

## Unit-3

<b>Introduction to Python</b>	<b>2</b>
<b>Data types in Python</b>	<b>5</b>
<b>Loop and Control statements</b>	<b>10</b>
<b>Python Functions</b>	<b>14</b>
<b>Exception Handling in Python</b>	<b>17</b>
<b>File Read Write Operations</b>	<b>18</b>
<b>Image Read Write Operations</b>	<b>22</b>
<b>Networking in Python</b>	<b>25</b>
<b>Introduction to Raspberry Pi</b>	<b>27</b>
<b>Basic Architecture of Raspberry Pi</b>	<b>30</b>
<b>Programming with Raspberry Pi</b>	
<b>Blinking LED</b>	<b>34</b>
<b>Interfacing of LED and Switch with Raspberry Pi</b>	<b>36</b>
<b>Implementation IOT with Raspberry Pi</b>	<b>37</b>
<b>Program for interfacing with sensors and actuators</b>	<b>40</b>

# INTRODUCTION TO PYTHON

## What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

**Python** is a dynamic, high level, free open source and interpreted programming language. It supports object-oriented programming as well as procedural oriented programming.

In Python, we don't need to declare the type of variable because it is a dynamically typed language.

For example, `x = 10`

Here, `x` can be anything such as String, int, etc.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

## What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

## Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

## Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

## Python Features

There are many features in Python, some of which are discussed below –

### 1. Easy to code:

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

### 2. Free and Open Source:

Python language is freely available at the official website and you can download it from the given download link below click on the **Download Python** keyword. Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

### 3. Object-Oriented Language:

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

### 4. GUI Programming Support:

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.

PyQt5 is the most popular option for creating graphical apps with Python.

### 5. High-Level Language:

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

### 6. Extensible feature:

Python is a **Extensible** language. We can write us some Python code into C or C++ language and also we can compile that code in C/C++ language.

### 7. Python is Portable language:

Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

### 8. Python is Integrated language:

Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.

### 9. Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called **bytecode**.

### 10. Large Standard Library

Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

## 11. Dynamically Typed Language:

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

## What is Python IDE

Python code editors are designed for the developers to code and debug program easily. Using these Python IDEs(Integrated Development Environment), you can manage a large codebase and achieve quick deployment.

Python IDE is a free and open source software that is used to write codes, integrate several modules and libraries.

It is available for installation into PC with Windows, Linux and Mac.

Examples: Spyder, PyCharm,jupyter etc.

## Data Types in Python

Python has five standard Data Types:

- [Numbers](#)
- [String](#)
- [List](#)
- [Tuple](#)
- [Dictionary](#)

Python sets the variable type based on the value that is assigned to it. Unlike more riggers languages, Python will change the variable type if the variable value is set to another value. For example:

```
var = 123 # This will create a number integer assignment  
var = 'john' # the `var` variable is now a string type.
```

# Numbers

Python numbers variables are created by the standard Python method:

```
var = 382
```

Most of the time using the standard Python number type is fine. Python will automatically convert a number from one type to another if it needs. But, under certain circumstances that a specific number type is needed (ie. complex, hexadecimal), the format can be forced into a format by using additional syntax in the table below:

Type		Format		Description
int		a = 10		Signed Integer
long		a = 345L		(L) Long integers, they can also be represented in octal and hexadecimal
float		a = 45.67		(.) Floating point real values
complex		a = 3.14J		(J) Contains integer in the range 0 to 255.

Most of the time Python will do variable conversion automatically. You can also use Python conversion functions ([int\(\)](#), [long\(\)](#), [float\(\)](#), [complex\(\)](#)) to convert data from one type to another. In addition, the `type` function returns information about how your data is stored within a variable.

```
message = "Good morning"
num = 85
pi = 3.14159

print(type(message)) # This will return a string
print(type(n)) # This will return an integer
print(type(pi)) # This will return a float
```

# String

Create string variables by enclosing characters in quotes. Python uses single quotes `'`, double quotes `"` and triple quotes `"""` to denote literal strings. Only the triple quoted strings `"""` also will automatically continue across the end of line statement.

```
firstName = 'john'
lastName = "smith"
message = """This is a string that will span across multiple lines. Using newline
characters
and no spaces for the next lines. The end of lines within this string also count as a
newline when printed"""
```

Strings can be accessed as a whole string, or a substring of the complete variable using brackets `[]`. Here are a couple examples:

```
var1 = 'Hello World!'
var2 = 'RhinoPython'

print var1[0] # this will print the first character in the string an `H`
print var2[1:5] # this will print the substring 'hinoP`
```

Python can use a special syntax to format multiple strings and numbers. The string formatter is quickly covered here because it is seen often and it is important to recognize the syntax.

```
print "The item {} is repeated {} times".format(element,count))
```

The `{}` are placeholders that are substituted by the variables `element` and `count` in the final string. This compact syntax is meant to keep the code more readable and compact.

