UNIT III

FUNDAMENTALS OF PYTHON PROGRAMMING & RASPBERRY PI

syllabus

- Introduction to python programming, Data Types & Data Structures, working with functions, Modules & Packages, File Handling, classes, REST full Web Services, Client Libraries, Introduction & programming Raspberry Pi3, Interfaces, Integrating Input Output devices with Raspberry Pi3.

Introduction to python programming

- Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Python is a high-level, interpreted language used in embedded systems for scripting, rapid development, and interfacing with hardware like the Raspberry Pi and ESP32 microcontrollers.
- **Why to Learn Python?**
  - Python is consistently rated as one of the world's most popular programming languages.
  - Python is fairly easy to learn, so if you are starting to learn any programming language then Python could be your great choice.
  - Today various Schools, Colleges and Universities are teaching Python as their primary programming language.

Introduction to python programming

- Python is Open Source which means its available free of cost.
- Python is simple and so easy to learn
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries include AI, ML etc.
- Python is much in demand and ensures high salary.

Applications of Python

- Python is used in:
  - Web development
  - Data Science
  - Artificial Intelligence
  - Machine Learning
  - Embedded Systems
  - Automation
  - Game development

Applications of Python

- The latest release of Python is 3.x. As mentioned before, Python is one of the most widely used language over the web.
- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the
- eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Applications of Python

- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.

Basic Python Example

- print("Hello, World!")
- Hello, World!
- **Python Variables Example**
- a = 10
- b = 20
- c = a + b
- print(c)
- 30

Why Python is Popular

- Simple syntax
- Used in modern technologies like AI and IoT
- High demand in industry

Data Types & Data Structures

- Data types define the type of data a variable can store. Data structures are used to organize and store data efficiently.
- **Data Types**
- Python has built-in data types:
- **1. Numeric**
- int → 10
- float → 10.5
- complex → 3+2j
- **Example**
- a = 10
- b = 3.5
- a = 10
- b = 3.5

Data types:

- Data types represent the size and type of the data
- In python data types are classified into two types

DATA TYPES


Single value/
Individual Datatype
Numaric
Data type
Integer
Float
Complex
Boolean
Data type
True
False
Multi value/
Collection Datatype
String
List
Tuple
Set
Dictionary

Single value/Individual data type:

- The data type which has only one value and it stored into a variable is called as Single value/ Individual data type

Example:

```
a = 10      ---> int
b = 15.5    ---> float
c = 3+4j    ---> complex
d = True    ---> bool
```

Multivalue/ Collection data type:

- The data type which has more than one value or Group of values are called as Multivalue/ Collection data type

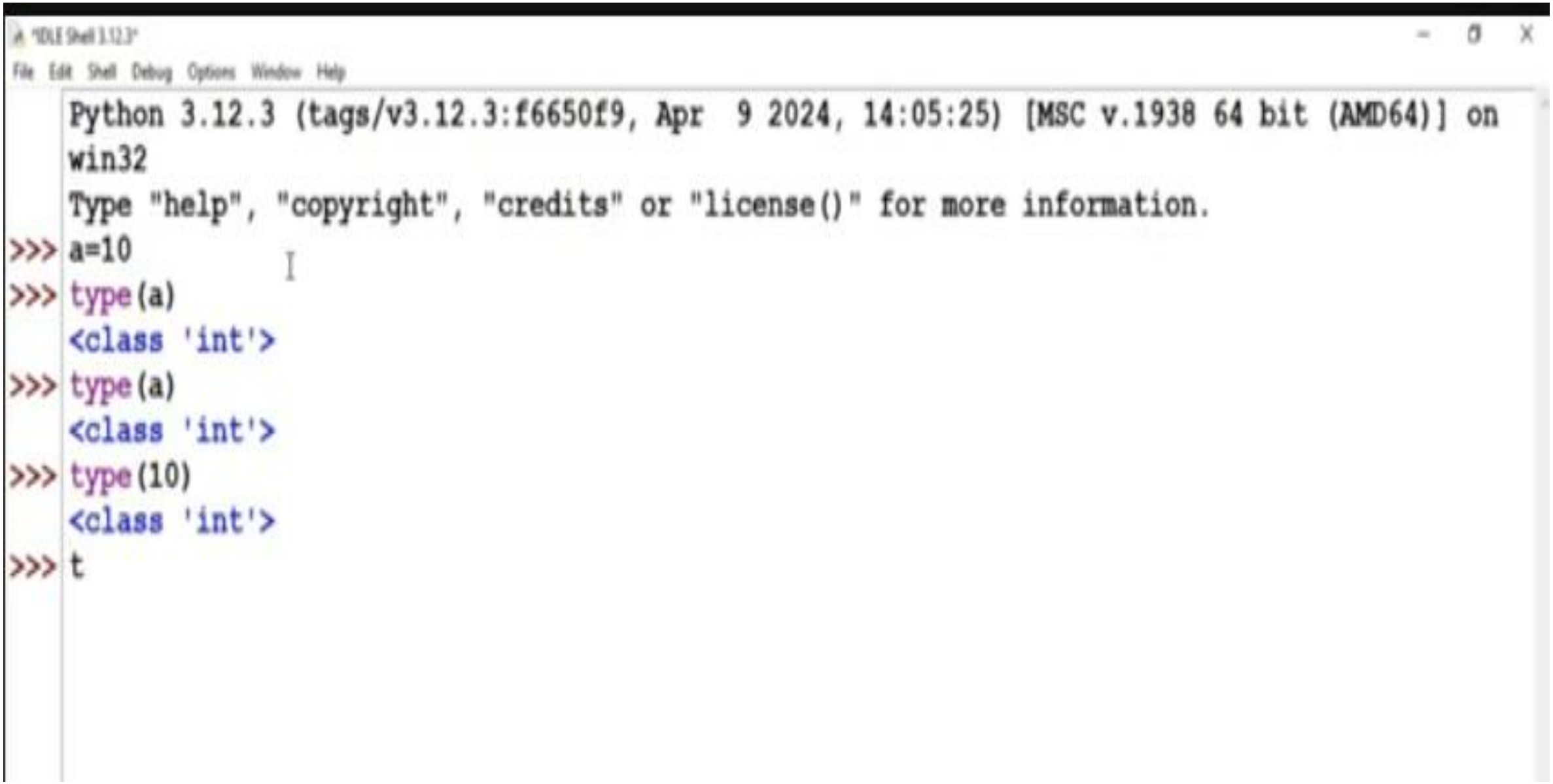
Example:

```
a = 'hai'                    ---> str
b = [1, 2, 3, 4, 5]          ---> list
c = (1, 2, 3, 4, 5)          ---> tuple
d = {1, 2, 3}                ---> set()
e = {1:'a', 2:'b', 3:'c'}    ---> dict
```

Integer:

- Integer are the real number which does not have decimal point
- Integer can be a positive number or a negative number

