Bengkel Autonomous Robot Application : Donkey Car (Part 3)

Nur Akhyar bin Nordin

Part Time Makers

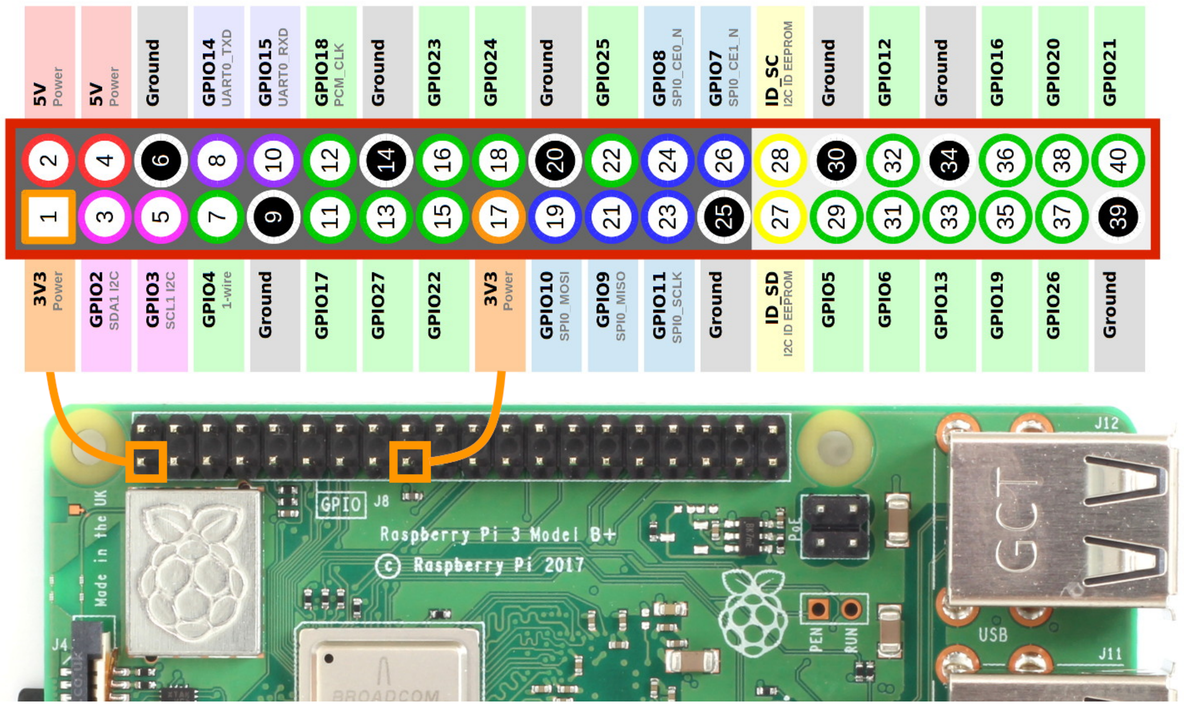
ayozzet

Python Introduction

Understanding RPi characteristic and specification

1. RPi build specification

   Raspberry Pi 3 Specifications
   SoC: Broadcom BCM2837
   CPU: 4× ARM Cortex-A53, 1.2GHz
   GPU: Broadcom VideoCore IV
   RAM: 1GB LPDDR2 (900 MHz)
   Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
   Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
   Storage: microSD
   GPIO: 40-pin header, populated
   Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)
2. Processor, Memory and I/O
3. GPIO and their functionality



Python library setup

1. Updating and upgrading systems
2. Installing appropriate library for certain task
3. Understanding type of library installation

Python exercise and some electronic fundamentals

1. Logic signal
2. Blinking the LED (digital output)
3. Turn on LED by button (digital input)
4. Potentiometer/Ultrasonic into terminal print (analog input)
5. DC Motor using L293N (PWM generation)

Python Exercise

1.0 Introduction to Python Programming

1.1 Remember features of Python environment

a. Simple - no need specific IDE to run
b. Free and Open Source - no charge, easy download
c. High Level Language - can be understable by human / English look-alike
d. Portable - Can be run on any platform (Win/Linux/MacOS)
e. Interpreted - because it goes through an interpreter which turn code into computer language
f. Object Oriented - python creating and using classes and objects are downright easy
g. Extensible - can write other language (eg: C++) in Python code
h. Embeddable - can put system into different states, set configurations and test all sort of real-world use cases
i. Extensive Libraries - huge collection libraries all over internet

1.2 Install Python

1.2.1 Describe Python installation on different platforms

Windows - 7, 8.x, 10
Linux - Raspberry Pi, Odroid, so many SBC on markets
Mac - OS X, MacOS
Others - AIX, IBM i, iOS, iPadOS, OS/390, z/OS, Solaris, VMS, HP-UX

1.2.2 Write a simple Python program

```
# This program prints Hello, world!

print('Hello, world!')
```

This is just an example programming using Python 3.x

1.3 Understand syntax in Python

1.3.1 Identify the basic syntax in Python

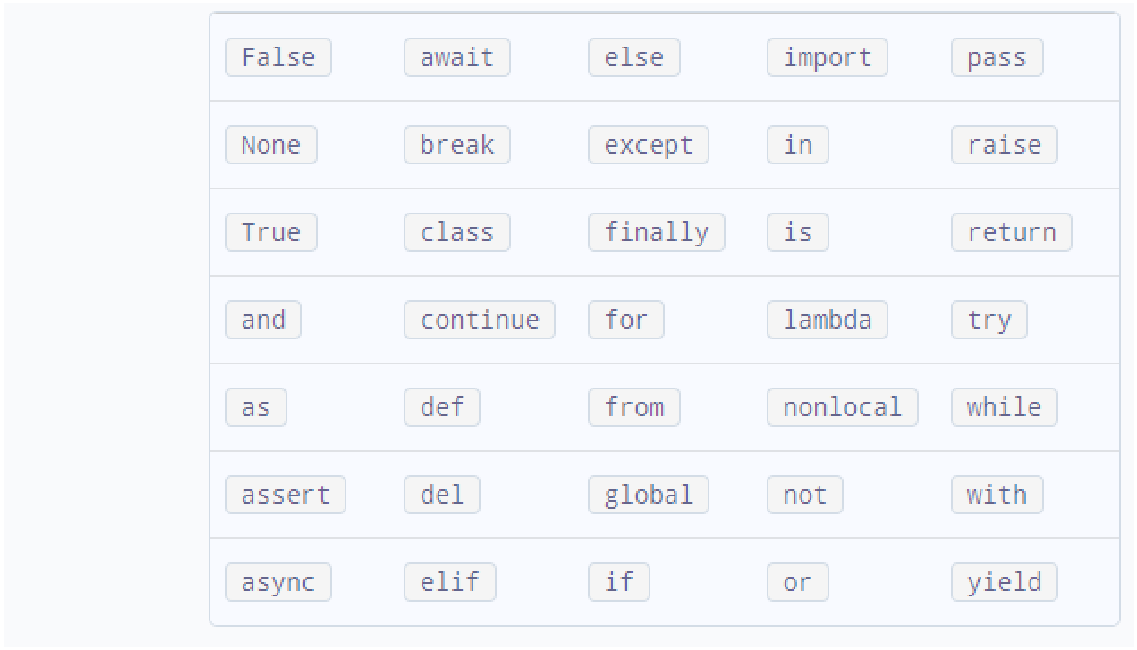
a. Identifier - An identifier is a name given to entities like class, functions, variable, etc. It helps to differentiate one entity from another

<u>Rules for writing identifiers</u>

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid examples.
2. An identifier cannot start with a digit. `1variable` is invalid, but `variable1` is a valid name.
3. Keywords cannot be used as identifiers.
   33 keywords in Python 3.7. All the keywords except `True`, `False` and `None` are in lowercase and they must be written as they are.