Python is currently transitioning to the format syntax above, but python can use an older syntax, which is being phased out, but is still seen in some example code:

```
print "The item %i is repeated %i times"% (element,count)
```

# List

Lists are a very useful variable type in Python. A list can contain a series of values. List variables are declared by using brackets `[ ]` following the variable name.

```
A = [ ] # This is a blank list variable
B = [1, 23, 45, 67] # this list creates an initial list of 4 numbers.
C = [2, 4, 'john'] # Lists can contain different variable types.
```

All lists in Python are zero-based indexed. When referencing a member or the length of a list the number of list elements is always the number shown plus one.

```
mylist = ['Rhino', 'Grasshopper', 'Flamingo', 'Bongo']
B = len(mylist) # This will return the length of the list which is 3. The index is 0, 1, 2, 3.
print mylist[1] # This will return the value at index 1, which is 'Grasshopper'
print mylist[0:2] # This will return the first 3 elements in the list.
```

You can assign data to a specific element of the list using an index into the list. The list index starts at zero. Data can be assigned to the elements of an array as follows:

```
mylist = [0, 1, 2, 3]
mylist[0] = 'Rhino'
mylist[1] = 'Grasshopper'
mylist[2] = 'Flamingo'
mylist[3] = 'Bongo'
print mylist[1]
```

Lists aren't limited to a single dimension. Although most people can't comprehend more than three or four dimensions. You can declare multiple dimensions by separating an with commas. In the following example, the MyTable variable is a two-dimensional array :

```
MyTable = [[], []]
```

In a two-dimensional array, the first number is always the number of rows; the second number is the number of columns.



# Tuple

Tuples are a group of values like a list and are manipulated in similar ways. But, tuples are fixed in size once they are assigned. In Python the fixed size is considered immutable as compared to a list that is dynamic and mutable. Tuples are defined by parenthesis ().

```
myGroup = ('Rhino', 'Grasshopper', 'Flamingo', 'Bongo')
```

Here are some advantages of tuples over lists:

1. Elements to a tuple. Tuples have no append or extend method.
2. Elements cannot be removed from a tuple.
3. You can find elements in a tuple, since this doesn't change the tuple.
4. You can also use the in operator to check if an element exists in the tuple.
5. Tuples are faster than lists. If you're defining a constant set of values and all you're ever going to do with it is iterate through it, use a tuple instead of a list.
6. It makes your code safer if you "write-protect" data that does not need to be changed.

It seems tuples are very restrictive, so why are they useful? There are many datastructures in Rhino that require a fixed set of values. For instance a Rhino point is a list of 3 numbers [34.5, 45.7, 0]. If this is set as tuple, then you can be assured the original 3 number structure stays as a point (34.5, 45.7, 0). There are other datastructures such as lines, vectors, domains and other data in Rhino that also require a certain set of values that do not change. Tuples are great for this.

# Dictionary

Dictionaries in Python are lists of Key:Value pairs. This is a very powerful datatype to hold a lot of related information that can be associated through keys. The main operation of a dictionary is to extract a value based on the key name. Unlike lists, where index numbers are used, dictionaries allow the use of a key to access its members. Dictionaries can also be used to sort, iterate and compare data.

Dictionaries are created by using braces ({}), with pairs separated by a comma (,) and the key values associated with a colon (:). In Dictionaries the Key must be unique. Here is a quick example on how dictionaries might be used:

```
room_num = {'john': 425, 'tom': 212}
room_num['john'] = 645 # set the value associated with the 'john' key to 645
```

```
print (room_num['tom']) # print the value of the 'tom' key.
room_num['isaac'] = 345 # Add a new key 'isaac' with the associated value
print (room_num.keys()) # print out a list of keys in the dictionary
print ('isaac' in room_num) # test to see if 'issac' is in the dictionary. This
returns true.
```

Dictionaries can be more complex to understand, but they are great to store data that is easy to access.

## Loops and Control Statements

Python programming language provides following types of loops to handle looping requirements.

### While Loop

Syntax :

```
while expression:
    statement(s)
```

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

```
# prints Hello Geek 3 Times

count = 0

while (count < 3):

    count = count+1

    print("Hello Geek")
```

Output:

Hello Geek

Hello Geek

Hello Geek

## If statements

Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

### Example

If statement:

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

[Try it Yourself »](#)

In this example we use two variables, `a` and `b`, which are used as part of the if statement to test whether `b` is greater than `a`. As `a` is 33, and `b` is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

# Indentation

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

## Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
    print("b is greater than a") # you will get an error
```

[Try it Yourself »](#)

---

---

## Elif

The `elif` keyword is python's way of saying "if the previous conditions were not true, then try this condition".

## Example

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

[Try it Yourself »](#)

In this example `a` is equal to `b`, so the first condition is not true, but the `elif` condition is true, so we print to screen that "a and b are equal".

---

# Else

The `else` keyword catches anything which isn't caught by the preceding conditions.

## Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

### Try it Yourself »

In this example `a` is greater than `b`, so the first condition is not true, also the `elif` condition is not true, so we go to the `else` condition and print to screen that "a is greater than b".

You can also have an `else` without the `elif`:

## Example

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

# Python For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the `for` keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the **for** loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

# Python Functions

---

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

## Creating a Function

In Python a function is defined using the **def** keyword:

### Example

```
def my_function():
    print("Hello from a function")
```

## Calling a Function

To call a function, use the function name followed by parenthesis:

### Example

```
def my_function():
    print("Hello from a function")
```

```
my_function()
```

- Example showing function returning multiple values **def greater(x, y):**
  - **if x > y:**  
    **return x, y**

