الوحدة الثانية: البرمجة بلغة بايثون

الربط بالحاسوب

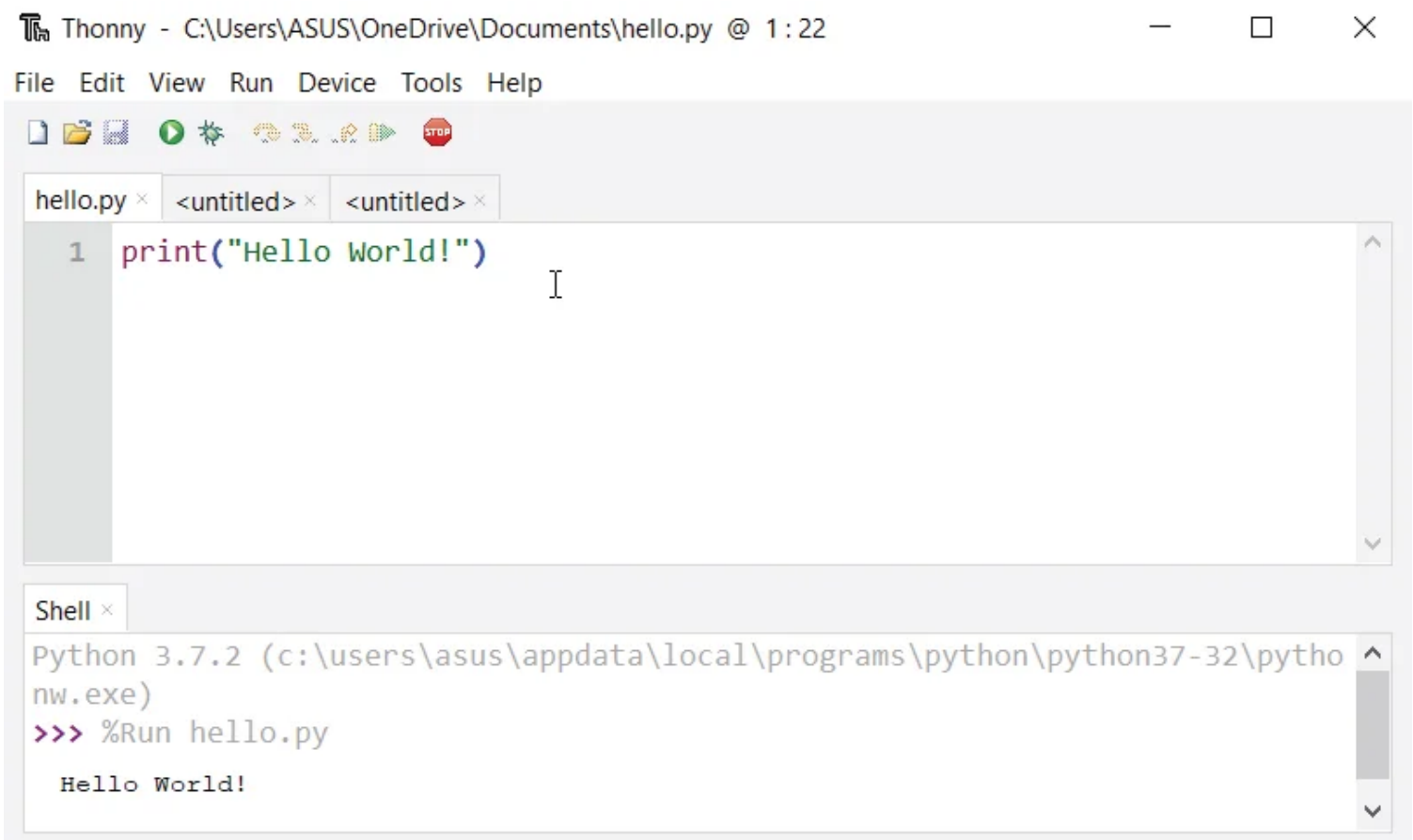
م. غنام الجعبري

عناصر لغة البرمجة

- تتألف لغة البرمجة بايثون من مجموعة من العناصر اهمها:
  - بناء البرنامج (Basic Syntax)
  - الكلمات المحجوزة (Keywords)
  - المتغيرات (Variables)
  - انواع البيانات (Data Types)
  - الدوال (Functions)
  - العوامل (Operators)
  - اتخاذ القرار (Decision Making)
  - الحلقات (Loops)

بناء البرنامج

- برنامج لطباعة عبارة "Hello, World!"



بناء البرنامج

- تستخدم لغة بايثون المسافات البادئة (indentation) في تحديد بداية الكود ونهايته بدلا من الاقواس المتعرجة { } في لغات البرمجة الاخرى
- يفضل استخدام 4 فراغات (whitespace) كمسافة بادئة بدلا من علامة التبويب (tab)

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

التعليقات

- تستخدم التعليقات (Comments) لتوضيح ما يقوم به جزء من الكود او لعدم تنفيذه
- يتم تجاهل التعليقات بالكامل في تنفيذ البرنامج
- في بايثون تبدأ التعليقات بالرمز # في بداية السطر

```
# This is a comment
# print out Hello
print('Hello')
```

الكلمات المحجوزة

- الكلمات المحجوزة (Keywords) هي كلمات معرفة مسبقا لا يمكن استخدامها في تسمية المتغيرات (variables) والدوال (functions) لانها جزء من بناء البرنامج