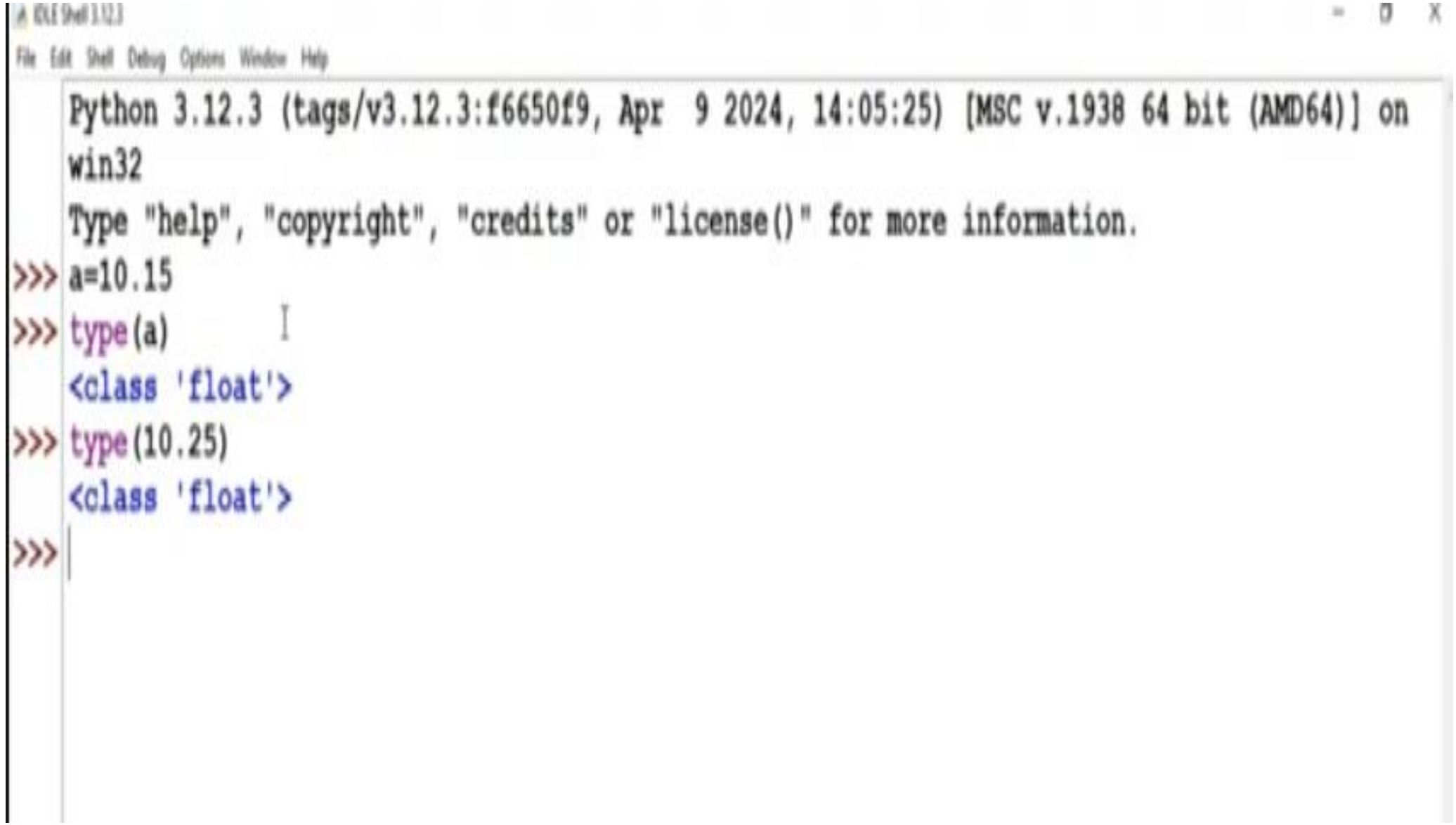
- **Representation** -> " int "
- **Default value** -> 0
- **Example** -> 10, -50


File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=10
>>> type(a)
<class 'int'>
>>> type(a)
<class 'int'>
>>> type(10)
<class 'int'>
>>> t

Float:

- Float is a real number with having a decimal point
- Float can be either positive or negative

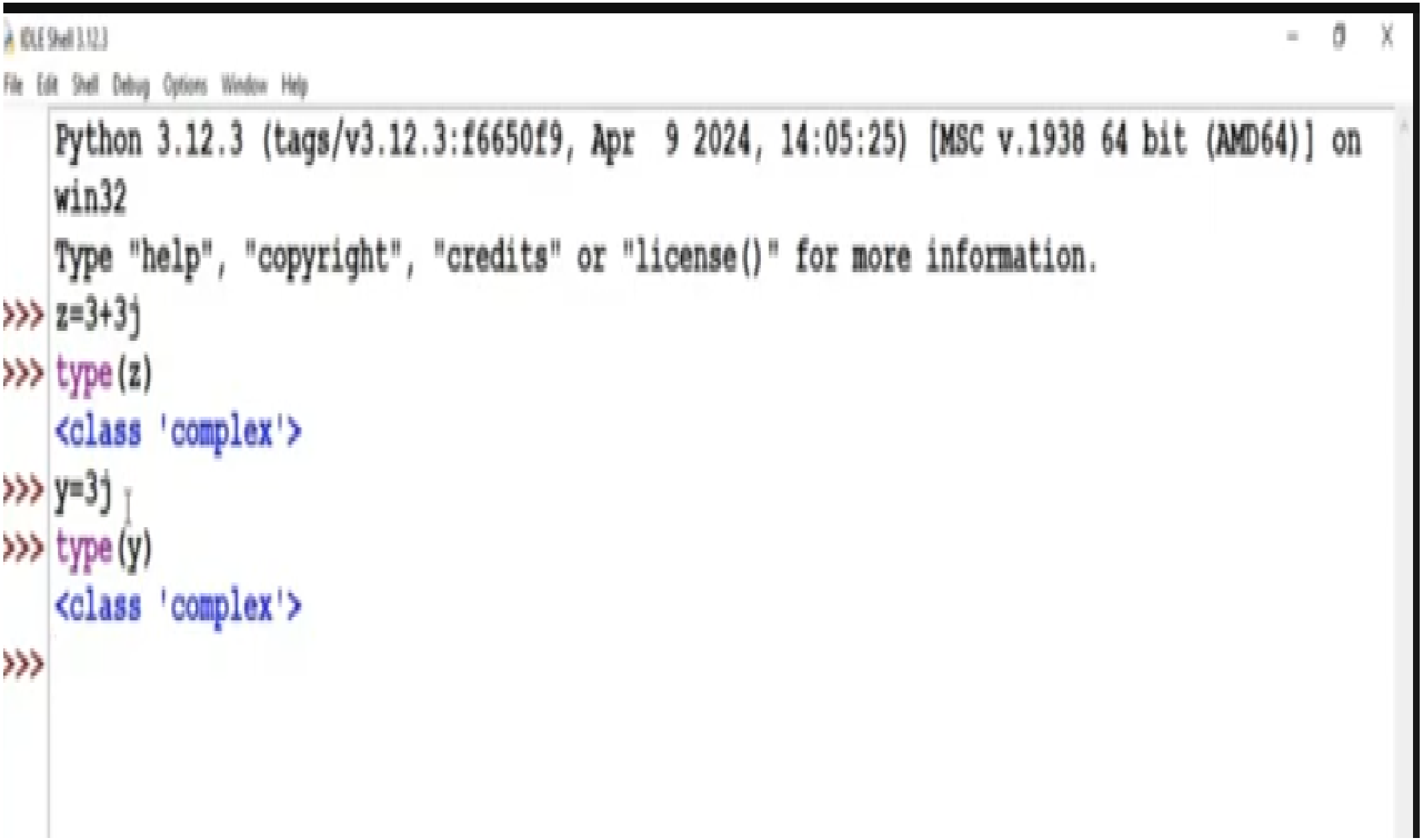
- **Representation** -> " float "
- **Default value** -> 0.0
- **Example** -> 4.5, -4.5


File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a=10.15
>>> type(a)
<class 'float'>
>>> type(10.25)
<class 'float'>
>>>

Complex:

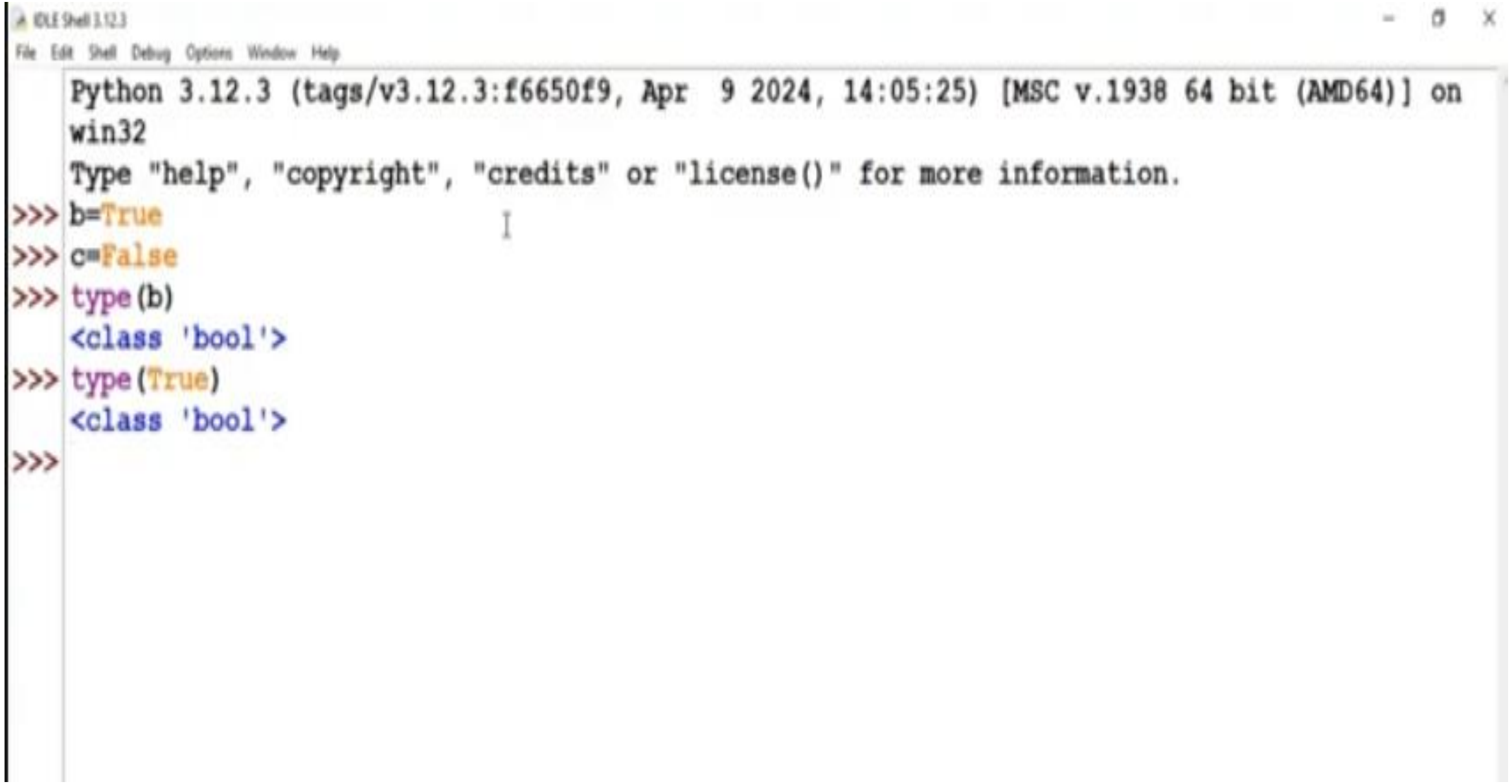
- The number which has both real and imaginary part is called as complex number
- For complex number the real part is not mandatory

- **Representation** -> " complex "
- **Default value** -> 0j
- **Example** -> 3+4j


File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> z=3+3j
>>> type(z)
<class 'complex'>
>>> y=3j
>>> type(y)
<class 'complex'>
>>>

Boolean:

- Boolean data type is used to represent the default and non-default data
- Boolean data type has two values
  - i) True
  - ii) False
- In python all the default values are represented as Boolean False and all the non-default values are represented as Boolean True
- **Representation** -> " bool "
- **Default value** -> False


IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> b=True
>>> c=False
>>> type(b)
<class 'bool'>
>>> type(True)
<class 'bool'>
>>>

Integer:

- Integer are the real number which does not have decimal point
- Integer can be a positive number or a negative number

- **Representation** -> " int "
- **Default value** -> 0
- **Example** -> 10, -50

Float:

- Float is a real number with having a decimal point
- Float can be either positive or negative

- **Representation** -> " float "
- **Default value** -> 0.0
- **Example** -> 4.5, -4.5

Complex:

- The number which has both real and imaginary part is called as complex number
- For complex number the real part is not mandatory

- **Representation** -> " complex "
- **Default value** -> 0j
- **Example** -> 3+4j

Boolean:

- Boolean data type is used to represent the default and non-default data
- Boolean data type has two values
  - i) True
  - ii) False
- In python all the default values are represented as Boolean False and all the non-default values are represented as Boolean True
- **Representation** -> " bool "
- **Default value** -> False

String:

- String is collection of characters, the characters can be **alphabets, numbers** or any **special symbols** enclosed between single quotes, double quotes or triple quotes

- **Syntax**

```
var = 'val1val2val3.....valn'
```

String:

- **Representation** -> " str "
- **Default value** -> ' '
- **Example**

```
v1 = 'abcd'
v2 = '1234'
v3 = '$@#&'
v4 = 'abc@123'
```

```
Python 3.12.3 (tags/v3.12.3:f6650f9, Apr  9 2024, 14:05:25) [MSC
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inform
>> a='happy'
>> a
'happy'
>> type(a)
<class 'str'>
```