```

else:
    return y, x
val = greater(10, 100) print(val)

```

- **Output:: (100,10)**

### **Functions as Objects**

Functions can also be assigned and reassigned to the variables

#### **Example:**

```

def add (a,b):
    return a+b
print (add(4,6))
c = add(4,6)
print c
Output:: 10 10

```

## **Variable Scope in Python :**

**Global variables:** These are the variables declared out of any function , but can be accessed inside as well as outside the function.

**Local variables:** These are the ones that are declared inside a function

### **Example of Global Variables :**

```

g_var = 10
def example():
    l_var = 100
    print(g_var)
example()           # calling the function

```

Output:: 10

## Example showing Variable scope

```
var = 10                # declaring globally
def example():
    var = 100           #declaring locally
    print(var)
example()               # calling the function
print(var)
```

**Output**

**100**  
**10**

## Modules concept in Python

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.



## The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

```
import module1[, module2[, ... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module `support.py`, you need to put the following command at the top of the script –

Syntax:

```
import module_name #At the top of the code
```

```
using module_name.var    #To access functions          and values with 'var' in
the module
```

Example:

```
import random
for i in range(1,10):
    val = random.randint(1,10)
    print (val)
```

Output:: varies with each execution

## We can also access only a particular function from a module. ▀

Example:

```
from math import pi
print (pi)
```

Output:: 3.14159

## Exception Handling in Python

An error that is generated during execution of a program, is termed as exception. ■

Syntax:

```
try:
    statements
except _Exception_:
    statements
else:
    statements
```

Example:

```
while True:
    try:
        n = input ("Please enter an integer: ")
        n = int (n)
        break
    except ValueError:
        print "No valid integer! "
print "It is an integer!"
```

## File Read Write Operations

Python allows you to read and write files  
No separate module or library required

## Three basic steps

Open a file

Read/Write

Close the file

### Opening Files in Python

Python has a built-in `open()` function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

`Open()` function is used to open a file, returns a file object

`open(file_name, mode)`

We can specify the mode while opening a file. In mode, we specify whether we want to read `r`, write `w` or append `a` to the file. We can also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like images or executable files.

Mode	Description
<code>r</code>	Opens a file for reading. (default)
<code>w</code>	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.

<code>x</code>	Opens a file for exclusive creation. If the file already exists, the operation fails.
<code>a</code>	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
<code>t</code>	Opens in text mode. (default)
<code>b</code>	Opens in binary mode.
<code>+</code>	Opens a file for updating (reading and writing)

## Read from a file:

`read()`: Reads from a file

`file=open('data.txt', 'r') file.read()`

## Write to a file:

`Write()`: Writes to a file

`file=open('data.txt', 'w') file.write('writing to the file')`

Closing a file: ■

`Close()`: This is done to ensure that the file is free to use for other resources

`file.close()`

## Using WITH to open a file:

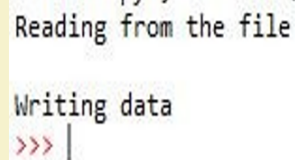
- Good practice to handle exception while file read/write operation

- Ensures the file is closed after the operation is completed, even if an exception is encountered
- with open("data.txt","w") as file:  
 file.write("writing to the text file")  
 file.close()

## File Read Write Operations code + image

```
with open("PythonProgram.txt","w") as file:
    file.write("Writing data")
file.close()
```

```
with open("PythonProgram.txt","r") as file:
    f=file.read()
    print('Reading from the file\n')
    print (f)
file.close()
```



```
Reading from the file
Writing data
>>> |
```

## File Read Write Operations (contd..)

### Comma Separated Values Files

- CSV module supported for CSV files

#### Read:

```
with open(file, "r") as csv_file:
    reader = csv.reader(csv_file)
    print("Reading from the CSV File\n")
    for row in reader:
        print(" ".join(row))
csv_file.close()
```

#### Write:

```
data = ["1,2,3,4,5,6,7,8,9".split(",")]
file = "output.csv"
with open(file, "w") as csv_file:
    writer = csv.writer(csv_file, delimiter=',')
    print("Writing CSV")
    for line in data:
        writer.writerow(line)
csv_file.close()
```

## File Read Write Operations (contd..)

```
import csv

#writing a csv file
data = ["1,2,3,4,5,6,7,8,9".split(",")]
file = "output.csv"
with open(file, "w") as csv_file:
    writer = csv.writer(csv_file, delimiter=',')
    print("Writing CSV")
    for line in data:
        writer.writerow(line)
csv_file.close()

#reading from a csv file
with open(file, "r") as csv_file:
    reader = csv.reader(csv_file)
    print("Reading from the CSV File\n")
    for row in reader:
        print(" ".join(row))
csv_file.close()
```

```
Writing CSV
Reading from the CSV File

1 2 3 4 5 6 7 8 9

>>>
```

## Image Read ,Write Operations

- Python supports PIL library for image related operations
- Install PIL through PIP

`sudo pip install pillow`

PIL is supported till python version 2.7. Pillow supports the 3x version of python.