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

المعرفات

- المعرفات (Identifiers) هي اسماء لتعريف المتغيرات والدوال والفئات (classes)
- في تعريف المتغيرات يجب اتباع القواعد التالية:
  - ان يحتوي المعرف على احرف صغيرة او كبيرة او ارقام وعلامة الشرطة السفلية فقط
  - ان لا يبدأ المعرف برقم وانما حرف او علامة الشرطة السفلية
  - ان لا يكون المعرف احد الاسماء المحجوزة
  - لا يوجد تحديد لطول المعرف
- لغة بايثون من اللغات المميزة لحالة الحرف (case-sensitive) اي ان كلمة number تختلف عن Number عند تسمية المتغيرات

المتغيرات والثوابت

- المتغيرات (Variables) هي مواقع في الذاكرة لتخزين القيم
- الثوابت (Constants) هي متغيرات قيمتها ثابتة مثل ثابت الدائرة (Pi) وثابت الجاذبية
- يفضل في تسمية المتغيرات والثوابت ان تكون الاسماء ذات معنى واستخدام الشرطة السفلية في فصل الاسم المكون من اكثر من كلمة واستخدام الاحرف الكبيرة في تسمية الثوابت

```
number = 10
name = "Ali"
```

```
PI = 3.14
GRAVITY = 9.8
```

انواع البيانات

- العدد الصحيح (Integer): عدد بدون كسور عشرية
- العدد الحقيقي (Float): عدد مع كسور عشرية
- السلسلة (String): نص محاط بعلامتي اقتباس مفردة (') او مزودجة (")
- القيم المنطقية (Boolean): قيمة صائبة (True) او خاطئة (False)

```
age = 42
price = 76.5
message = "Hello, Python!"
is_raining = True
```

انواع البيانات

- تتضمن لغة بايثون انواع اخرى من البيانات تشمل القائمة (List) والمجموعة (Tuple) والمجموعة الجزئية (Set) والدليل (Dictionary)

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)
my_set = {1, 2, 3}
my_dict = {1: 'apple', 2: 'ball'}
```

الدوال

- تحتوي لغة بايثون على مجموعة من الدوال التي تم بناؤها مسبقا (built-in functions)
- من الدوال القياسية التي تستخدم في عمليات الادخال والاخراج دالة ()print ودالة ()input

```
print("Hello, Python!")
print(42)
print("Value:", 3.14)
```

```
name = input("Enter your name: ")
age = input("Enter your age: ")
```

الوحدات النمطية

- تستخدم الوحدات النمطية (Modules) في تقسيم البرنامج الى وحدات اصغر حجما
- الوحدة النمطية هي ملف بايثون يحتوي على تعريفات ودوال يمكن استدعاؤها في البرنامج او في وحدة نمطية اخرى
- لاضافة وحدة نمطية الى البرنامج نستخدم كلمة import، ولاستدعاء بعض التعريفات او الدوال في الوحدة النمطية نستخدم كلمة from

```
import math
print("The value of pi is", math.pi)
```

```
import math as m
print("The value of pi is", m.pi)
```

```
from math import pi
print("The value of pi is", pi)
```

العوامل

- العوامل الحسابية:
  - الجمع (+)
  - الطرح (-)
  - الضرب (*)
  - قسمة الاعداد الحقيقية (/)
  - قسمة الاعداد الصحيحة (//)
  - باقي القسمة (%)
  - القوة او الأس (**)

```
x = 15
y = 4

print('x + y =', x+y)
# Output: x + y = 19

print('x - y =', x-y)
# Output: x - y = 11

print('x * y =', x*y)
# Output: x * y = 60

print('x / y =', x/y)
# Output: x / y = 3.75

print('x // y =', x//y)
# Output: x // y = 3

print('x % y =',x%y)
# Output: x % y = 3

print('x ** y =', x**y)
# Output: x ** y = 50625
```

العوامل

- عوامل المقارنة:
  - اكبر من (>)
  - اصغر من (<)
  - يساوي (==)
  - لا يساوي (!=)
  - اكبر من او يساوي (>=)
  - اصغر من او يساوي (<=)

```
x = 10
y = 12

print('x > y is', x>y)
# Output: x > y is False

print('x < y is', x<y)
# Output: x < y is True

print('x == y is', x==y)
# Output: x == y is False

print('x != y is', x!=y)
# Output: x != y is True

print('x >= y is', x>=y)
# Output: x >= y is False

print('x <= y is', x<=y)
# Output: x <= y is True
```

العوامل

- العوامل المنطقية:
  - منطق و (and)
  - منطق أو (or)
  - منطق النفي (not)

- عوامل التعيين:
  - التعيين (=)
  - التعيين مع الجمع (+=)
  - التعيين مع الطرح (-=)
  - التعيين مع الضرب (*=)
  - التعيين مع القسمة (/=)

```
x = True
y = False

print('x and y is', x and y)
# Output: x and y is False

print('x or y is', x or y)
# Output: x or y is True

print('not x is', not x)
# Output: not x is False
```

```
x = 10
x += 5  # Equivalent to x = x + 5
x -= 5  # Equivalent to x = x - 5
x *= 5  # Equivalent to x = x * 5
x /= 5  # Equivalent to x = x / 5
```

جملة if...else

- تستخدم جملة if...else في اتخاذ القرار
- تحتوي جملة if على جملة اختيارية هي else
- تختبر جملة if...else الشرط قبل تنفيذ الكود
  - اذا كان الشرط صائبا (True) يتم تنفيذ الكود داخل جملة if وتجاوز الكود داخل جملة else
  - اذا كان الشرط خاطئا (False) يتم تجاوز الكود داخل جملة if وتنفيذ الكود داخل جملة else