4. Cannot use special symbols like !, @, #, $, % etc. in our identifier.
5. An identifier can be of any length.
6. Python is a case-sensitive language.
7. Using underscore for multiple words is allowed but can't use it as the first character. This is because to differentiate either it is a docstring or not.

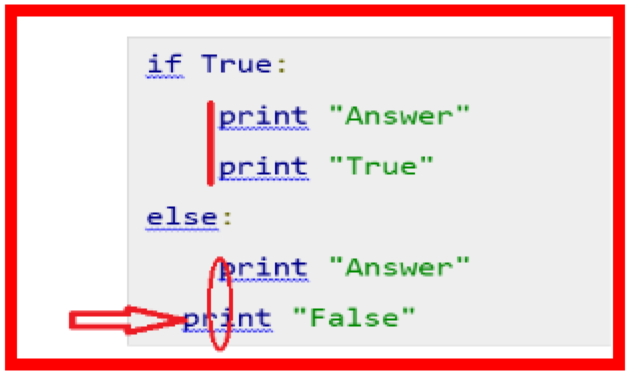
b. Reserved Words



`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`

Additional : exec,

c. Lines and Indentation - to define a block of code
Generally using four (4) whitespaces OR tabs.



```
if True:
    print "Answer"
    print "True"
else:
    print "Answer"
  print "False"
```

<-- Error example

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

Also can be ignored in line continuation.

```
if True:
    print('Hello')
    a = 5
```

```
if True: print('Hello'); a = 5
```

d. Multi-line statement - to make it easier to write and read if the statements are too long
Example 1 : using backslash `\`

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

Example 2 : using parentheses `( )` , brackets `[ ]` and braces `{ }`

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

```
colors = ['red',
          'blue',
          'green']
```

e. Quotation - can be use as comment or also represent as string

```
print('Hello')
```

Single Quote ' OR Double Quote "

f. Comments - as a reference but not included into our created programming code

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

Multiple line of comment example:

```
#This is a long comment
#and it extends
#to multiple lines
```

<--using **#**

```
"""This is also a
perfect example of
multi-line comments"""
```

<-- using **'''** or **"""**

g. Multiple statements - similar to multi-line statements but this is how to make multiple line statements into a single line.

```
a = 1; b = 2; c = 3
```

h. Docstring - https://www.programiz.com/python-programming/statement-indentation-comments

1.4 Identify basic structure in Python

1.4.1 Explain the data types in Python

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`

a. Numbers

int - positive or negative whole numbers (without a fractional part)
float - any real number with floating point
complex - a number with a real and imaginary component

```
a = 0b1010 #Binary Literals
b = 100 #Decimal Literal
c = 0o310 #Octal Literal
d = 0x12c #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2

#Complex Literal
x = 3.14j

print(a, b, c, d)
print(float_1, float_2)
print(x, x.imag, x.real)
```

Output:

```
10 100 200 300
10.5 150.0
3.14j 3.14 0.0
```

b. Strings - a collection of one or more characters put in single **'___'** , double **"___"** or triple quotes **'''___'''**

```
strings = "This is Python"
char = "C"
multiline_str = """This is a multiline string with more than one line code."""
unicode = u"\u00dcnic\u00f6de"
raw_str = r"raw \n string"

print(strings)
print(char)
print(multiline_str)
print(unicode)
print(raw_str)
```

Output :

```
This is Python
C
This is a multiline string with more than one line code.
Ünicöde
raw \n string
```

c. List - A collection which is ordered and **changeable**. Allow duplicate members. Items separated by commas **;** are enclosed within brackets **[ ]**. All items in a list do not to be of the same type.

```
a = [1, 2.2, 'python']
```

Method	**Description**
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

d. Tuple - A collection which is ordered and **unchangeable**. Allow duplicate members. Items separated by commas **;** are enclosed within brackets **( )**. All items in a list do not to be of the same type. Used to write-protect data and are usually faster than list as they cannot change dynamically.

```
t = (5,'program', 1+3j)
```

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

e. Dictionary - A collection which is unordered, changeable and indexed. No duplicate members. Defines within braces **{ }** with each item being a pair in the form `key:value`. Key and value can be of any type.

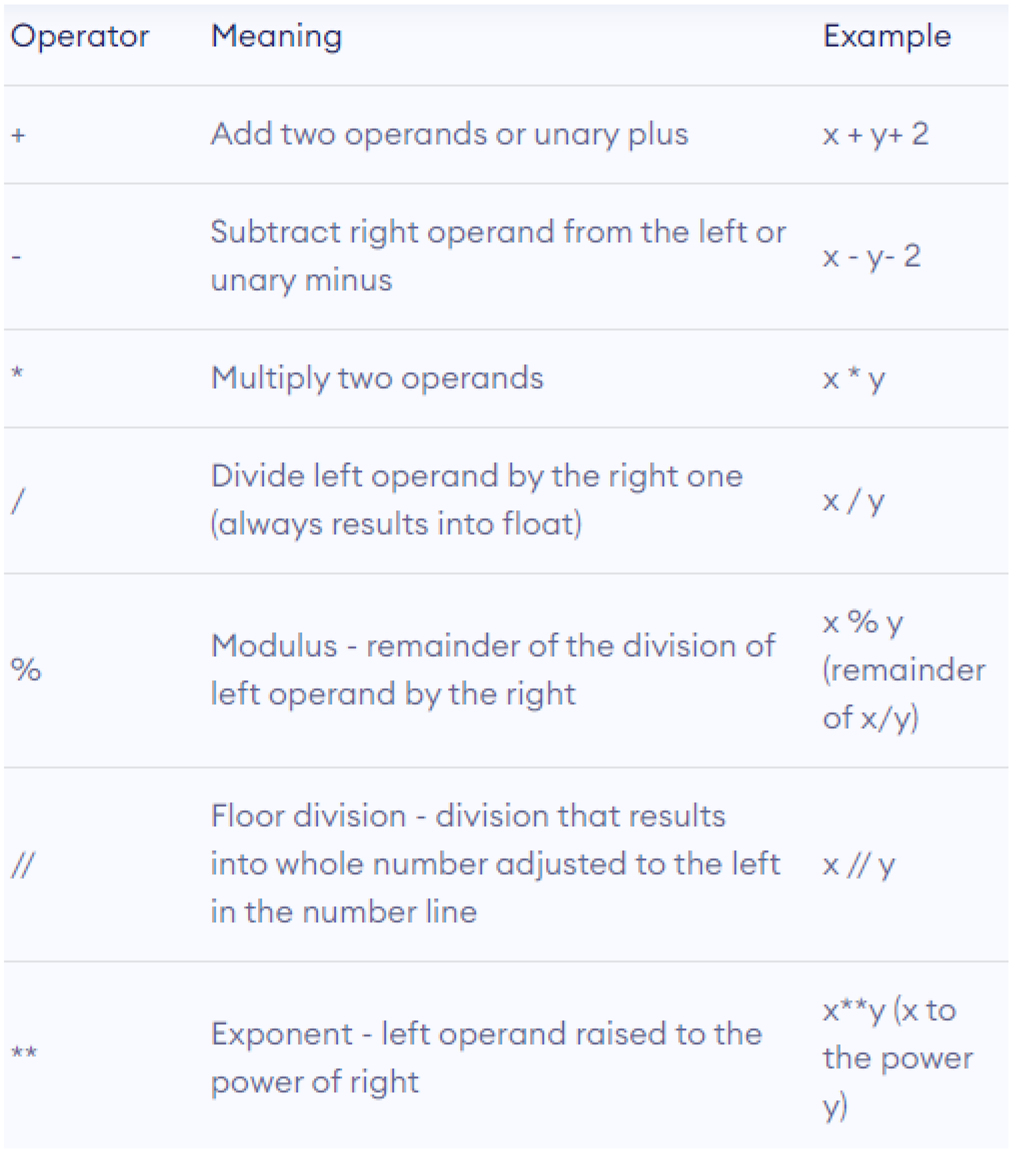
```
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1]);

print("d['key'] = ", d['key']);

# Generates error
print("d[2] = ", d[2]);
```