Reading Image in Python:

- PIL: Python Image Library is used to work with image files
- from PIL import Image**

- Open an image file

**image=Image.open(image\_name)**

- Display the image

**image.show()**

**Resize():** Resizes the image to the specified size

**image.resize(255,255)**

**Rotate():** Rotates the image to the specified degrees, counter clockwise

**image.rotate(90)**

**Format:** Gives the format of the image

**Size:** Gives a tuple with 2 values as width and height of the image, in pixels

**Mode:** Gives the band of the image, 'L' for grey scale, 'RGB' for true colour image

**print(image.format, image.size, image.mode)**

Convert image to different mode:

- Any image can be converted from one mode to 'L' or 'RGB' mode

**conv\_image=image.convert('L')**

- Conversion between modes other than 'L' and 'RGB' needs conversion into any of these 2 intermediate modes

# Output

Converting a sample image to Grey Scale

```
from PIL import Image

im = Image.open('/home/saswati/VRP_Linux/Images/i3.jpg')
im.show()
grey_image=im.convert('L')
grey_image.show()
grey_image.save('GreyScaleImage.jpg')
```

## Output





## Networking in Python

- Python provides network services for client server model.
- Socket support in the operating system allows to implement clients and servers for both connection-oriented and connectionless protocols.
- Python has libraries that provide higher-level access to specific application-level network protocols.

### Syntax for creating a socket:

**s = socket.socket (socket\_family, socket\_type, protocol=0)**

**socket\_family** – AF\_UNIX or AF\_INET

**socket\_type** – SOCK\_STREAM or SOCK\_DGRAM

**protocol** – default '0'.

### Example - simple server

- The socket waits until a client connects to the port, and then returns a connection object that represents the connection to that client.

```
import socket
import sys
```

```
# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
# Bind the socket to the port
server_address = ('10.14.88.82', 2017)
print >>sys.stderr, 'starting up on %s port %s' % server_address
sock.bind(server_address)
```

```
# Listen for incoming connections
sock.listen(1)

connection, client_address = sock.accept()

#Receive command
data = connection.recv(1024)
print(data)
sock.close()
```

## Example - simple client

```
import socket
import sys

# Create a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Connect to Listener socket
client_socket.connect(("10.14.88.82", 2017))
print>>sys.stderr,'Connection Established'

#Send command
client_socket.send('Message to the server')
print('Data sent successfully')
```

## Output

```
starting up on 10.14.88.82 port 2017
Message to the server
saswati@saswati-BK361AA-ACJ-CQ3236IX:~/Desktop$
```

```
Connection Established
Data sent successfully
saswati@saswati-BK361AA-ACJ-CQ3236IX:~/Desktop$
```

## INTRODUCTION TO RASPBERRY PI

### What is a Raspberry Pi?

Raspberry pi is a low cost minicomputer with the physical size of a credit card. Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do. In addition to this , Raspberry Pi allows interfacing sensors, actuators , through th general purpose I/O pins(GPIO) .Since Raspberry Pi runs Linus Operating system, it supports Python “out of the box”.

### Differences between Raspberry Pi and Arduino

Both Arduino and Raspberry Pi are good teaching tools for students, beginners and hobbyists. Let us see some of the differences between Raspberry Pi and Arduino.

- The main difference between them is: Arduino is microcontroller board, while Raspberry Pi is a microprocessor based mini computer (SBC).
- The Microcontroller on the Arduino board contains the CPU, RAM and ROM. All the additional hardware on Arduino Board is for power supply, programming and IO Connectivity. Raspberry Pi SBC has all features of a computer with a processor, memory, storage, graphics driver, connectors on the board.
- Raspberry Pi needs an Operating System to run. Arduino doesn't need any operating system. All you need is a binary of the compiled source code.
- Raspberry Pi comes with a fully functional operating system called Raspberry Pi OS (previously known as Raspbian OS). Although Pi can use different operating systems, Linux is preferred by Raspberry Pi Foundation. You can install Android, if you want. **Arduino does not have any operating system. You just need a firmware instructing the Microcontroller what task to do.**
- The clock speed of Arduino is 16 MHz while the clock speed of Raspberry Pi is around 1.2 GHz.
- Raspberry Pi is good for developing software applications using Python, while Arduino is good for interfacing Sensors and controlling LEDs and Motors.
- This doesn't mean we cannot connect sensors and LEDs to Raspberry Pi. To encourage learning programming by controlling hardware, the Raspberry Pi consists of a 40-pin GPIO, through which you can connect

different electronic components like LEDs, Buttons, Sensors, Motors etc. On Arduino, the GPIO is called as Digital IO (for digital Input and Output) and Analog IN (for Analog Input).