```
num = 3
if num > 0:
    print(num, "is a positive number.")
print("This is always printed.")
```

```
num = 3
if num >= 0:
    print("Positive or Zero")
else:
    print("Negative number")
```

جملة if...elif...else

- تستخدم جملة if...elif...else في اختبار اكثر من شرط
- كلمة elif هي اختصار للكلمتين else if

```
num = 3.4

if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

حلقة for

- تستخدم حلقة for لتكرار الكود في الحلقة عدد معين من المرات
- يمكن توليد سلسلة من الاعداد في بايثون باستخدام دالة ()range
- تستخدم حلقة for في استعراض عناصر سلسلة مثل قائمة (list) او مجموعة (tuple)

```
for i in range(10):
    print(i)
```

```
for i in range(1, 10):
    print(i)
```

```
for i in range(1, 10, 2):
    print(i)
```

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0
for x in numbers:
    sum = sum + x
print("The sum is", sum)
```

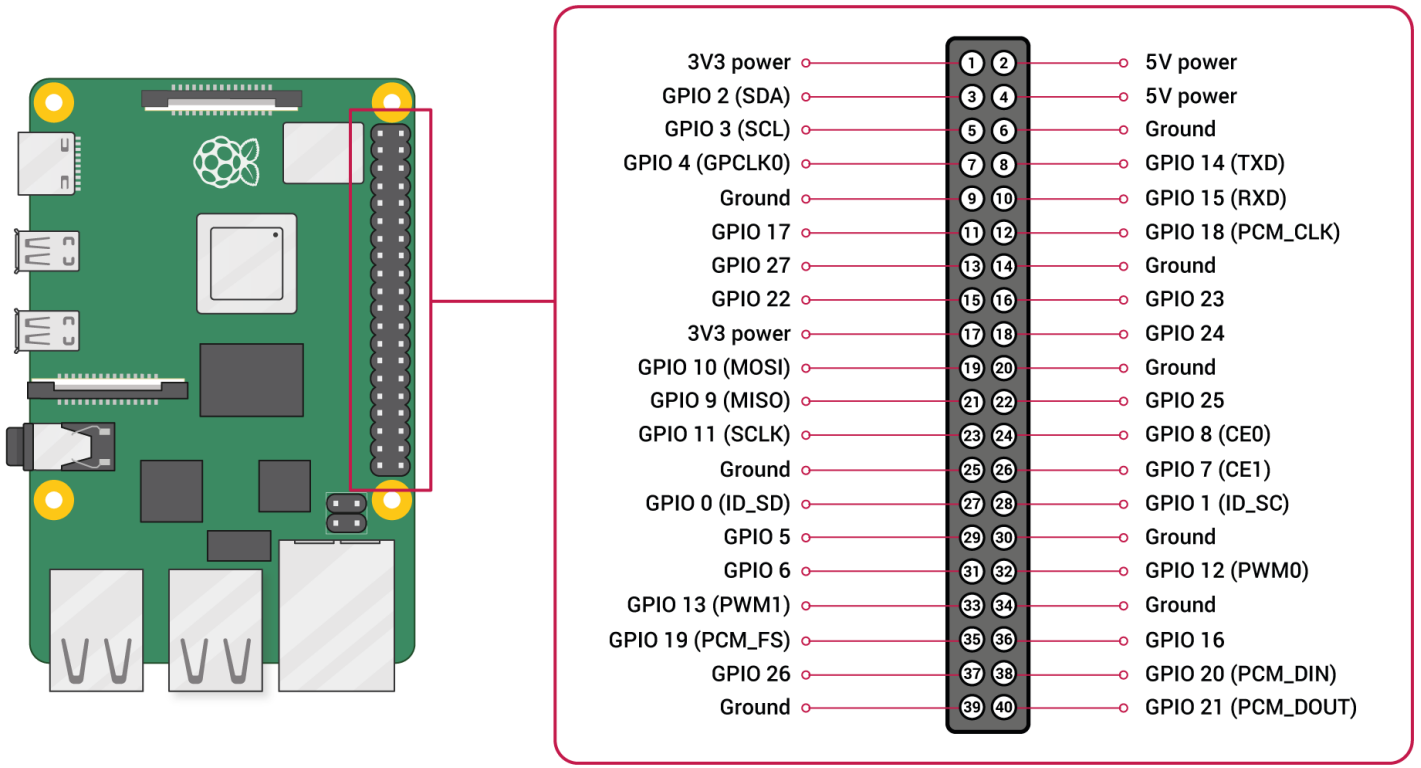
حلقة while

- تستخدم حلقة while لتكرار الكود في الحلقة طالما الشرط يتحقق اي في حالة عدم معرفة عدد المرات
- تختبر حلقة while الشرط قبل تنفيذ الكود
  - اذا كان الشرط صائبا ينفذ الكود في الحلقة ويقيم الشرط مرة أخرى قبل تكرار الحلقة
  - اذا كان الشرط خاطئا يتم انهاء الحلقة
  - يتكرر تنفيذ الكود في الحلقة حتى يصبح الشرط خاطئا