Output:

```
<class 'dict'>
d[1] =  value
d['key'] =  2
Traceback (most recent call last):
  File "<string>", line 9, in <module>
KeyError: 2
```

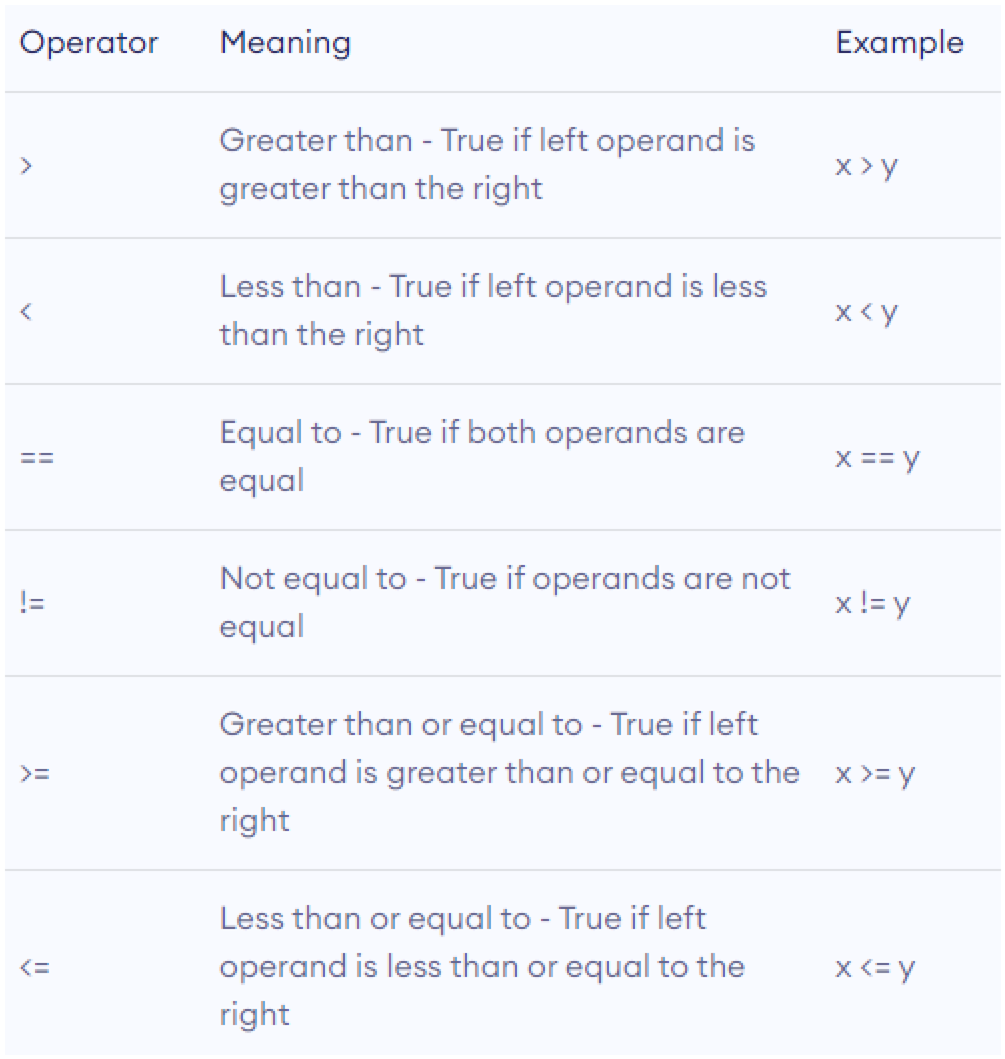
1.4.2 Explain basic operators in Python

a. Arithmetic Operators



Operator	Meaning	Example
+	Add two operands or unary plus	x + y+ 2
-	Subtract right operand from the left or unary minus	x - y- 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results into whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