- Using Arduino Shields, which plug into the Arduino Pin headers, you can add a dedicated feature or functionality like a Motor Driver, Ethernet Connection, SD Card Reader, Wi-Fi, Touchscreens, cameras etc. to Arduino. While Raspberry Pi is a self-contained board, you can add external hardware like Touchscreen, GPS, RGB panels etc. to Raspberry Pi. The Raspberry Pi Hardware Attached on Top or HAT Expansion Boards are inspired by Arduino Shields, using which you can add additional functionality to Raspberry Pi. They are connected to the GPIO Pins.
- The power requirements of Raspberry Pi and Arduino are completely different. Even though they both are powered by USB (micro-USB or USB Type C for Raspberry Pi and USB Type B for Arduino), Raspberry Pi needs more more current than Arduino. So, you need a power adapter for Raspberry Pi but you can power Arduino from the USB port of a Computer.
- Power interruption for Raspberry Pi may cause damage to the hardware, software or applications. In case of Arduino, if there is any power cut it again restarts. So, Raspberry Pi must be properly shutdown before disconnecting power.
- Arduino uses Arduino IDE for developing the code. While Raspberry Pi can use Python IDLE, Eclipse IDE or any other IDE that is supported by Linux. You can also program using the terminal itself with any text editor like Vim.
- Using the open-source hardware and software files of Arduino, you can essentially create your own Arduino board. This is not possible with Raspberry Pi as it is not open-source.
- The cost of original Arduino UNO is \$23 but there are several clones of Arduino which are available for less than \$4. Coming to Raspberry Pi, the original Raspberry Pi SBC was around \$35, but the latest Raspberry Pi 4 Model B is available in different price points (\$35, \$55 or \$75) depending on the memory configuration.