```
sum = 0
i = 1

while i <= 10:
    sum = sum + i
    i = i + 1

print("The sum is", sum)
```

برمجة راسبيري باي

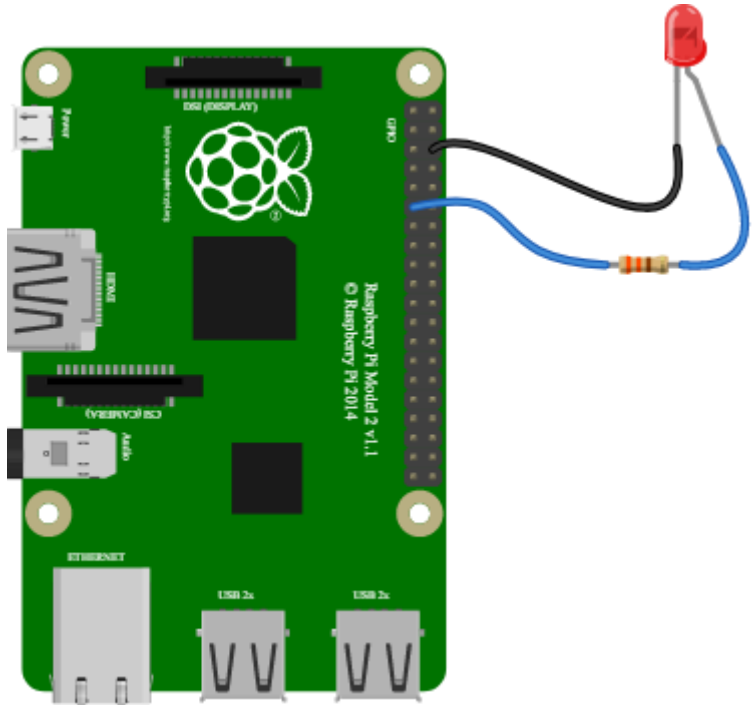
- ابر الادخال والاخراج (GPIO) في لوحة راسبيري باي التي تحتوي على 40 ابرة



مثال: غمّاز

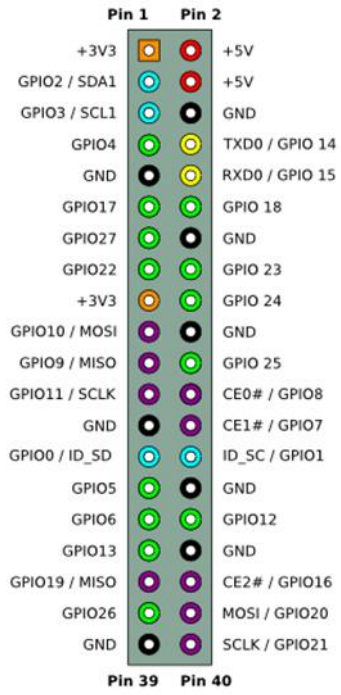
- برنامج لتشغيل واطفاء دايود ضوئي بشكل منتظم (blinking LED)

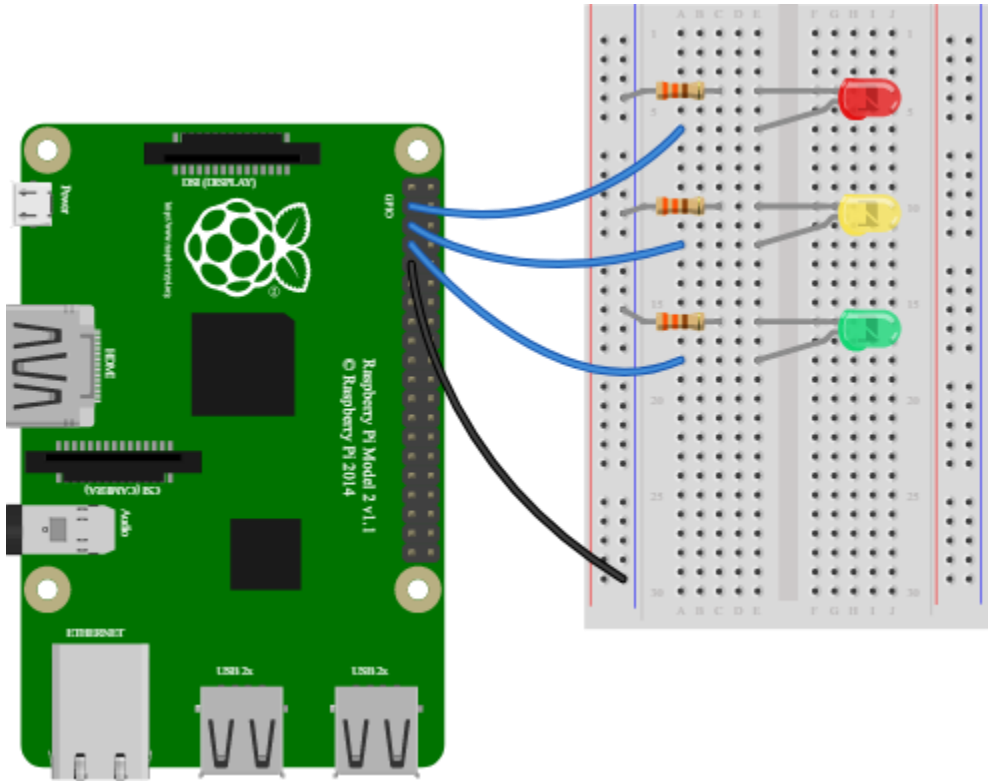
```
import RPi.GPIO as GPIO
import time

LED = 17

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)

while True:
    GPIO.output(LED, GPIO.HIGH)
    time.sleep(1)

    GPIO.output(LED, GPIO.LOW)
    time.sleep(1)