b. Comparison / Relational Operators

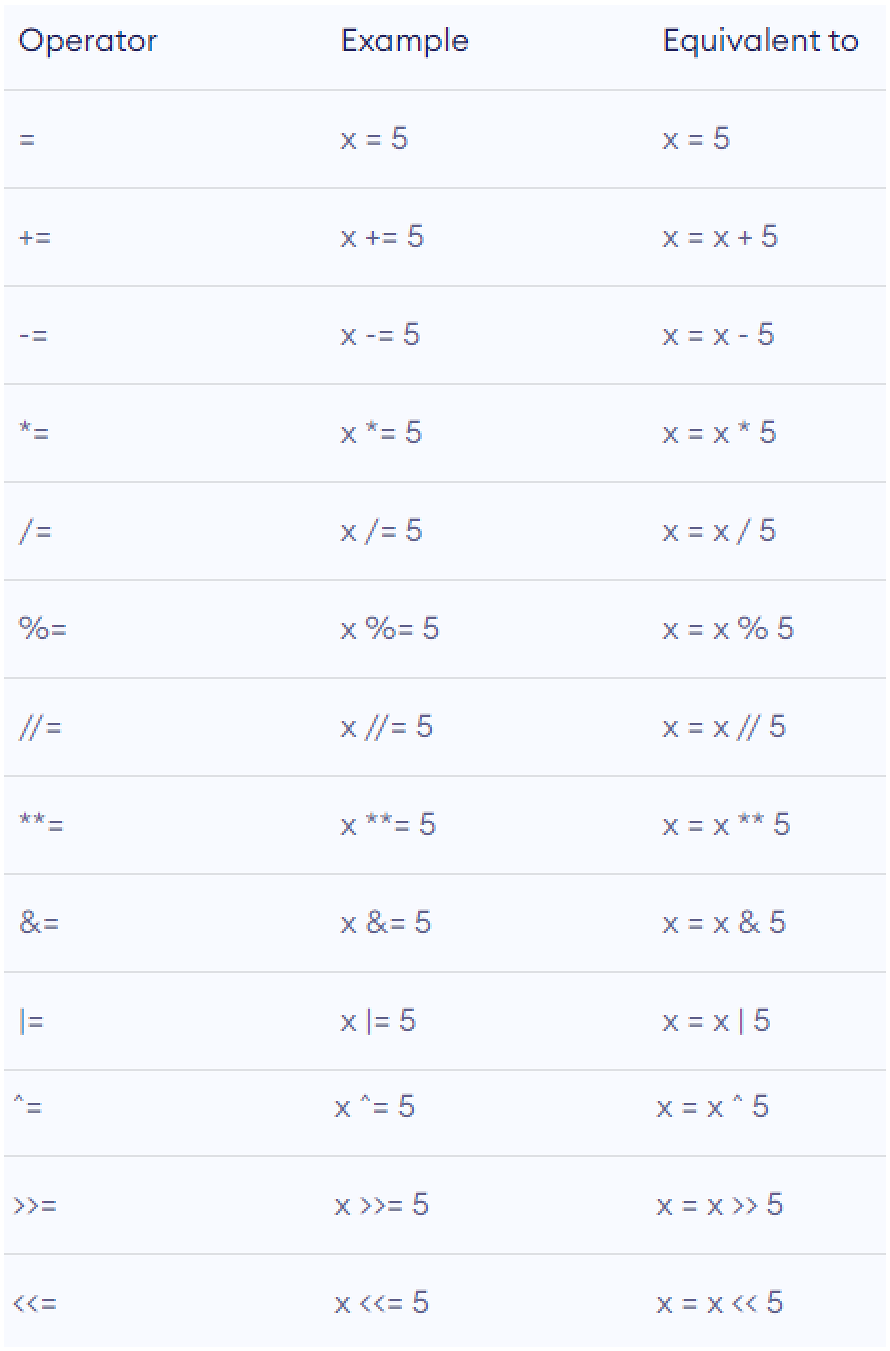


Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

Example:

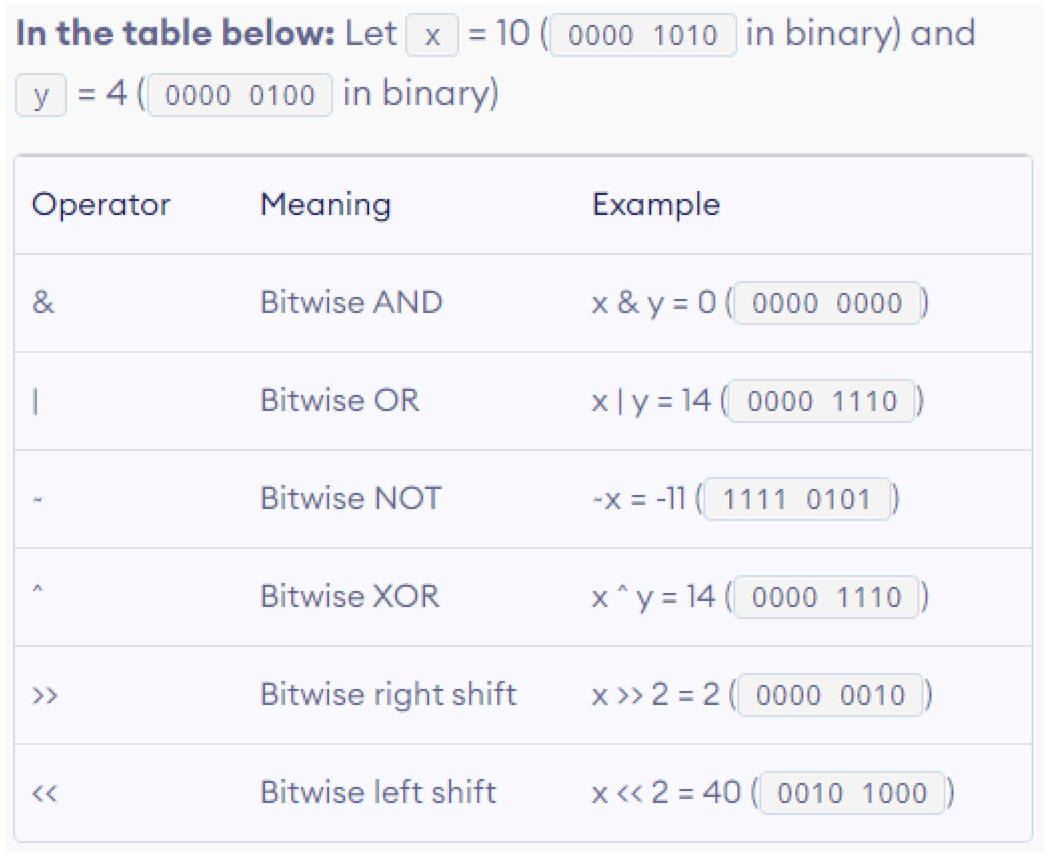
```
x = 10
y = 12

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

c. Assignment Operators



Operator	Example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

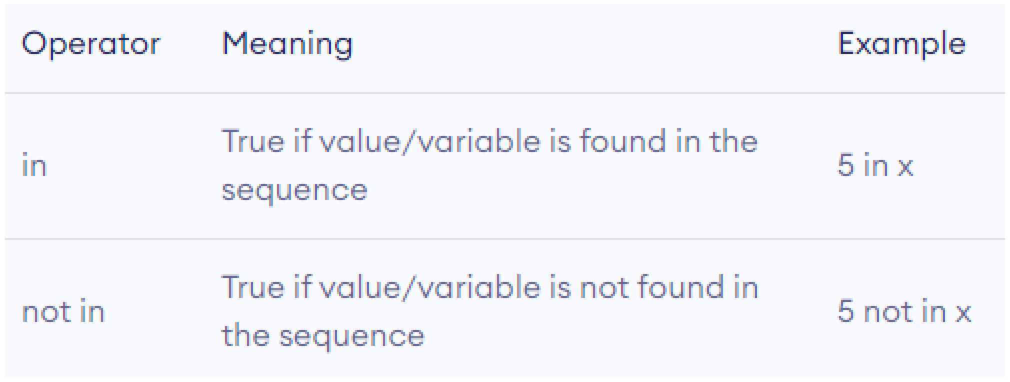
d. Bitwise Operators



In the table below: Let `x` = 10 (`0000 1010` in binary) and `y` = 4 (`0000 0100` in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (`0000 0000`)
|	Bitwise OR	x | y = 14 (`0000 1110`)
~	Bitwise NOT	~x = -11 (`1111 0101`)
^	Bitwise XOR	x ^ y = 14 (`0000 1110`)
>>	Bitwise right shift	x >> 2 = 2 (`0000 0010`)
<<	Bitwise left shift	x << 2 = 40 (`0010 1000`)

e. Membership Operators - to test whether a value or variable is found in a sequence (string, list, set and dictionary)



Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Example:

```
x = 'Hello world'
y = {1:'a',2:'b'}

# Output: True
print('H' in x)

# Output: True
print('hello' not in x)

# Output: True
print(1 in y)

# Output: False
print('a' in y)
```

f. Identity Operators - to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

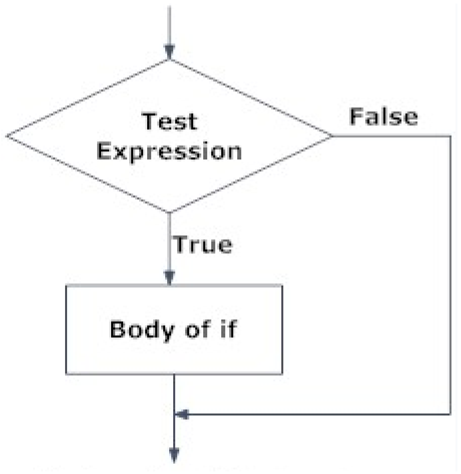
Example:

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

# Output: False
print(x1 is not y1)

# Output: True
print(x2 is y2)

# Output: False
print(x3 is y3)
```

1.4.3 Explain program control in Python

a. Decision making - Python interprets non-zero values as `True` . `None` and `0` are interpreted as `False` .

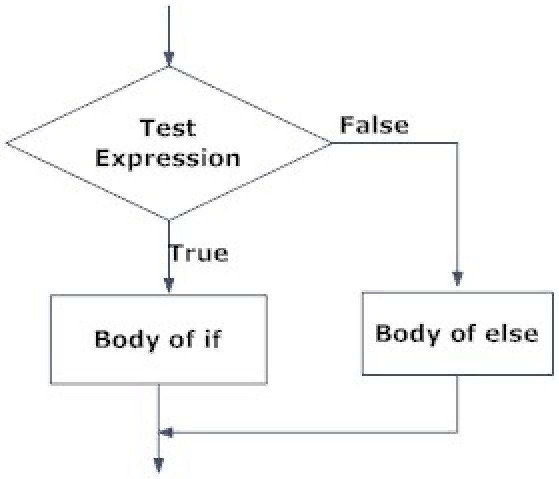
   1. if Statement Syntax



Example:

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

num = -1
if num > 0:
    print(num, "is a positive number.")
print("This is also always printed.")
```

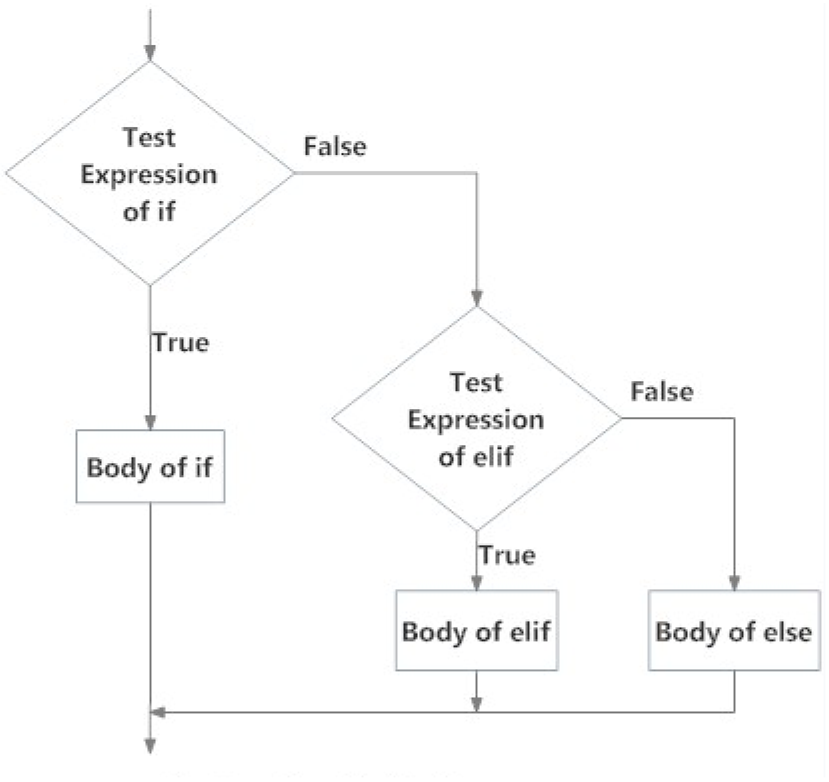
2. if...else Statement Syntax



Example:

```
num = 3

# Try these two variations as well.
# num = -5
# num = 0

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

3. if...elif..else Statement Syntax



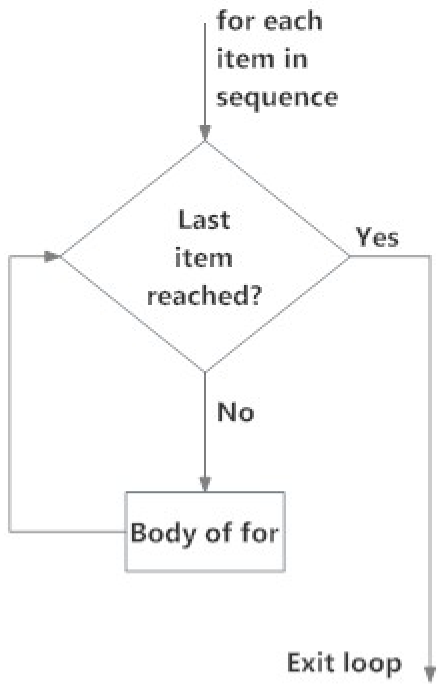
Example:

```
num = 3.4

# Try these two variations as well:
# num = 0
# num = -4.5

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

4. Nested if Statements
   Example:

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

b. Looping
   1. for Loop



Example:

```
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
        sum = sum+val

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

2. range() function - also can be define the start, stop and step size as `range(start, stop,step_size)` . step_size defaults to 1 if not provided.

   Example 1:

```
print(range(10))

print(list(range(10)))

print(list(range(2, 8)))

print(list(range(2, 20, 3)))
```

   Output example 1:

```
range(0, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
[2, 5, 8, 11, 14, 17]
```

   Example 2:

```
genre = ['pop', 'rock', 'jazz']

# iterate over the list using index
for i in range(len(genre)):
        print("I like", genre[i])
```

Output example 2:

```
I like pop
I like rock
I like jazz
```

3. for Loop with else - the keyword `break` can be used to stop a for loop.
   Example 1:

```
digits = [0, 1, 5]

for i in digits:
    print(i)
else:
    print("No items left.")
```