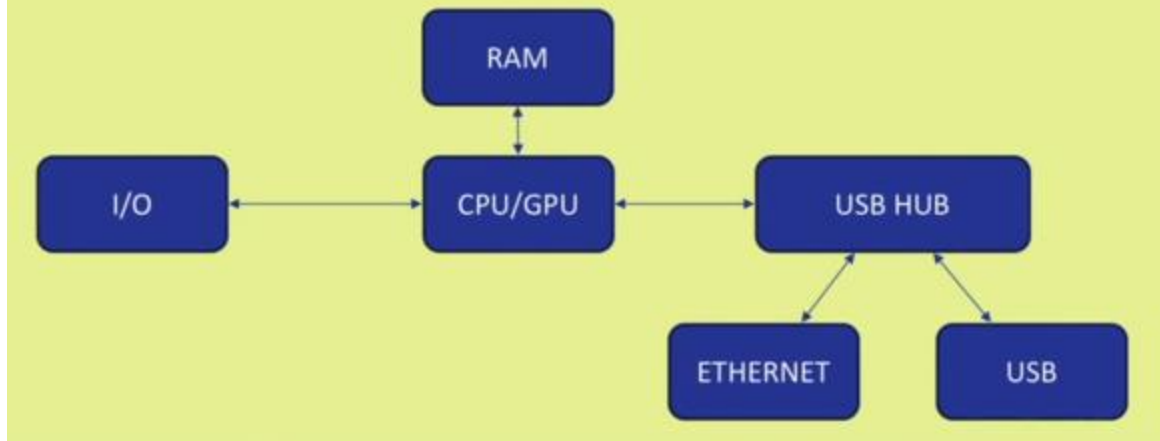
# Specific ations

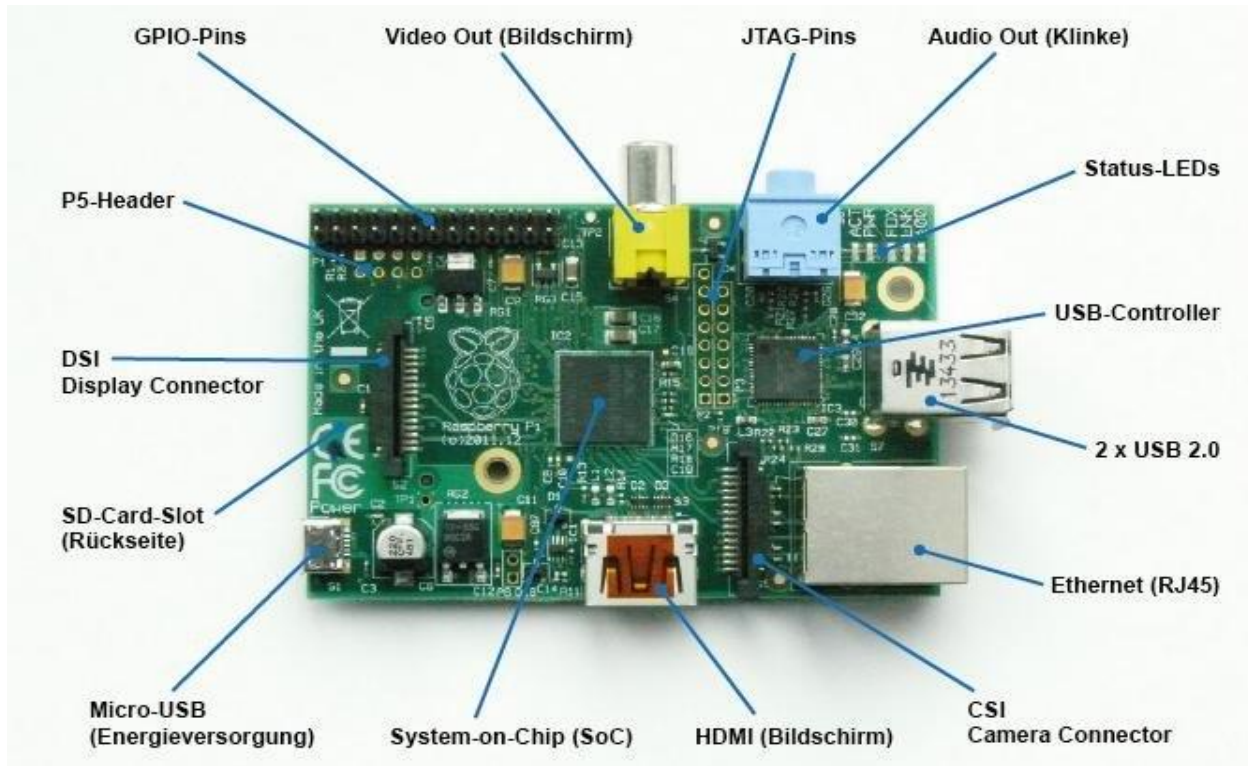
## Key features

	Raspberry pi 3 model B	Raspberry pi 2 model B	Raspberry Pi zero
<i>RAM</i>	<i>1GB SDRAM</i>	<i>1GB SDRAM</i>	<i>512 MB SDRAM</i>
<i>CPU</i>	<i>Quad cortex A53@1.2GHz</i>	<i>Quad cortex A53@900MHz</i>	<i>ARM 11@ 1GHz</i>
<i>GPU</i>	<i>400 MHz video core IV</i>	<i>250 MHz video core IV</i>	<i>250 MHz video core IV</i>
<i>Ethernet</i>	<i>10/100</i>	<i>10/100</i>	<i>None</i>
<i>Wireless</i>	<i>802.11/Bluetooth 4.0</i>	<i>None</i>	<i>None</i>
<i>Video output</i>	<i>HDMI/Composite</i>	<i>HDMI/Composite</i>	<i>HDMI/Composite</i>
<i>GPIO</i>	<i>40</i>	<i>40</i>	<i>40</i>

## Basic Architecture of Raspberry Pi

## Basic Architecture





# Raspberry Pi hardware specifications

## Explanation about board components

### 1. System on Chip

The **System on Chip (SoC)** architecture that the Raspberry Pi 2 implements is the Broadcom BCM2836, which we touched upon earlier in this chapter. This contains a CPU, GPU, SDRAM, and single USB port. Each of these items is discussed in more detail under the appropriate heading.

## 2. CPU

A central processing unit is the brain of your Raspberry Pi. It is responsible for processing machine instructions, which are the result of your compiled programs.

## 3. GPU

The graphics processing unit (GPU) is a specialist chip designed to handle the complex mathematics required to render graphics.

## 4. 4 USB 2.0 ports and 1 SoC on-board USB

The previous version of the Raspberry Pi Model B contained only a single microUSB port and a two standard USB ports. The Raspberry Pi 2 has been expanded to include an onboard 5-port USB hub.

This allows you to connect four standard USB cables to the device and a single microUSB cable. The micro USB port can be used to power your Raspberry Pi 2.

## 5. SD card Slot

Raspberry Pi does not have a builtin operating a system and storage. You can Plug-in SD card loaded with a Linux image to the SD card slot.

## 6. DSI display connector –

used to attach an LCD panel. On the other side of the board is a **microSD** card slot that holds the operating system.

## 7. Camera Serial Interface(CSI)

This interface can be used to connect a camera module to Raspberry Pi.

## 8. Status LED's :

Raspberry Pi has 5 status LED's



Status LED	FUNCTIONS
ACT	SD card access
PWR	3.3 V power is present
FDX	Full Duplex Lan connected
LNK	Link/Network activity
100	100 Mbit Lan connected

## 9. Ethernet port

If you plan to place your Raspberry Pi near a router or switch or have enough Ethernet cable, then you can connect your Raspberry Pi directly with the Ethernet jack.

The Raspberry Pi 2 supports 10/100 Mbps Ethernet, and the USB adapter in the third/fourth port of USB hub can also be used for Ethernet via a USB to Ethernet adapter

## 10. GPIO pins

The main method for interacting with electronic components and expansion boards is through the **general purpose input/output (GPIO)** pins on the Raspberry Pi.

The Raspberry Pi 2 Model B contains 40 pins in total.

GPIO pins can accept both input and output commands and can be controlled by programs in a variety of languages running on the Raspberry Pi.

for example :The input could be readings from a temperature sensor, and the output a command to another device to switch an LED on or off.

## 11. HDMI Output :

This port provides both video and audio output. You can connect the Raspberry Pi to a monitor using HDMI cable .

## **12. Composite Video output :**

Raspberry Pi comes with composite video output with RCA jack that supports both PAL and NTSC video output. RCA jack can be used to connect old televisions that have an RCA input only.

## **13. Audio Output :**

Raspberrypi has 3.5mm audio output jack .This audio jacks use for providing audio output to old Televisions along with RCA jack for video .The audio quality from this jack is inferior to the HDMI output

# **Programming with Raspberry Pi**

## **Program for Blinking LED**

With the circuit created we need to write the Python script to blink the LED. Before we start writing the software we first need to install the Raspberry Pi GPIO Python module. This is a library that allows us to access the GPIO port directly from Python.

### **Installing GPIO library:**

- Open terminal
- Enter the command “sudo apt-get install python-dev” to install python development

- Enter the command “sudo apt-get install python-rpi.gpio” to install GPIO library.