GPIO.cleanup()
```



تمرين: الاشارة الضوئية

- اكتب برنامج لتشغيل الاشارة الضوئية باستخدام لوحة راسبيري باي؛ تشغيل الدايود الاحمر لمدة 5 ثوان ثم الدايود الاصفر لمدة ثانية ثم الدايود الاخضر لمدة 5 ثوان ثم الدايود الاصفر لمدة ثانية؟

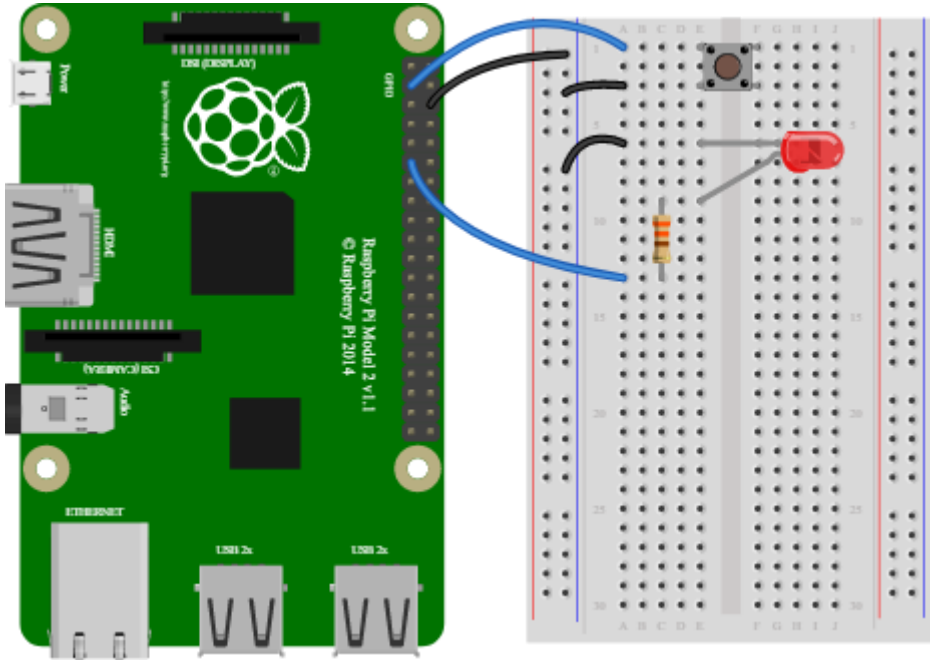




مثال: زر تشغيل

- برنامج لتشغيل دايود ضوئي عند الضغط على الزر

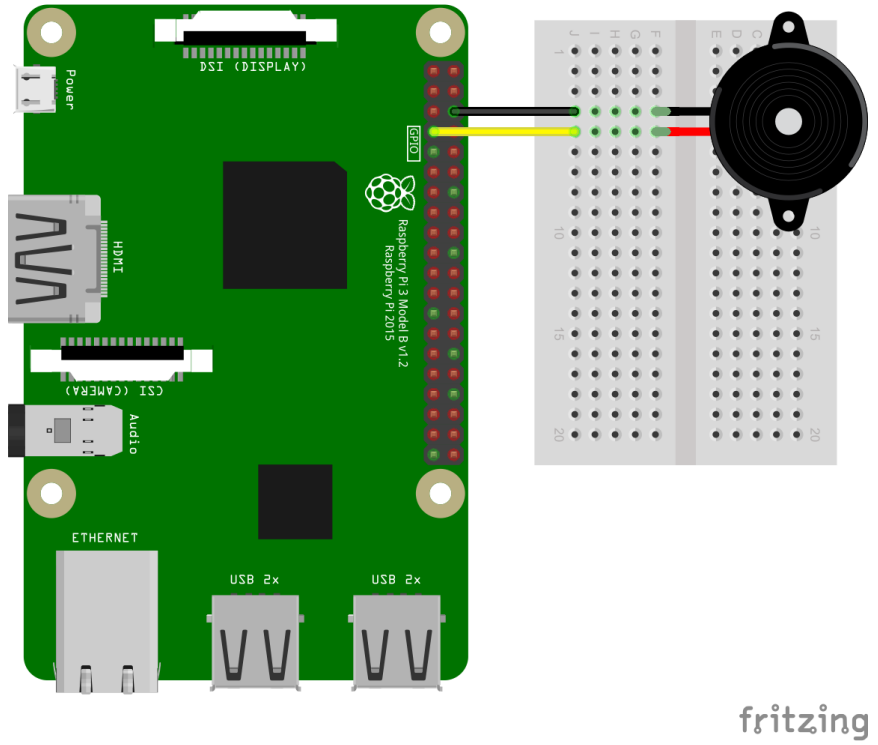
```
import RPi.GPIO as GPIO
import time

LED = 17
BUTTON = 2
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
GPIO.setup(BUTTON, GPIO.IN)