Output example 1:

```
0
1
5
No items left.
```

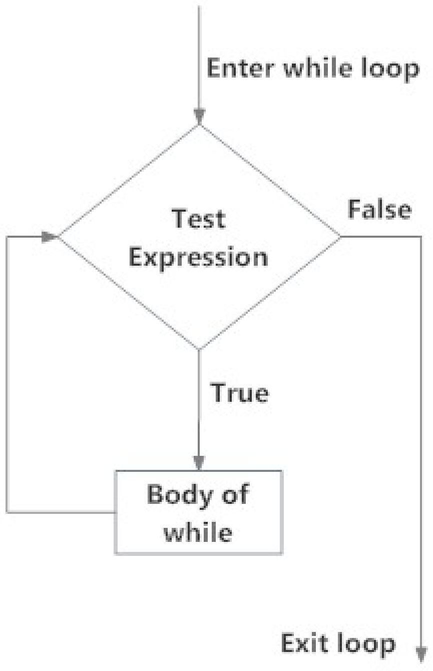
Example 2:

```
student_name = 'Soyuj'

marks = {'James': 90, 'Jules': 55, 'Arthur': 77}

for student in marks:
    if student == student_name:
        print(marks[student])
        break
else:
    print('No entry with that name found.')
```

Output example 2:

```
No entry with that name found.
```

4. while Loop - used to iterate over a block of code as long as the test expression (condition) is true



Example:

```
# n = int(input("Enter n: "))

n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1    # update counter

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

Output:

```
Enter n: 10
The sum is 55
```

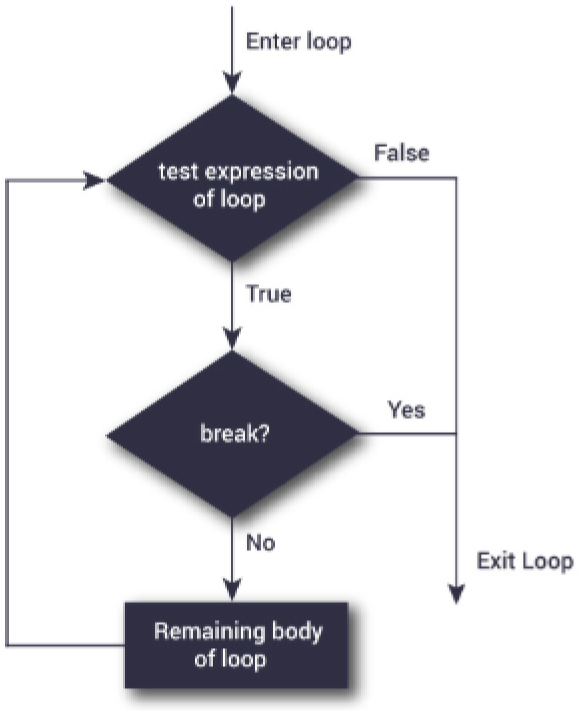
5. while Loop with else - can be terminated with break statement
   Example:

```
counter = 0

while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

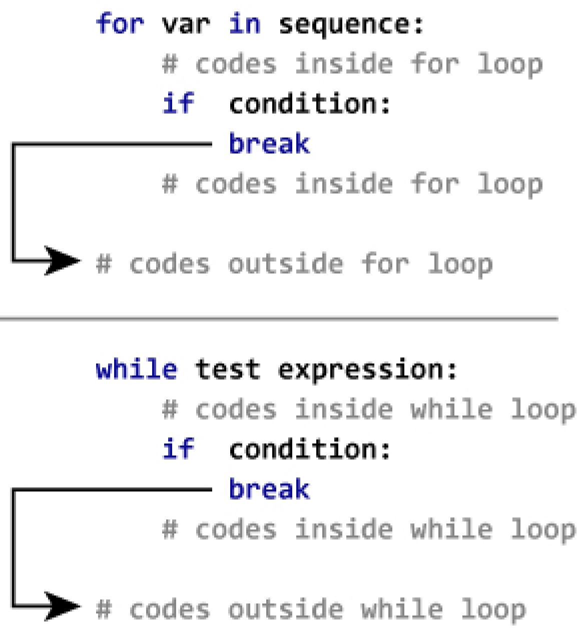
   Output:

```
Inside loop
Inside loop
Inside loop
Inside else
```

6. loop with break statement - terminates the loop containing it



for Loop and while Loop situation:



```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

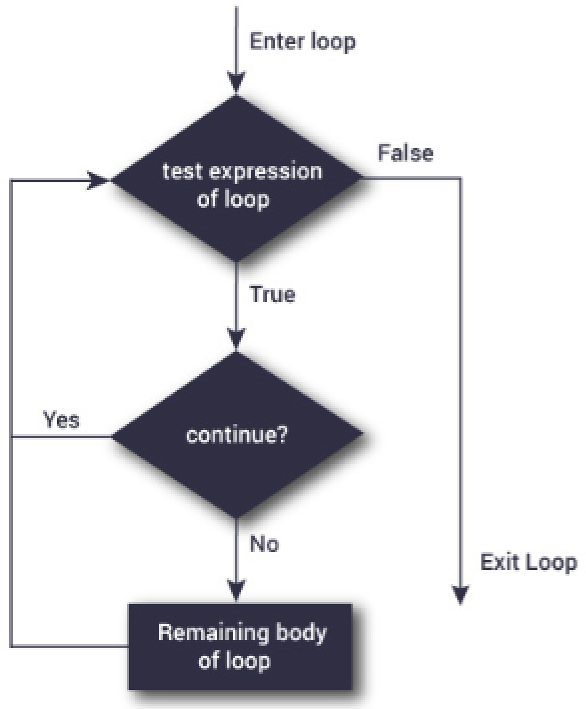
Example:

```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

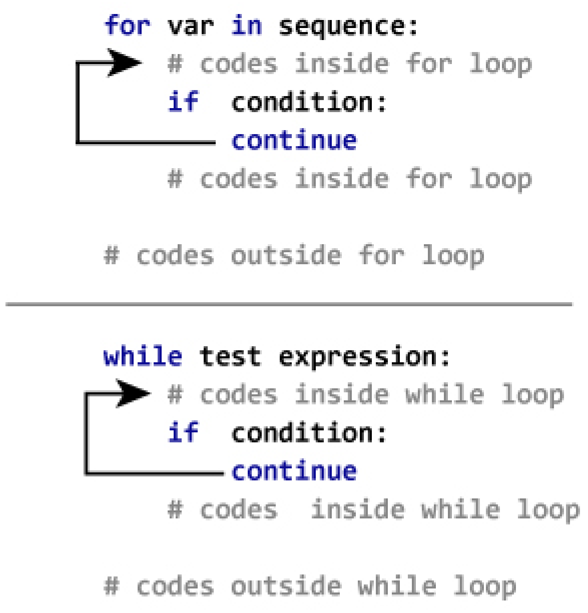
Output:

```
s
t
r
The end
```

7. loop with continue statement - to skip the rest of the code inside a loop for the current iteration only



for loop and while loop situation:



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

Example:

```
for val in "string":
    if val == "i":
        continue
    print(val)

print("The end")
```

Output:

```
s
t
r
n
g
The end
```

1.4.4 Describe modules in Python

To break down large programs into small manageable and organized files.

Modules provide reusability.

Python Built-in Modules - type `help ('modules')` in command prompt OR in IDLE.