Task:

```
>>> S = 'hello welcome to python'
>>> len(S)  = ?
```

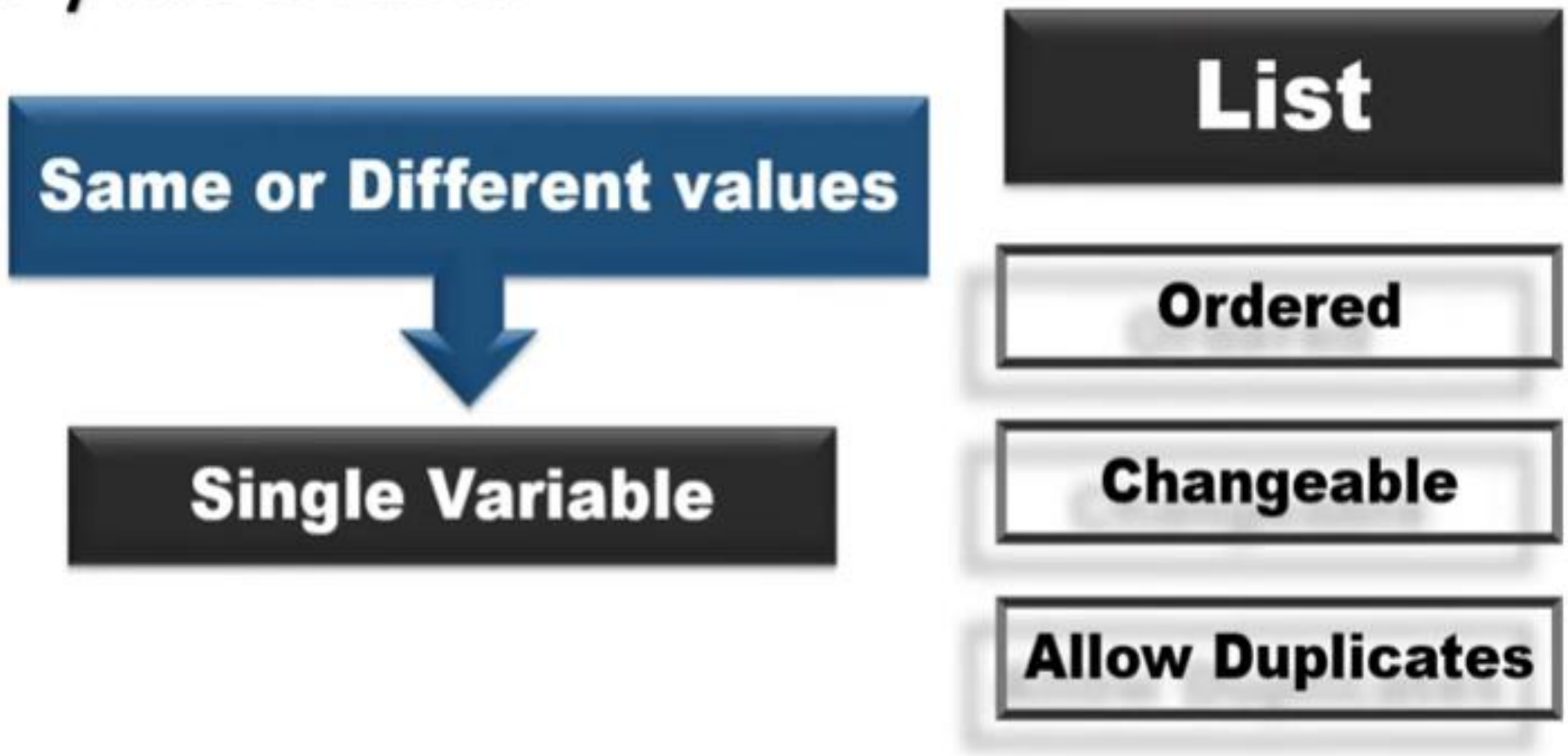
a) 20 b) 23 c) 26 d) 27

Find the length of the string

Python Lists

Python Lists

Python Variables in Tamil | Python Tutorial for Beginners-Pytho...

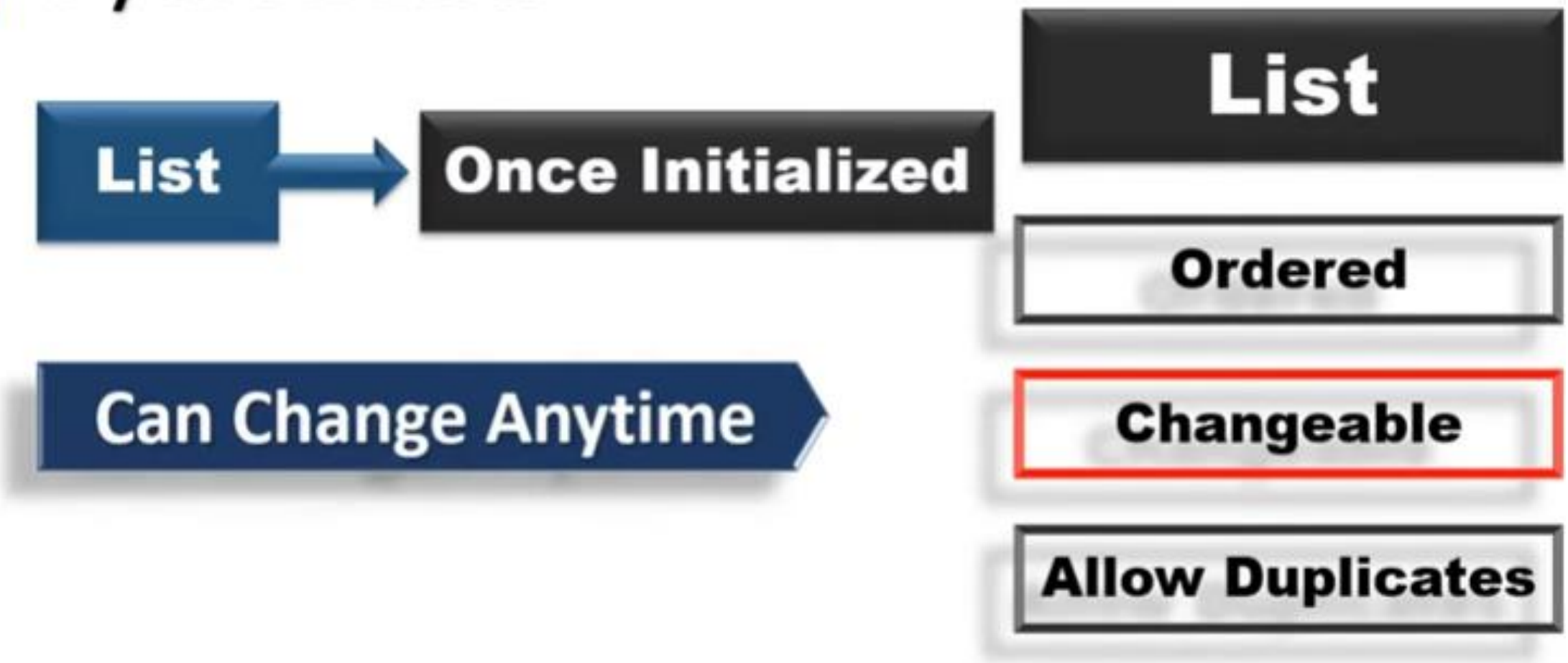


Python Lists

List

- **Ordered**
- **Changeable**
- **Allow Duplicates**

Python Lists



Python Lists

Appears many time

List

- Ordered
- Changeable
- Allow Duplicates

Python Lists

Creating List

Empty List

Square bracket [ ]

list() Constructor

```
listname=[]
```

Python Lists

Creating List

Empty List

Square bracket [ ]

```
listname=[]
print("Empty List",listname)
```

Output

```
Empty List []
```

1. List Operations

- **List is a sequence of values, which can be of different types.**
- **The values in list are called "elements" or ''items''**
- **Each elements in list is assigned a number called "position" or "index".**
- A list that contains no elements is called an **empty list**. They are created with **empty brackets[]**
- A list within another list is called **nested list.**

LIST OPERATIONS:

- **1.Concatenation of list**
- **2.Repetition of list**

Concatenation:

the '+' operator concatenates the list

```
a = [1,2,3]
b = [4,5,6]
c = a+b
print (c)
```

Repetition:

the '*' operator repeats a list a given number of times

```
>>> a = [1,2,3]
>>> print (a*2)= [1,2,3,1,2,3]
```

List Methods

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

#List Methods

Example for clear method:

```
a=[10,20.3,'c',"hello"]
print(a)
a.clear()
```

Example for copy,count method:

```
a=[10,20.3,'c',"hello"]
print(a)
b=a.copy()
print(b)
x=b.count(10)
print(x)
y=b.count("hello")
print(y)
```

Output:

```
[10, 20.3, 'c', 'hello']
[10, 20.3, 'c', 'hello']
1
1
```

Example1:

```
fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)

print(fruits)
```

Output:

['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']

Example 2:

```
fruits = ['apple', 'banana', 'cherry', ‘banana’]

fruits.remove("banana") # removes only the first occurrence of "banana" in the list

print(fruits)
```

Output:

['apple', 'cherry',’banana’]

Example 3

```
cars = ['Ford', 'BMW', 'Volvo']
cars.sort()
print(cars)
```

Output:

```
['BMW', 'Ford', 'Volvo']
```

Example 4:

```
subjects=[‘Eng’ , ’Tamil’ , ’Maths’]
del subjects[0]
print(subjects)
```

Output:

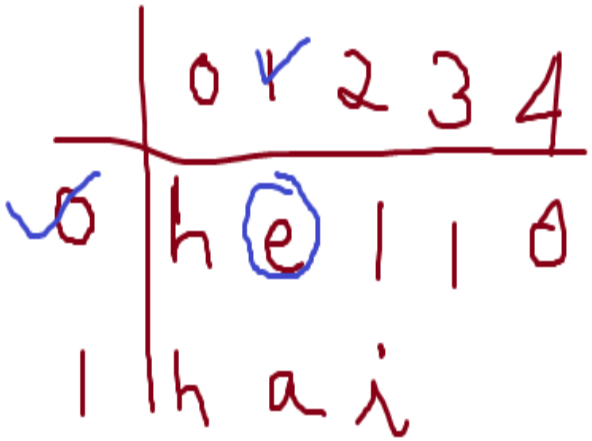
```
Tamil , Maths
```

Example 5:

```
a=[ “hello” , ”hai” ]
print(a[0][1])
```

Output:

```
e
```



```
# Common Elements in two list
fruits1=['apple','mango']
fruits2=['orange','banana','apple']
s1=set(fruits1)  #convert into sets
s2=set(fruits2)
s3=s1.intersection(s2)
common=list(s3)
print(common)
```

Output:

['apple']

Reverse Example:

```
a=[1,2,3,4]
a.reverse()
print("reversed list is",a)
```

Output:

Reversed list is [4,3,2,1]

2. Tuples

- **A tuple is a collection of elements of dissimilar datatype**.
- Values in tuple are enclosed in parentheses and separated by comma.
- The elements in the tuple **cannot be modified** as in list (i.e) tuple are **immutable objects**.

```
t1 = ('a','b','c','d')
t2=()
t3=(10,)
print(t3)
print(t1[0:2])
```

Tuple Assignment

- **Each value is assigned to its respective variable**.
- All expressions on right side are evaluated before any of assignments.
- **Example:**

```
student=('AAA',123)

name,roll=student[0],student[1]

print(name)

print(roll)
```

Output:

```
AAA

123
```

Tuple Built in functions

Methods	Description
len()	Returns number of items
max()	Maximum item
min()	Minimum item
sum()	Sum of items
any()	Returns true if there exists any item, checks if at least one element in an iterable evaluates to True
sorted()	Sort elements in tuple
reversed()	Sort elements in reverse order
enumerate()	Provides index to each elements in a tuple

3. Dictionaries

- **A dictionary is an unordered set of key: value pair.**
- In a list, the indices have to be integers; but, in a dictionary, it can be any type.
- **A dictionary contains a collection of indices, which are called keys, and a collection of values.**
- Each key is associated with a single value.
- The association of a key and a value is called a **key-value pair.**
- **Dictionary is created by enclosing with curly braces {}.**

Example

```
>>>dictionary={"RollNo":101,2:(1,2,3),"Name":"
   Ramesh",20:20.50,”Loc":['Chennai']}
>>> dictionary
{'RollNo': 101, 2: (1, 2, 3), 'Name': 'Ramesh', 20:
   20.5, 'Loc': ['Chennai']}
```

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Creating a dictionary

Syntax:

dict={} # empty dictionary

dict={key1:value1,key2:value2…..,keyn:valuen}

Key: user defined variable names

Value: values assigned for the key

Example:

member_1={'Name':‘AAA','Dept':'IT'}

Accessing Elements on a dictionary

```
member={'Name':‘AAA','Dept':'IT'}
x=member['Name']
print(x)
```

Output

AAA

Get method

```
member={'Name': 'AAA','Dept':'IT'}
x=member.get('Dept')
print(x)
```

Output

IT

- member.get('Dept') retrieves the value associated with the key 'Dept', which is 'IT'.

Adding Dictionary Entries

```
dict={'name’:’XXX','age':24,'ranking':'5th‘}
print(dict)
dict['status']='regular‘
print(dict)
```

Output:

{'name': ‘XXX', 'age': 24, 'ranking': '5th'}

{'name': ‘XXX', 'age': 24, 'ranking': '5th', 'status': 'regular'}

Modifying Dictionaries Entry

```
dict={'name’:’xxx','age':24,'ranking':'5th'}
print(dict)
dict['ranking']='3rd‘
print(dict)
```

Output

{'name': ‘xxx', 'age': 24, 'ranking': '5th'}

{'name': ‘xxx', 'age': 24, 'ranking': ‘3rd'}

Removing or deleting elements from a dictionary

```
dict={'name’:’xxx','age':24,'ranking':'5th'}
print(dict)
dict.pop('age')
print(dict)
```

Output:

{'name': ‘xxx', 'age': 24, 'ranking': '5th'}

{'name': ‘xxx', 'ranking': '5th'}

Popitem()

```
dict={'name’:’xxx','age':24,'ranking':'5th'}
print(dict)
dict.popitem()
print(dict)
```

Output

{'name': ‘xxx', 'age': 24, 'ranking': '5th'}

{'name': ‘xxx', 'age': 24}

Clear()

```
dict={'name’:’xxx','age':24,'ranking':'5th'}
dict.clear()
print(dict)
```

Output

{} # empty dictionary

Deleting a dictionary

```
dict={'name’:’xxx','age':24,'ranking':'5th'}
del dict
print(dict)
```

Output:

Traceback error

Modules and Packages

• Python Modules

A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module.

• Modules in Python provides us the flexibility to organize the code in a logical way.

Example:

- In this example, we will create a module named as file.py which contains a function func that contains a code to print some message on the console.

- Let's create the module named as file.py.

```
#displayMsg prints a message to the name being passed.

def displayMsg(name)
    print("Hi "+name);
```

Loading the module in our python code

- ❑ The import statement
- ❑ The from-import statement

• The import statement

• The import statement is used to import all the functionality of one module into another. Here, we must notice that we can use the functionality of any python source file by importing that file as the module into another python source file.

• We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.

• The syntax to use the import statement is given below.

```
import module1,module2,…….. module n
```

Example:

```
import file;
name = input("Enter the
name?")
file.displayMsg(name)
```

Output:

```
Enter the name?John
Hi John
```

The from-import statement

- Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module. This can be done by using from? import statement. The syntax to use the from-import statement is given below.

```
from < module-name> import <name 1>, <name 2>..,<name n>
```

calculation.py:

- #place the code in the calculation.py
  - def summation(a,b):
    return a+b
  - def multiplication(a,b):
    return a*b;
  - def divide(a,b):
    return a/b;

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py

a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))