while True:
    if GPIO.input(BUTTON) == GPIO.HIGH:
        GPIO.output(LED, GPIO.HIGH)
        time.sleep(0.2)
    else:
        GPIO.output(LED, GPIO.LOW)
        time.sleep(0.2)
GPIO.cleanup()
```



مثال: تنبيه صوتي

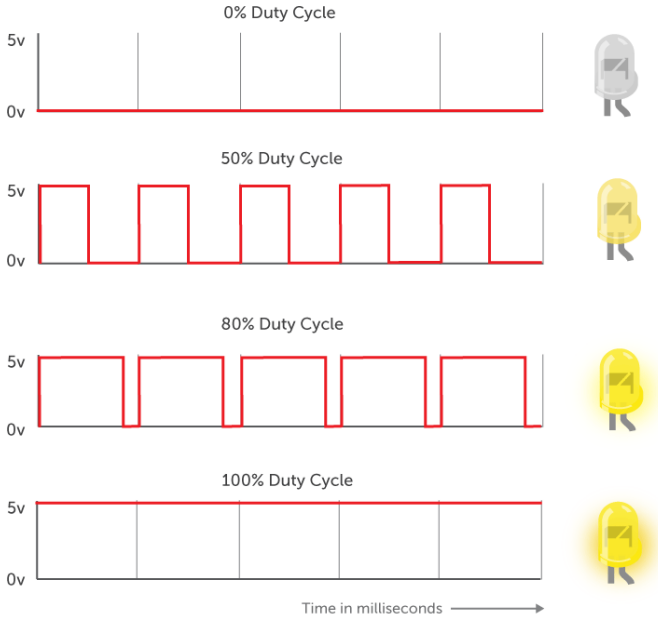
- برنامج لتشغيل تنبيه صوتي باستخدام Buzzer

```
import RPi.GPIO as GPIO
import time

buzzer = 17

GPIO.setmode(GPIO.BCM)
GPIO.setup(buzzer, GPIO.OUT)

while True:
    GPIO.output(buzzer, GPIO.HIGH)
    time.sleep(1)

    GPIO.output(buzzer, GPIO.LOW)
    time.sleep(1)

GPIO.cleanup()
```



تقنية PWM

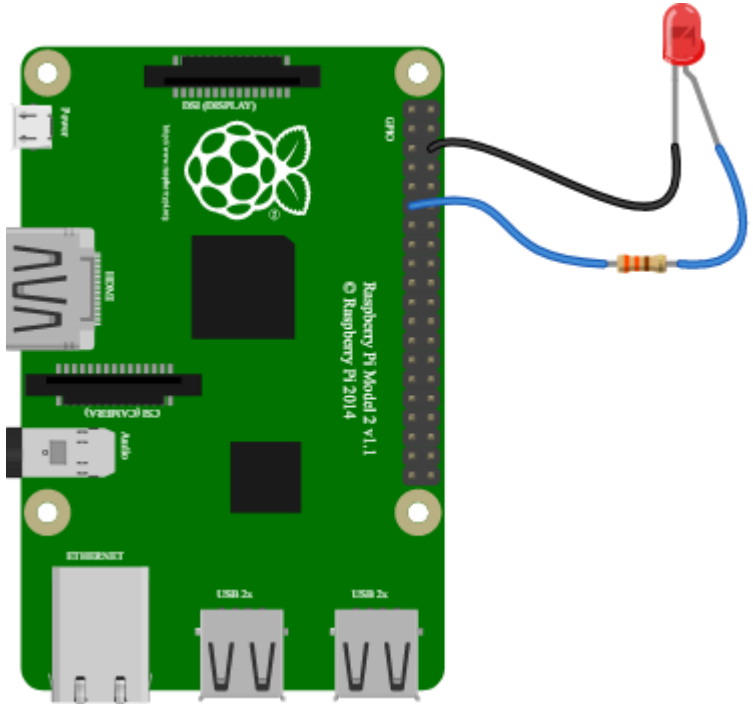
- تستخدم تقنية PWM في تحديد القيمة التماثلية عن طريق التحكم بعرض النبضة الكهربائية او الوقت الذي تكون به الاشارة مرتفعة في الدورة الواحدة
- دورة التشغيل (Duty Cycle) تمثل نسبة الوقت الذي تكون به الاشارة مرتفعة في الدورة الواحدة مثلا 50% تعني ان الاشارة تكون مرتفعة خلال نصف الوقت في الدورة الواحدة



مثال: شدة الاضاءة

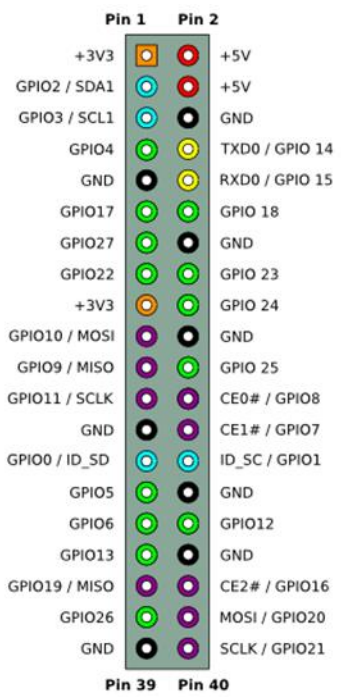
- برنامج لزيادة شدة الاضاءة تدريجيا (LED dimming) باستخدام تقنية PWM

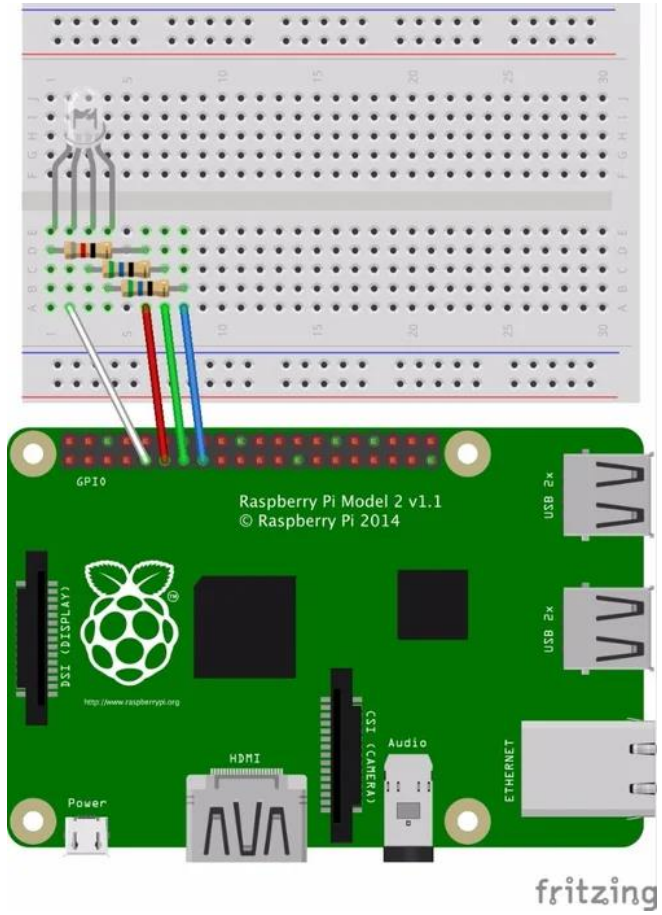
```
import RPi.GPIO as GPIO
import time