Connection :

- 1.connect the negative terminal of LED to ground pin of Pi
- 2.connect the positive terminal of LED to output pin of Pi

### Basic python coding:

Open terminal enter the command

**sudo nano filename.py**

This will open the nano editor where you can write your code

Ctrl+O : Writes the code to the file

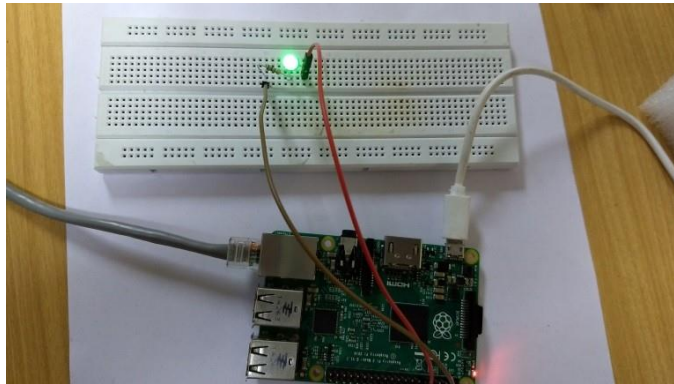
- Ctrl+X : Exits the editor

## Code

Code:

```
import RPi.GPIO as GPIO      #GPIO library
import time
GPIO.setmode(GPIO.BOARD)    # Set the type of board for pin numbering
GPIO.setup(11, GPIO.OUT)    # Set GPIO pin 11as output pin
for i in range (0,5):
    GPIO.output(11,True)    # Turn on GPIO pin 11
    time.sleep(1)
    GPIO.output(11,False)
    time.sleep(2)
    GPIO.output(11,True)
GPIO.cleanup()
```

Output : The LED blinks in a loop with delay of 1 and 2 seconds.



## Program for interfacing of LED and switch with Raspberry Pi

In this program LED is connected to GPIO pin 18 and switch is connected to pin 25. In the infinite While loop the value of Pin25 is checked and the state of LED is toggled if switch is pressed .

### Code :

```
from time import sleep
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
#switch Pin
GPIO.setup(25,GPIO.IN)
#LED Pin
GPIO.setup(18,GPIO.OUT)
state=False
```

```

def LED(pin):
    state=not state
    GPIO.output(pin,state)
while True:
    try :
        If(GPIO.input(25)==True):
            LED(pin)
            Sleep(1)
    except KeyboardInterrupt:
        exit()

```

# Implementation of IoT with Raspberry Pi

## Internet Of Things is

Creating an interactive environment  
 Network of devices connected together

## Sensor

Electronic element  
 Converts physical quantity into electrical signals  
 Can be analog or digital

### DHT11 vs DHT22

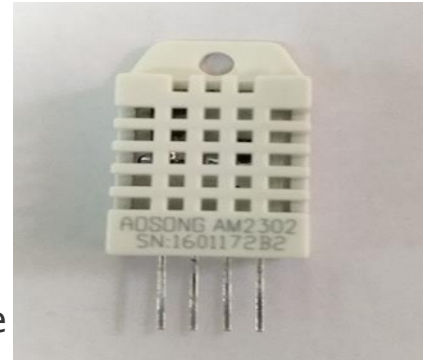
We have two versions of the DHT sensor, they look a bit similar and have the same pinout, but have different characteristics. Here are the specs:

## DHT11

- Ultra low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while requesting data)
- Good for 20–80% humidity readings with 5% accuracy
- Good for 0–50°C temperature readings  $\pm 2^{\circ}\text{C}$  accuracy
- No more than 1 Hz sampling rate (once every second)
- Body size 15.5mm x 12mm x 5.5mm
- 4 pins with 0.1" spacing

## DHT22 / AM2302 (Wired version)

- Low cost
- 3 to 5V power and I/O
- 2.5mA max current use during conversion (while data)
- Good for 0–100% humidity readings with 2–5% accuracy
- Good for  $-40$  to  $80^{\circ}\text{C}$  temperature readings  $\pm 0.5^{\circ}\text{C}$  accuracy
- No more than 0.5 Hz sampling rate (once every 2 seconds)
- Body size 15.1mm x 25mm x 7.7mm
- 4 pins with 0.1" spacing



Adafruit provides a library to work with the DHT22 sensor

### Syntax to read temperature and humidity

```
temperature,humidity=Adafruit_DHT.read_retry().
```

## Actuator

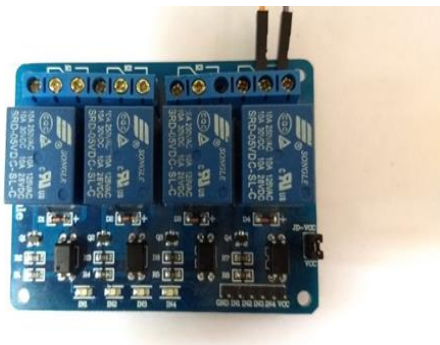
Mechanical/Electro-mechanical device

Converts energy into motion

Mainly used to provide controlled motion to other components

## Relay

Relays are the switches which aim at closing and opening the circuits electronically as well as electromechanically. It controls the opening and closing of the circuit contacts of an electronic circuit.



