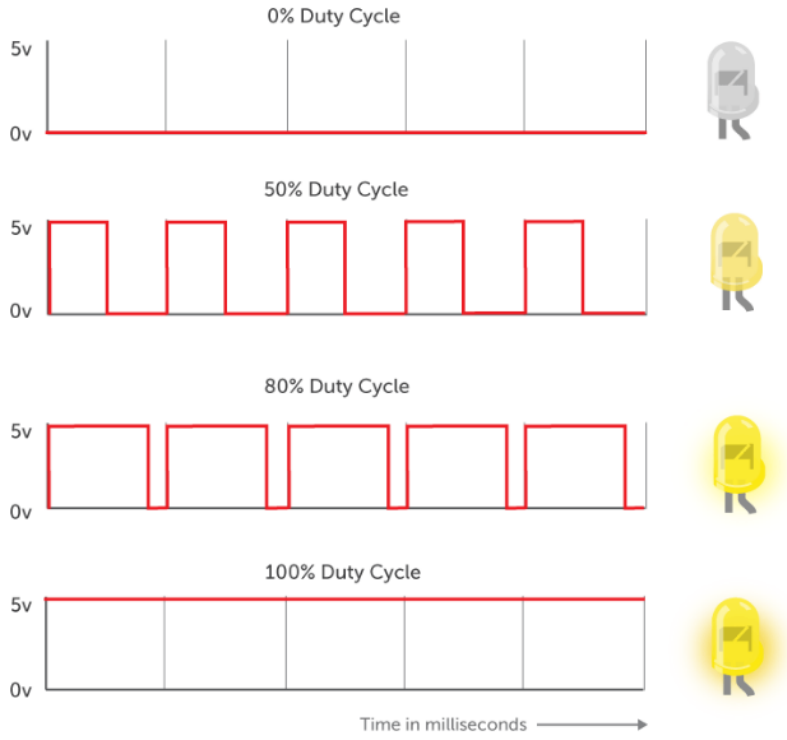
```
p.stop()
```

مثال 1:

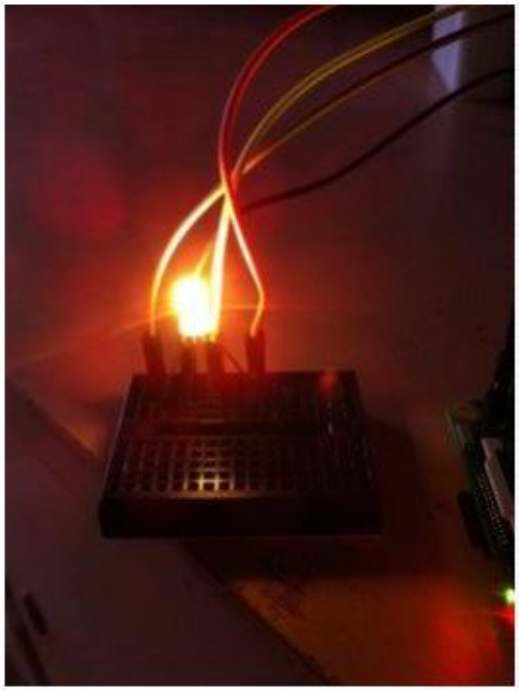
```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 0.5)
p.start(1)
input('Press return to stop:')   # use raw_input for Python 2
p.stop()
GPIO.cleanup()
```

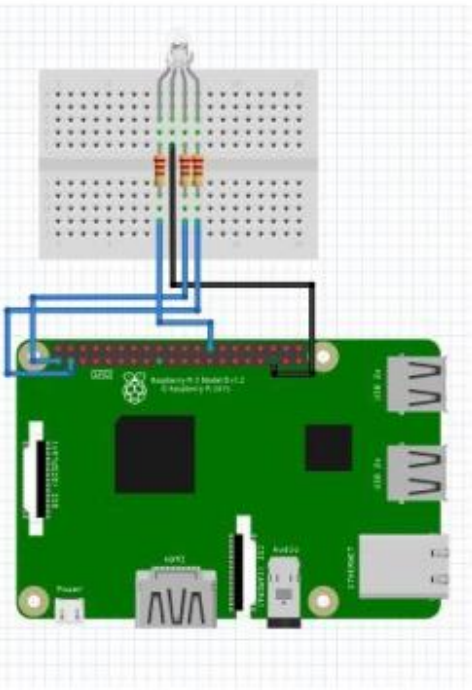
مثال 2: التحكم بشدة اضاءة LED

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 50)  # channel=12 frequency=50Hz
p.start(0)
try:
    while 1:
        for dc in range(0, 101, 5):
                            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```



التطبيق العملي: التحكم بإضاءة RGB LED





يجب الانتباه إلى المسافات tab في هذا البرنامج

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
RUNNING = True
green = 11
red = 12
blue = 13
GPIO.setup(red, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)
GPIO.setup(blue, GPIO.OUT)
Freq = 50
RED = GPIO.PWM(red, Freq)
GREEN = GPIO.PWM(green, Freq)
BLUE = GPIO.PWM(blue, Freq)
try:
while RUNNING:
RED.start(100)
GREEN.start(1)
BLUE.start(1)
for x in range(1,101):
GREEN.ChangeDutyCycle(x)
time.sleep(0.09)
for x in range(1,101):
```

```
RED.ChangeDutyCycle(101-x)
time.sleep(0.09)
for x in range(1,101):
GREEN.ChangeDutyCycle(101-x)
BLUE.ChangeDutyCycle(x)
time.sleep(0.09)
for x in range(1,101):
RED.ChangeDutyCycle(x)
time.sleep(0.09)
except KeyboardInterrupt:
RUNNING = False
GPIO.cleanup()
```