LED = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)

pwm = GPIO.PWM(LED, 50)  # frequency=50Hz
pwm.start(0)

while True:
   for duty_cycle in range(0, 101, 5):
      pwm.ChangeDutyCycle(duty_cycle)
      time.sleep(0.1)

pwm.stop()
GPIO.cleanup()
```



تمرين: دايود RGB

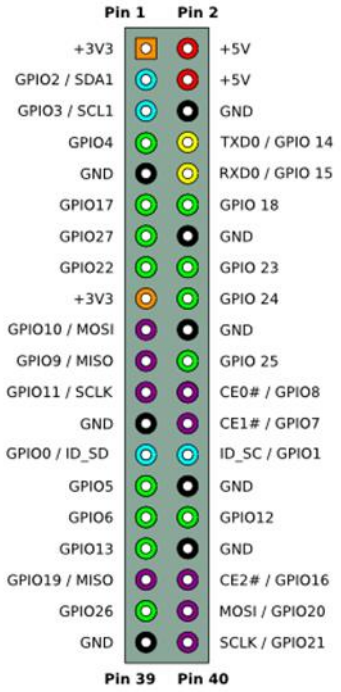
- اكتب برنامج لتشغيل الدايود الضوئي الملون (RGB LED) باستخدام تقنية PWM

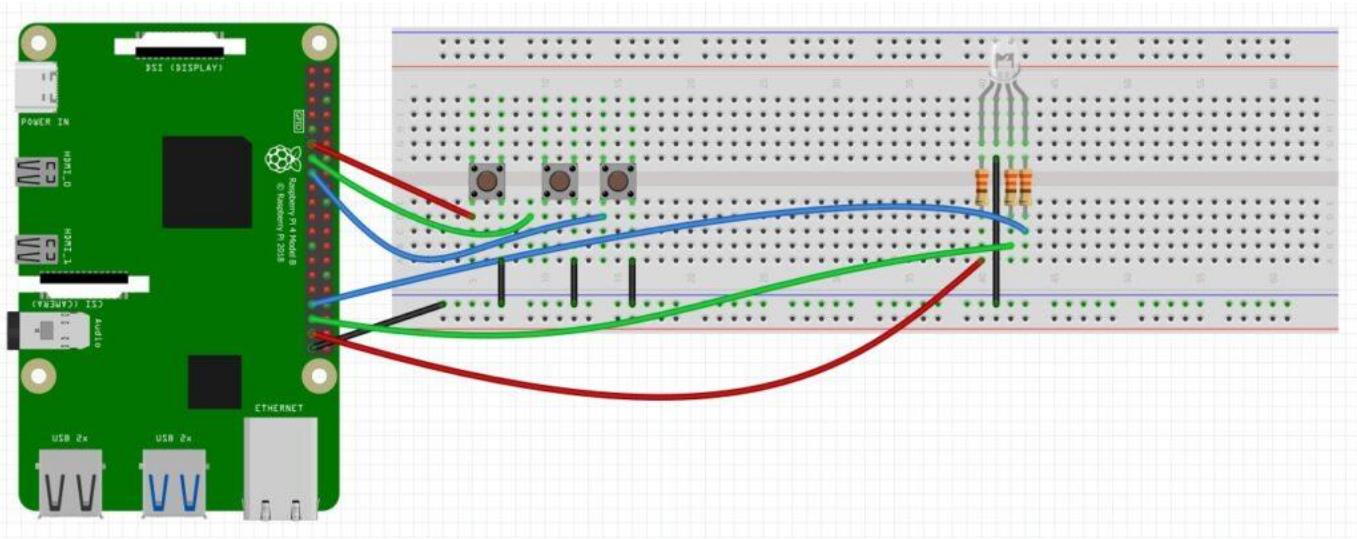




تمرين: التحكم باللون

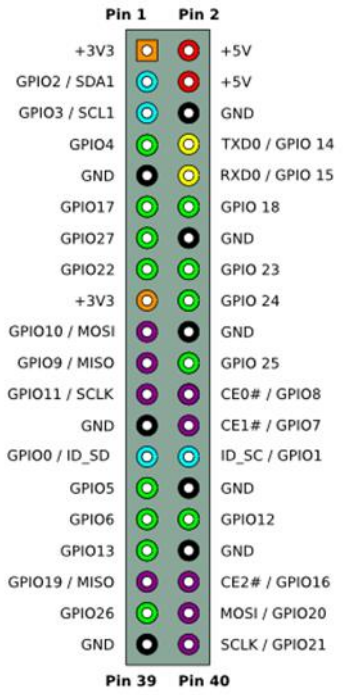
- اكتب برنامج للتحكم باللون في الدايود الضوئي الملون (RGB LED) من خلال ازرار التشغيل: زر لتشغيل اللون الاحمر، زر لتشغيل اللون الاخضر، زر لتشغيل اللون الازرق

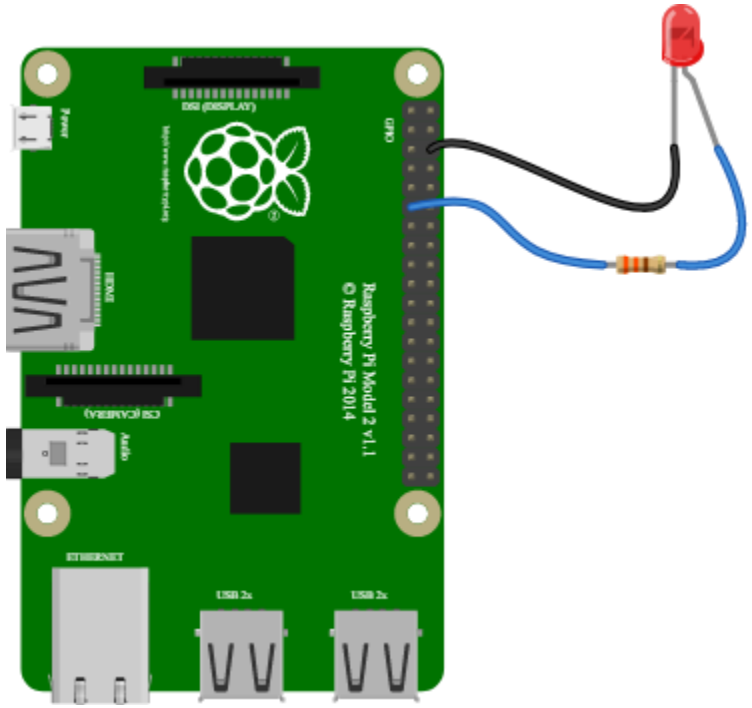