1. OS Module
2. Sys Module
3. Math Module
4. Statistics Module
5. Collection Module
6. Random Module

Python **File** - example.py

```
def add(a, b):
    """This program adds two
    numbers and return the result"""

    result = a + b
    return result
```

<--example.py

Python **Module** - example (without extension)

```
>>> import example
>>> example.add(4,5.5)
9.5
```

<--example

a. import statement from Statement
   Example:

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

   Output:

```
The value of pi is 3.141592653589793
```

   Example (import by renaming):

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

b. from .. import * Statement
   Example 1:

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

   Example 2:

```
>>> from math import pi, e
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

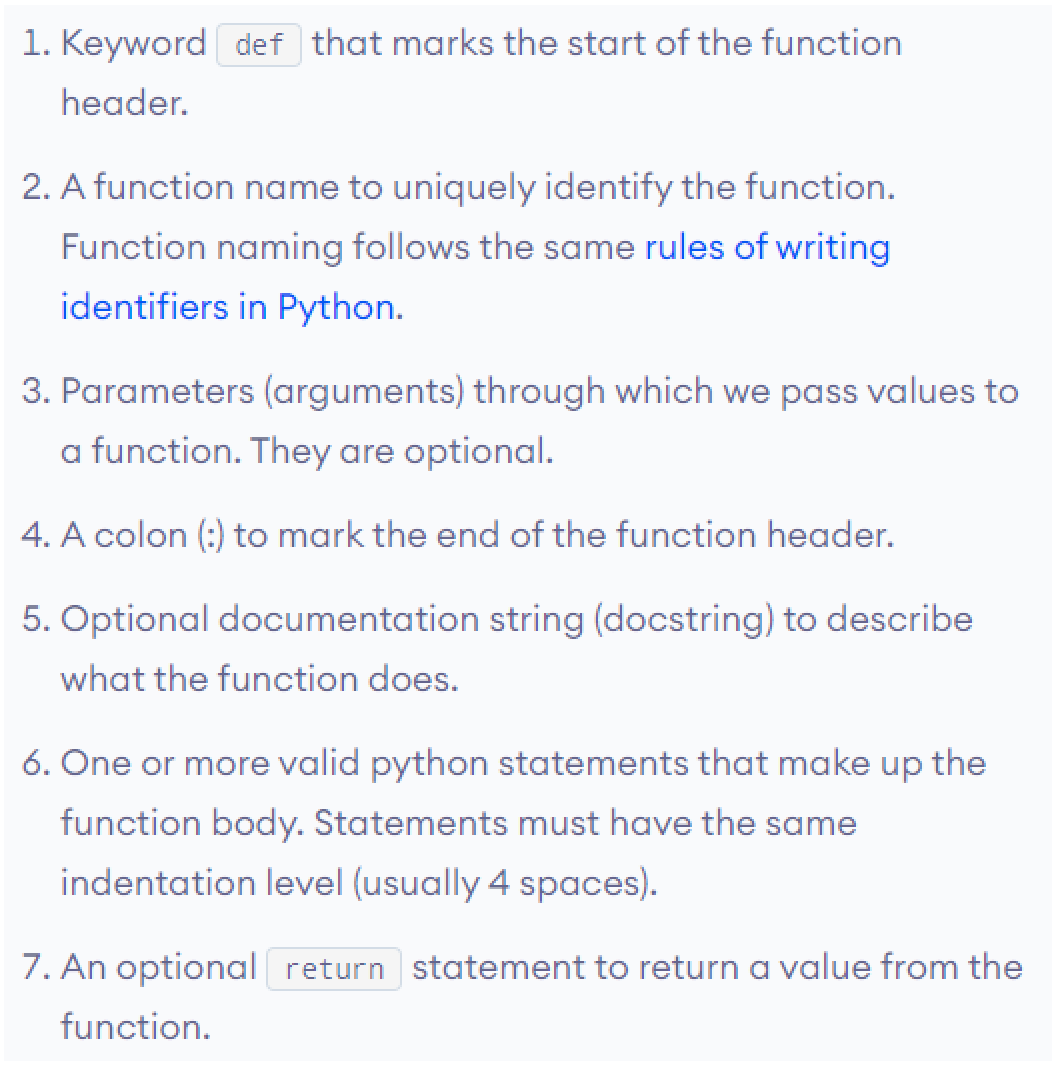
   Example 3:

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

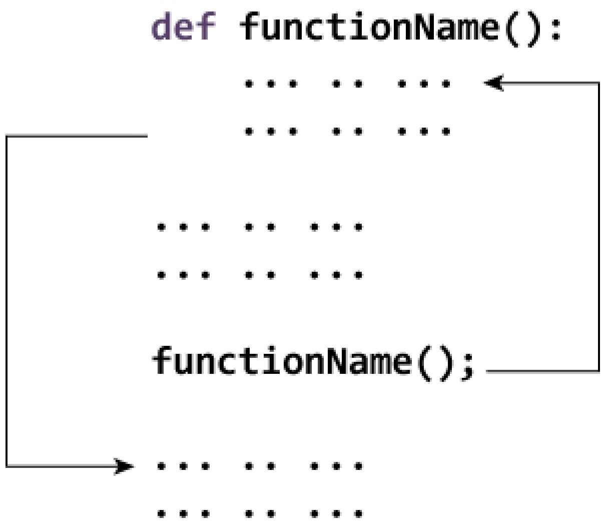
1.4.5 Functions

Types of functions:

1. Built-in functions
   Function that are built into Python
2. User-defined functions
   Functions defined by the users themselves



1. Keyword `def` that marks the start of the function header.
2. A function name to uniquely identify the function. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of the function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have the same indentation level (usually 4 spaces).
7. An optional `return` statement to return a value from the function.



Scope and Lifetime of variables:

```
def my_func():
        x = 10
        print("Value inside function:",x)

x = 20
my_func()
print("Value outside function:",x)
```

Output:

```
Value inside function: 10
Value outside function: 20
```

a. Advantages of user-defined functions
   1. Help to decompose a large program into small segments
   2. Function can be used to include those codes and execute when needed by calling that function if code is repeated in a program
   3. Can divide the workload by making different functions
      Example:

```
def add_numbers(x,y):
   sum = x + y
   return sum

num1 = 5
num2 = 6

print("The sum is", add_numbers(num1, num2))
```

Example to call function:

```
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
```

Output for above example:

```
>>> greet('Paul')
Hello, Paul. Good morning!
```

b. Function arguments
   Example 1:

```
def greet(name, msg):
    """This function greets to
    the person with the provided message"""
    print("Hello", name + ', ' + msg)

greet("Monica", "Good morning!")
```

Output 1:

```
Hello Monica, Good morning!
```

Example 2:

```
def greet(name, msg="Good morning!"):
    """
    This function greets to
    the person with the
    provided message.

    If the message is not provided,
    it defaults to "Good
    morning!"
    """

    print("Hello", name + ', ' + msg)


greet("Kate")
greet("Bruce", "How do you do?")
```

Output 2:

```
Hello Kate, Good morning!
Hello Bruce, How do you do?
```

Example 3: (Any error?)

```
# 2 keyword arguments
greet(name = "Bruce",msg = "How do you do?")

# 2 keyword arguments (out of order)
greet(msg = "How do you do?",name = "Bruce")

1 positional, 1 keyword argument
greet("Bruce", msg = "How do you do?")
```

Example 4: Arbitrary

```
def greet(*names):
    """This function greets all
    the person in the names tuple."""