 ‘’’we do not need to specify the module name while accessing summation()’’’
```

Output:

- Enter the first number10
- Enter the second number20
- Sum = 30

File Handling in Python

- File handling refers to the process of performing operations on a file, such as creating, opening, reading, writing and closing it through a programming interface.
- It involves managing the data flow between the program and the file system on the storage device, ensuring that data is handled

Why do we need File Handling

- To store data permanently, even after the program ends.
- To access external files like .txt, .csv, .json, etc.
- To process large files efficiently without using much memory.
- To automate tasks like reading configs or saving outputs.

Opening a File

- To open a file, we can use open() function, which requires file-path and mode as arguments.
- *file = open('filename.txt', 'mode')*
- **Closing a File**
- The file.close() method closes the file and releases the system resources. If the file was opened in write or append mode, closing ensures that all changes are properly saved.
- *file = open("geek.txt", "r")*
- *# Perform file operations*
- *file.close()*

Libraries in Python

- In Python, a library is a group of modules that contain functions, classes and methods to perform common tasks like data manipulation, math operations, web scraping and more.
- Python libraries make coding faster, cleaner and more efficient by providing ready-to-use solutions for different domains such as data science, web development, machine learning and automation.

Popular External Python Libraries

01 Numpy

02 Tensorflow

03 Pandas

04 SciPy

05 Matplotlib

06 Pytorch

07 Scikit-learn

Working of Python Library

- When you import a library in Python, it gives access to pre-written code stored in separate modules. In simple terms instead of writing the logic for a task, you import the library that already has it.
- **For example**, on Windows, libraries are stored as .dll (Dynamic Link Libraries) and on Linux/macOS as .so files. When you run your code, Python automatically loads these modules and makes their functions available to use.

Types of Python Library

- Python libraries are divided into two main types:
  - **1. Built-in Python Standard Library**
  - **2. External Python Libraries**

1. Built-in Python Standard Library

- It is a collection of modules that come bundled with every Python installation, we don't need to install anything separately. Most of these modules are written in C for better performance.
- **Examples of built-in modules:**
- **math:** Mathematical operations
- **os:** Interact with the operating system