تمرين: شدة الاضاءة

- اكتب برنامج لزيادة شدة الاضاءة تدريجيا ثم انقاص شدة الاضاءة تدريجيا باستخدام PWM

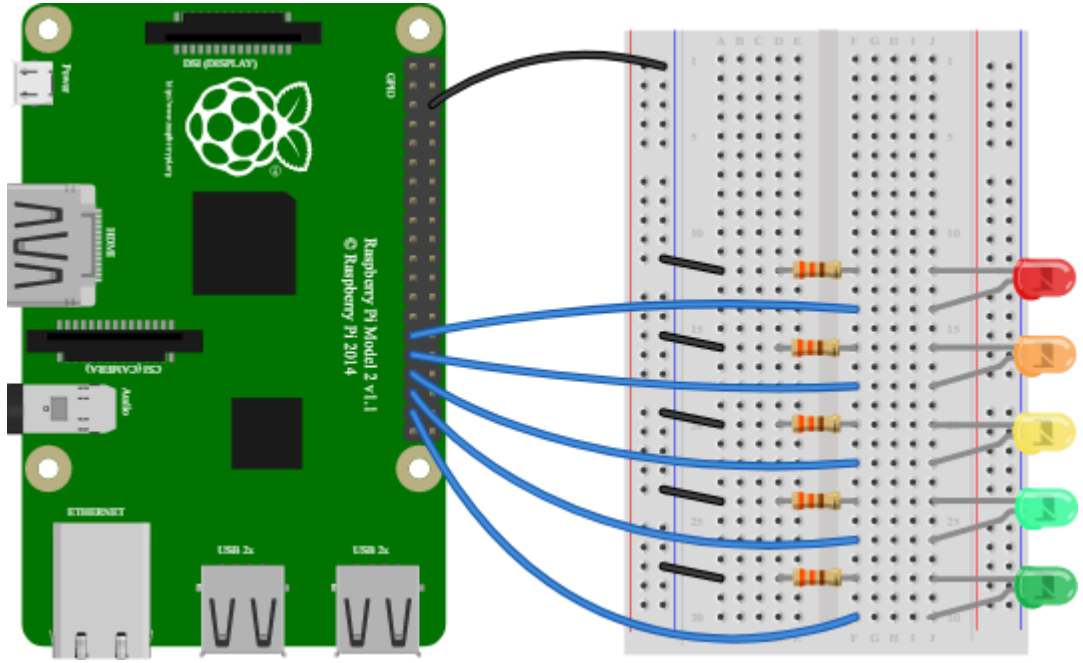




مثال: انارة متتابعة

- برنامج لانارة مجموعة من الدايودات الضوئية بشكل متتابع وبفاصل زمني 1 ثانية

```
import RPi.GPIO as GPIO
import time

leds = (5, 6, 13, 19, 26)

GPIO.setmode(GPIO.BCM)
for led in leds:
    GPIO.setup(led, GPIO.OUT)

while True:
    for led in leds:
        GPIO.output(led, GPIO.HIGH)
        time.sleep(1)
    for led in leds:
        GPIO.output(led, GPIO.LOW)
    time.sleep(1)
GPIO.cleanup()
```



تمرين: انارة متتابعة

- اكتب برنامج لانارة مجموعة من الدايودات الضوئية بشكل متتابع ثم اطفاء المجموعة بشكل متتابع وبفاصل زمني 1 ثانية؟