    # names is a tuple with arguments
    for name in names:
        print("Hello", name)


greet("Monica", "Luke", "Steve", "John")
```

Output 4: Arbitrary

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

Python GPIO Exercises

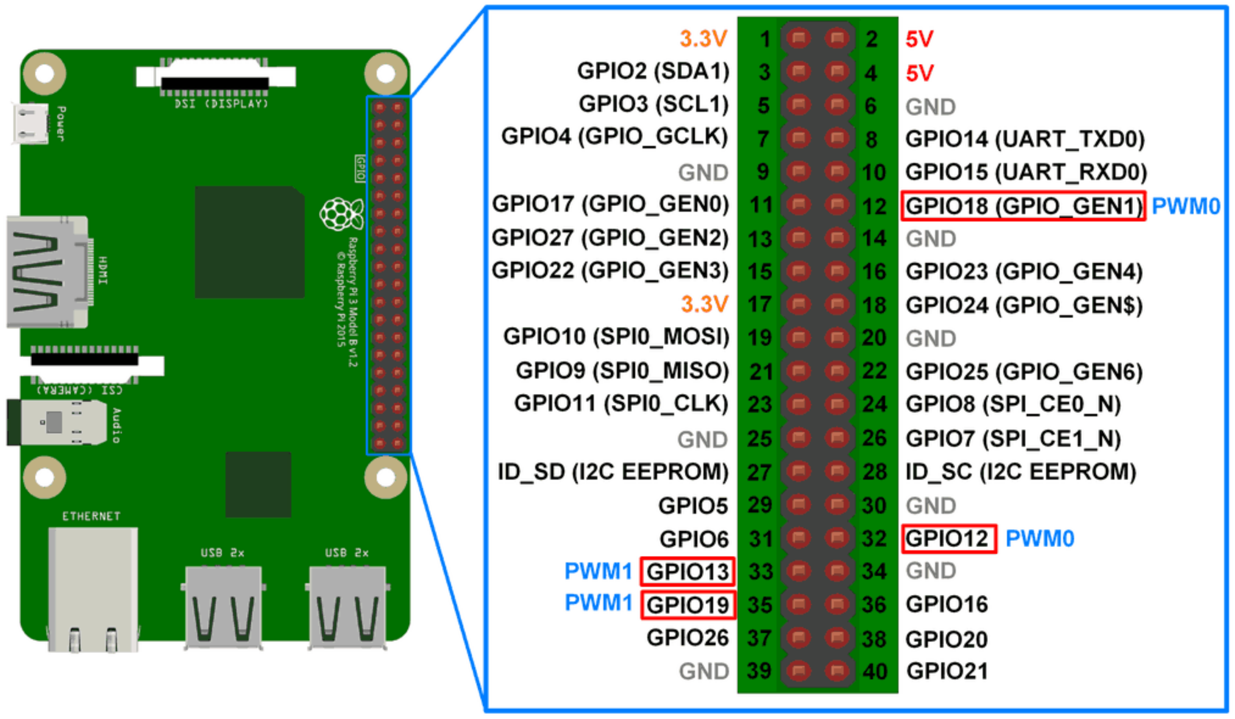
1. https://thepihut.com/blogs/raspberry-pi-tutorials/gpio-and-python-29-leds
2. https://thepihut.com/blogs/raspberry-pi-tutorials/gpio-and-python-39-blinking-led
3. https://www.hackster.io/hardikrathod/push-button-with-raspberry-pi-6b6928

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(10, GPIO.IN, pull_up_down=GPIO.PUD_UP)#Button to GPIO23
GPIO.setup(17, GPIO.OUT)  #LED to GPIO24

#try:
while True:
    button_state = GPIO.input(10)
    if button_state == False:
        GPIO.output(17, True)
        print('Button Pressed...')
        time.sleep(0.2)
    else:
        GPIO.output(17, False)
#except:
#    GPIO.cleanup()
```

4. https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/

```
 #Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped by User")
        GPIO.cleanup()
```

5. https://www.electronicwings.com/raspberry-pi/raspberry-pi-pwm-generation-using-python-and-c
   PWM Generation channel (LED)



DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
GPIO
Raspberry Pi 3 Model B v1.2
© Raspberry Pi 2015
3.3V 1 2 5V
GPIO2 (SDA1) 3 4 5V
GPIO3 (SCL1) 5 6 GND
GPIO4 (GPIO_GCLK) 7 8 GPIO14 (UART_TXD0)
GND 9 10 GPIO15 (UART_RXD0)
GPIO17 (GPIO_GEN0) 11 12 GPIO18 (GPIO_GEN1) PWM0
GPIO27 (GPIO_GEN2) 13 14 GND
GPIO22 (GPIO_GEN3) 15 16 GPIO23 (GPIO_GEN4)
3.3V 17 18 GPIO24 (GPIO_GEN$)
GPIO10 (SPI0_MOSI) 19 20 GND
GPIO9 (SPI0_MISO) 21 22 GPIO25 (GPIO_GEN6)
GPIO11 (SPI0_CLK) 23 24 GPIO8 (SPI_CE0_N)
GND 25 26 GPIO7 (SPI_CE1_N)
ID_SD (I2C EEPROM) 27 28 ID_SC (I2C EEPROM)
GPIO5 29 30 GND
GPIO6 31 32 GPIO12 PWM0
PWM1 GPIO13 33 34 GND
PWM1 GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
GND 39 40 GPIO21

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

ledpin = 12                             # PWM pin connected to LED
GPIO.setwarnings(False)                 #disable warnings
GPIO.setmode(GPIO.BOARD)                #set pin numbering system
GPIO.setup(ledpin,GPIO.OUT)
pi_pwm = GPIO.PWM(ledpin,1000)          #create PWM instance with frequency
pi_pwm.start(0)                         #start PWM of required Duty Cycle
while True:
    for duty in range(0,101,1):
        pi_pwm.ChangeDutyCycle(duty)    #provide duty cycle in the range 0-100
        sleep(0.01)
    sleep(0.5)

    for duty in range(100,-1,-1):
        pi_pwm.ChangeDutyCycle(duty)
        sleep(0.01)
    sleep(0.5)
```

6. PWM Generation channel (DC Motor)

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

PWM1 = 12                                  # PWM pin connected to Enable
MTR1A = 38
MTR1B = 36
GPIO.setwarnings(False)                    #disable warnings
GPIO.setmode(GPIO.BOARD)                   #set pin numbering system
GPIO.setup(PWM1,GPIO.OUT)
GPIO.setup(MTR1A,GPIO.OUT)
GPIO.setup(MTR1B,GPIO.OUT)
pi_pwm = GPIO.PWM(PWM1,1000)               #create PWM instance with frequency
pi_pwm.start(0)                            #start PWM of required Duty Cycle
while True:
   GPIO.output(MTR1A,True)                 #Forward direction setup
   GPIO.output(MTR1B,False)
   for duty in range(0,101,1):
      pi_pwm.ChangeDutyCycle(duty)         #provide duty cycle in the range 0-100
      sleep(0.01)
   sleep(0.5)

   for duty in range(100,-1,-1):
      pi_pwm.ChangeDutyCycle(duty)
      sleep(0.01)
   sleep(0.5)
```