- **datetime:** Date and time operations
- **random:** Generate random numbers
- **json:** Handles JSON data encoding and decoding.

2. External Python Libraries

- External (third-party) libraries are not included with Python by default. You can install them easily using the pip package manager. Popular External Python Libraries:
- NumPy: It, short for Numerical Python, is the core library for numerical and scientific computing in Python. It provides powerful tools for creating and manipulating arrays, matrices and multidimensional data.
- Pandas: It is a data analysis and manipulation library built on top of NumPy. It introduces data structures like DataFrame and Series that make it easy to handle structured data efficiently.

External Python Libraries

- **Matplotlib:** It is a data visualization library that helps you create a wide variety of static, animated and interactive plots. It supports charts such as bar graphs, line charts, histograms and scatter plots.
- **SciPy:** It, short for Scientific Python, extends the functionality of NumPy by providing tools for advanced mathematical, scientific and engineering computations. It includes modules for optimization, integration, signal processing and linear algebra.
- **TensorFlow:** It is an open-source machine learning and deep learning framework developed by Google. It allows developers to build and train neural networks for tasks such as image recognition, natural language processing and predictive

- Scikit-learn: It is a popular machine learning library built on top of NumPy and SciPy. It offers simple and efficient tools for classification, regression, clustering and dimensionality reduction.
- Scrapy: It is a web scraping and data extraction library designed to efficiently crawl websites and gather structured information. It allows developers to create spiders that automatically navigate web pages and collect data.
- PyTorch: It, developed by Facebook's AI Research Lab, is a deep learning framework that provides dynamic computation graphs and easy debugging. It supports GPU acceleration, making it suitable for high-performance training of neural networks.

- **PyGame:** It is a cross-platform library used for developing 2D games and multimedia applications. It provides modules for handling graphics, sound and user input with ease.
- **PyBrain:** It, short for Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network, is a beginner-friendly machine learning library. It offers pre-built algorithms and flexible tools for training neural networks and reinforcement learning models.

Using Libraries in Python Programs