## Program For Temperature Dependent Auto Cooling System

### System Overview

Sensor and actuator interfaced with Raspberry Pi

Read data from the sensor

Control the actuator according to the reading from the sensor

Connect the actuator to a device

### Requirements

DHT Sensor (DHT22/AM2302)

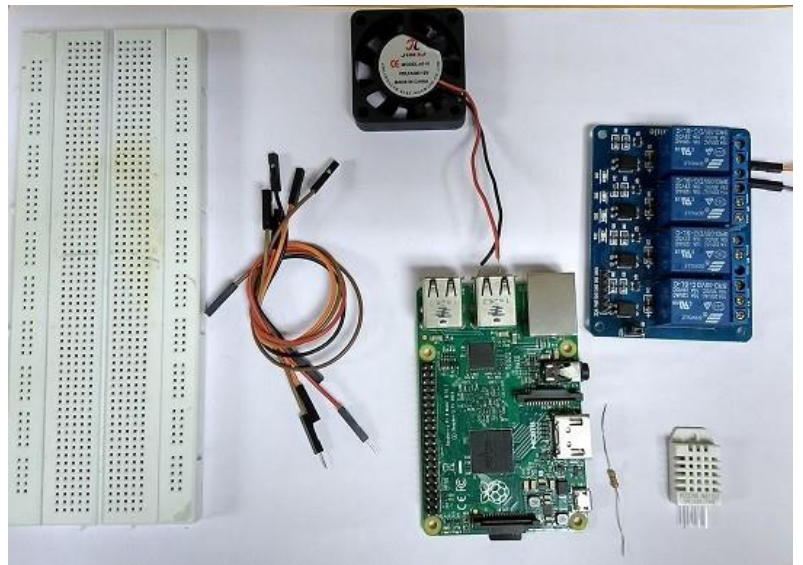
4.7K ohm resistor

Relay

Jumper wires

Raspberry Pi

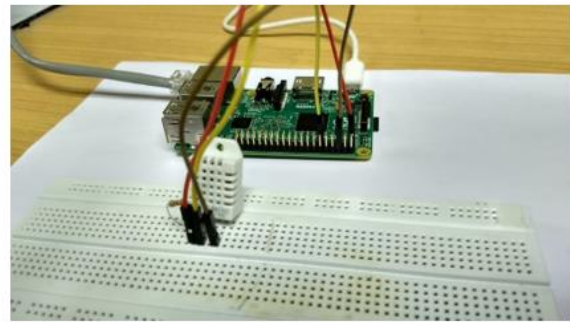
Mini fan





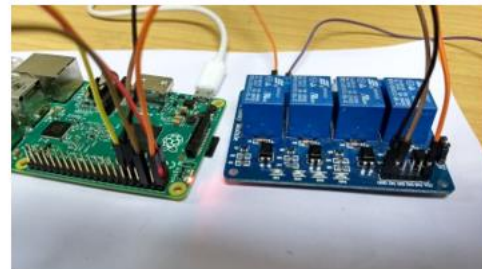
### Sensor interface with Raspberry Pi

- Connect pin 1 of DHT sensor to the 3.3V pin of Raspberry Pi
- Connect pin 2 of DHT sensor to any input pins of Raspberry Pi, here we have used pin 11
- Connect pin 4 of DHT sensor to the ground pin of the Raspberry Pi



### Relay interface with Raspberry Pi

- Connect the VCC pin of relay to the 5V supply pin of Raspberry Pi
- Connect the GND (ground) pin of relay to the ground pin of Raspberry Pi
- Connect the input/signal pin of Relay to the assigned output pin of Raspberry Pi (Here we have used pin 7)



Adafruit provides a library to work with the DHT22 sensor

- Install the library in your Pi-
  - Get the clone from GIT
 

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.g...
```
  - Go to folder Adafruit\_Python\_DHT

```
cd Adafruit_Python_DHT
```
  - Install the library
 

```
sudo python setup.py install
```

### Connection: Relay(Switch)

Connect the relay pins with the Raspberry Pi as mentioned in previous slides

**Set the GPIO pin connected with the relay's input pin as output in the sketch**

```
GPIO.setup(13,GPIO.OUT)
```

Set the relay pin high when the temperature is greater than 30

### Connection: Fan

- Connect the Li-po battery in series with the fan
  - NO terminal of the relay -> positive terminal of the Fan.
  - Common terminal of the relay -> Positive terminal of the battery
  - Negative terminal of the battery -> Negative terminal of the fan.
- Run the existing code. The fan should operate when the surrounding temperature is greater than the threshold value in the sketch

```

import RPi.GPIO as GPIO
from time import sleep
import Adafruit_DHT                                     #importing the Adafruit library

GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
sensor = Adafruit_DHT.AM2302                            # create an instance of the sensor type

print ('Getting data from the sensor')
#humidity and temperature are 2 variables that store the values received from the sensor
humidity, temperature = Adafruit_DHT.read_retry(sensor,17)
print ('Temp={0:0.1f}*C humidity={1:0.1f}%'.format(temperature, humidity))

#code for fan operating
GPIO.setup(13,GPIO.OUT)
if temperature > 30:
    print ('Temp > 30')
    GPIO.output (FAN,0)
    print('Fan on')
    sleep(5)
    print('Fan off ')
    GPIO.output(FAN,1)
else:
    GPIO.output(FAN,1)
    print ('Temp below max value.FAN OFF')

```

**Output :** The fan is switched ON whenever the temperature value reaches above Threshold value set in the code.