- To use any library, it first needs to be imported into your Python program using the import statement. Once imported, you can directly call the functions or methods defined inside that library. You can import libraries in three main ways:
- **Import the entire library:** import library_name for example, import math
- **Import a specific function or class:** from library_name import function_name for example, from math import sqrt
- **Import a library with an alias:** import library_name as alias for example, import pandas as pd

Example 1:

- import math
- A = 16
- print(math.sqrt(A))

- 4.0

Example 2:

- from numpy import array, mean
- a = array([10, 20, 30, 40, 50])
- print(mean(a))

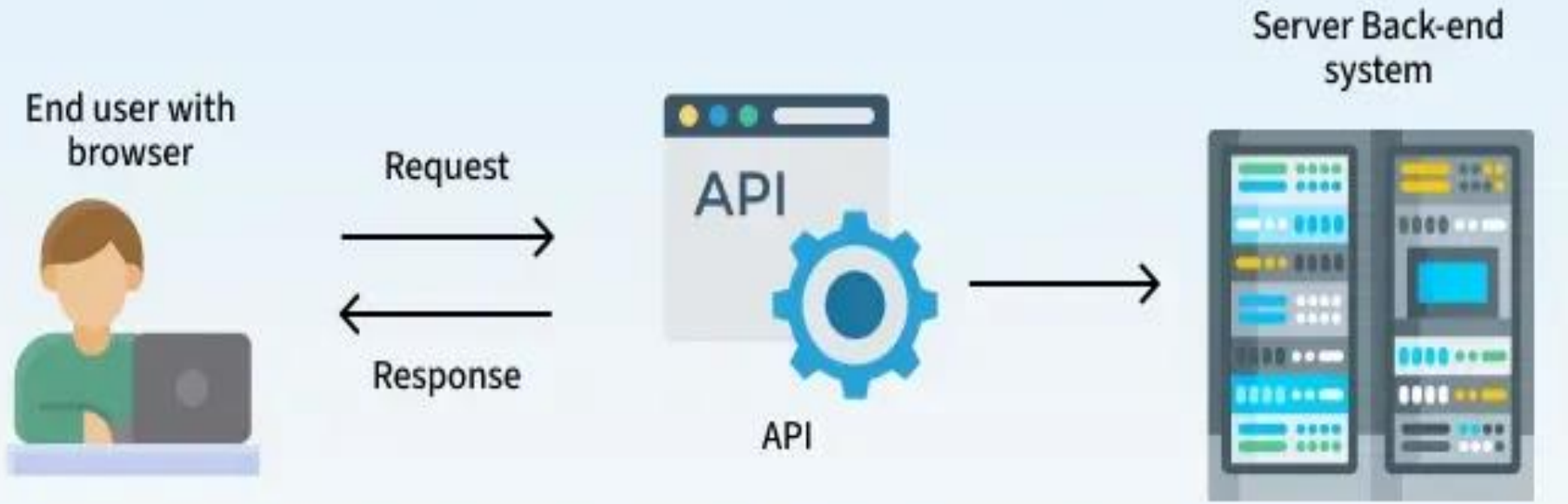
- 30

RESTful Web Services

- **RESTful Web Services** are a way of designing and developing **web services** that use **REST (Representational State Transfer) principles.**
-
- **They enable applications to communicate over the web using standard HTTP methods, such as GET, POST, PUT and DELETE.**

What is an API?

APIs are like messengers that allow different software to talk to each other and share data seamlessly.



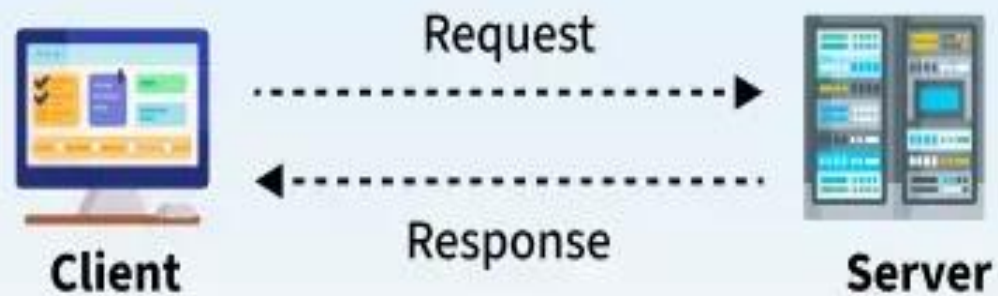
Python APIs allow us to easily send secure HTTP requests to interact with web services:

GET: Retrieve data from a server.

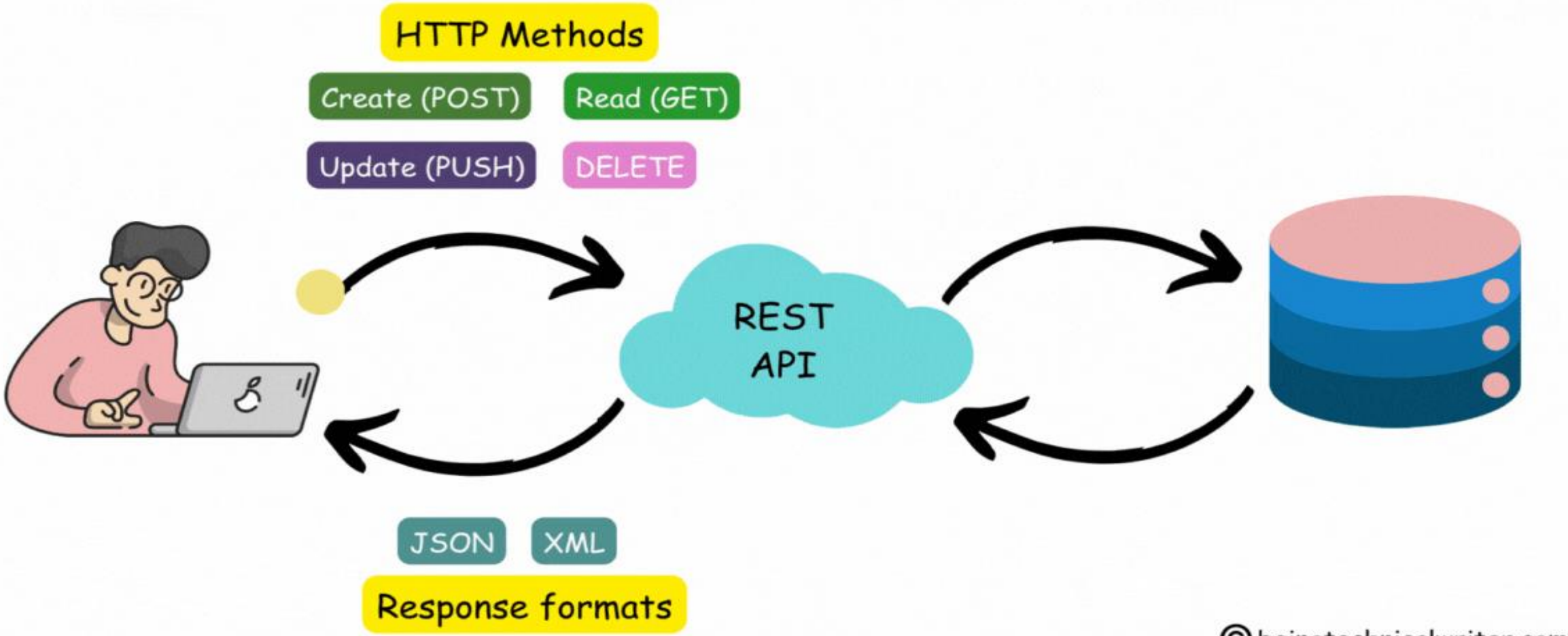
POST: Send data to a server.

PUT: Update existing data on a server.

DELETE: Remove data from a server.



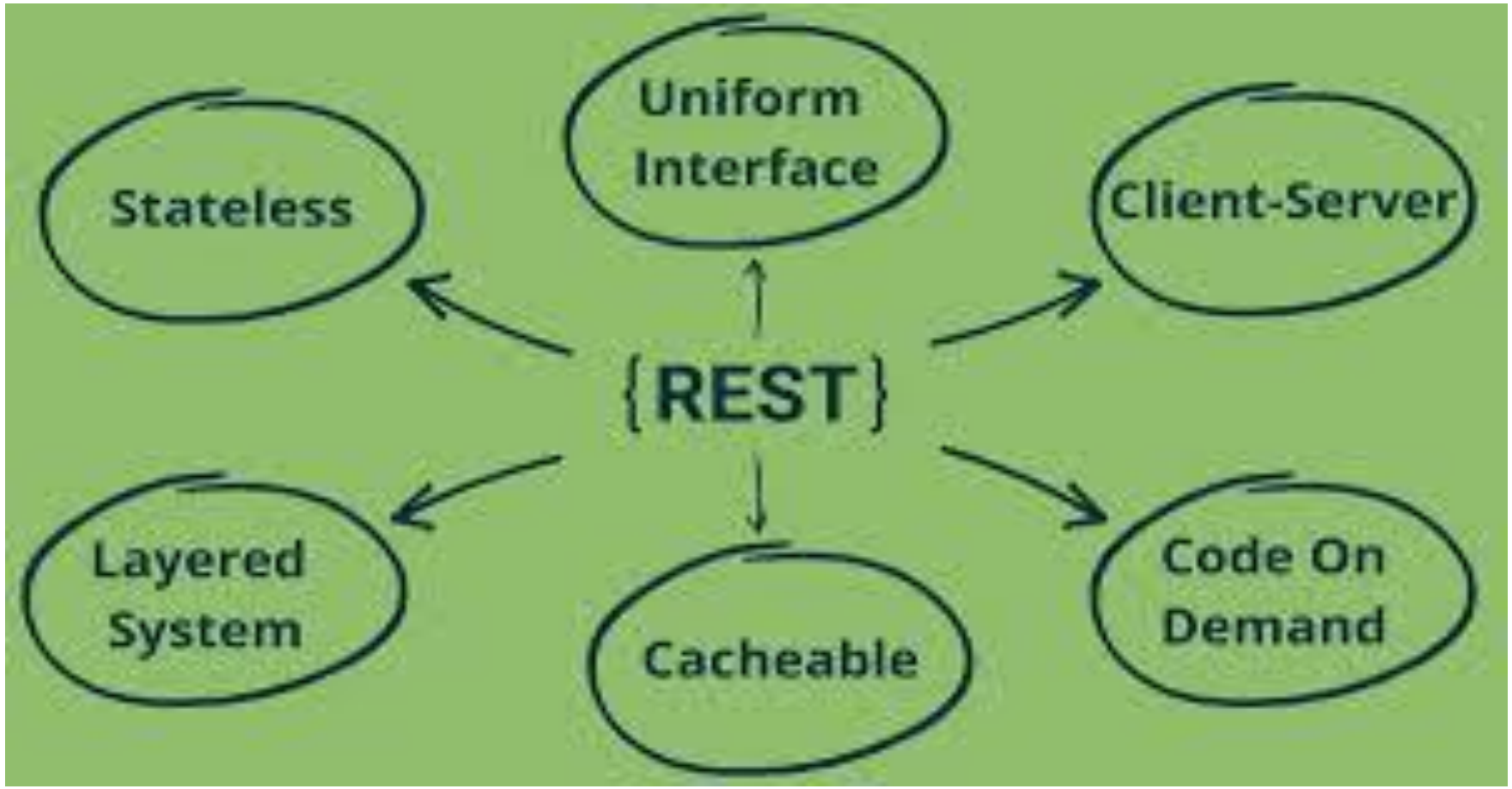
RESTful APIs



What is an API?

- An **application programming interface (API)** defines the rules that you must follow to communicate with other software systems. Developers expose or create APIs so that other applications can communicate with their applications programmatically.

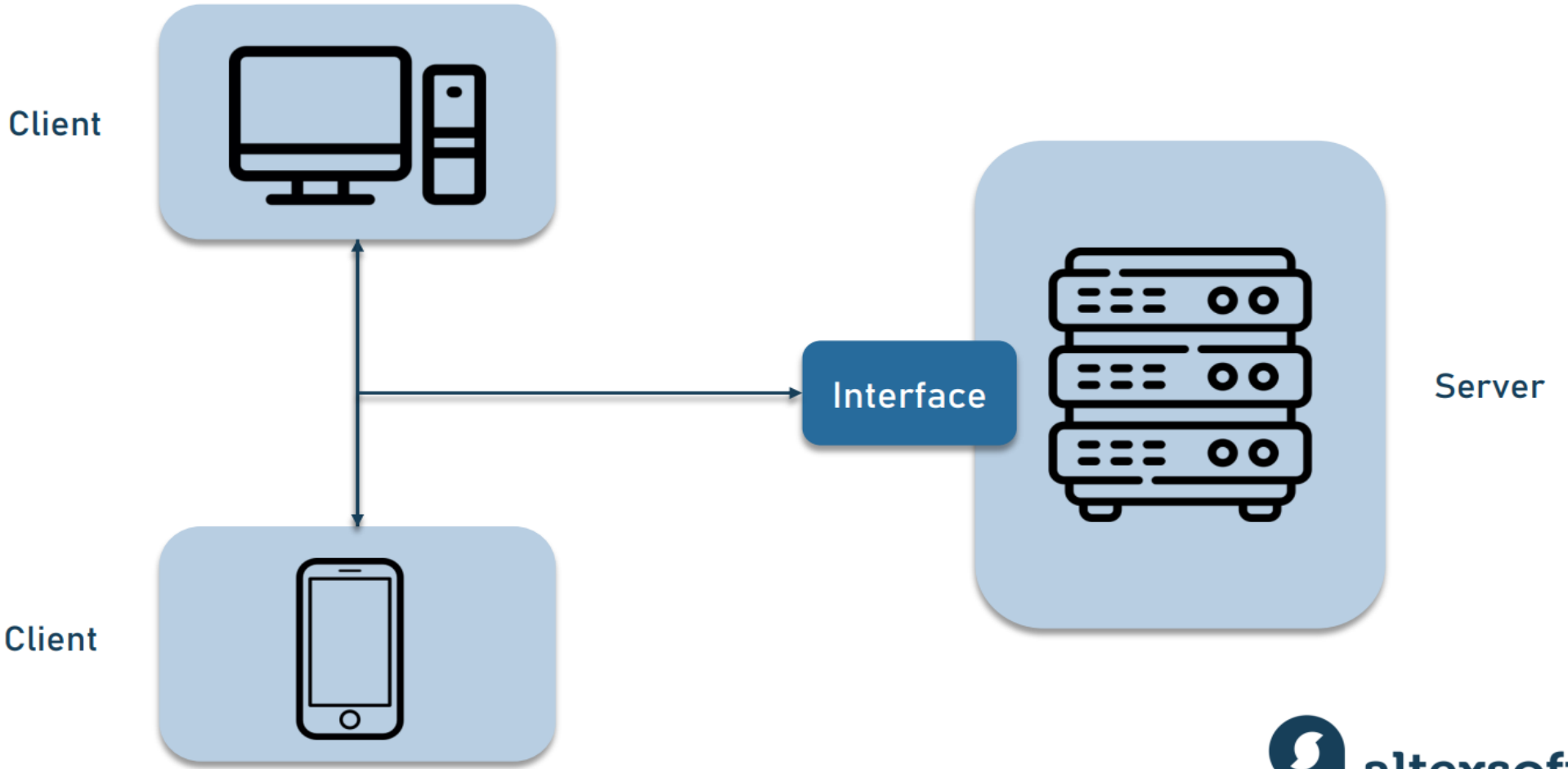
REST architectural style



uniform interface

- **The uniform interface** is fundamental to the design of any RESTful webservice. It indicates that the server transfers **information in a standard format**.
- The formatted resource is called a representation in REST.
- This format can be different from the internal representation of the resource on the server application.

UNIFORM INTERFACE



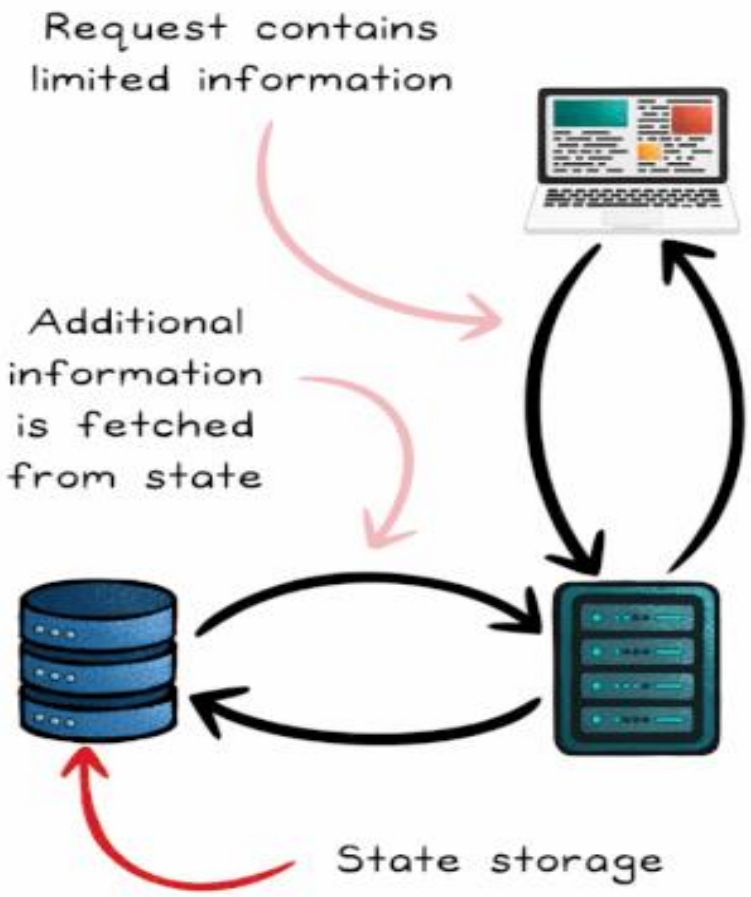
altexsoft

statelessness

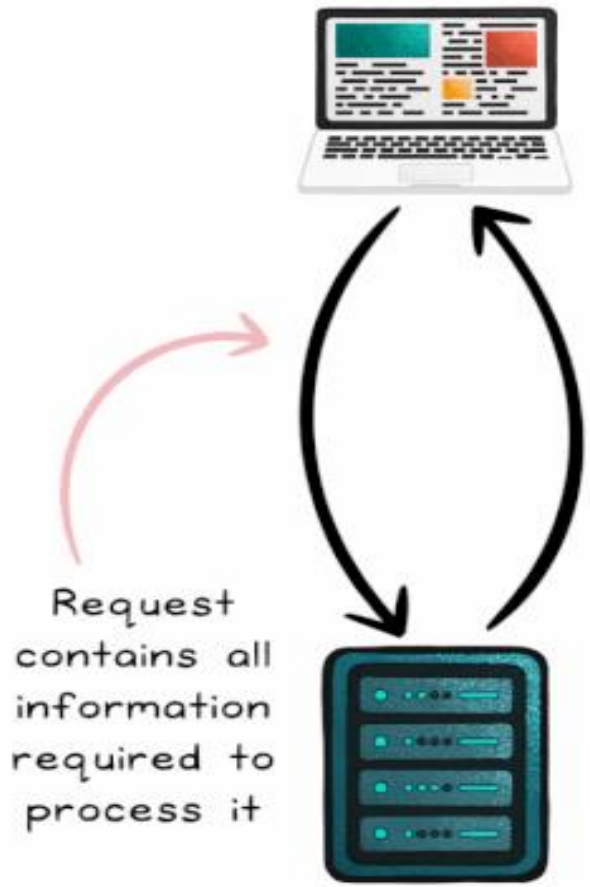
- **statelessness** refers to a communication method in which the server completes every client request independently of all previous requests.
- Clients can request resources in any order, and every request is stateless or isolated from other requests.

Stateful vs Stateless

Stateful

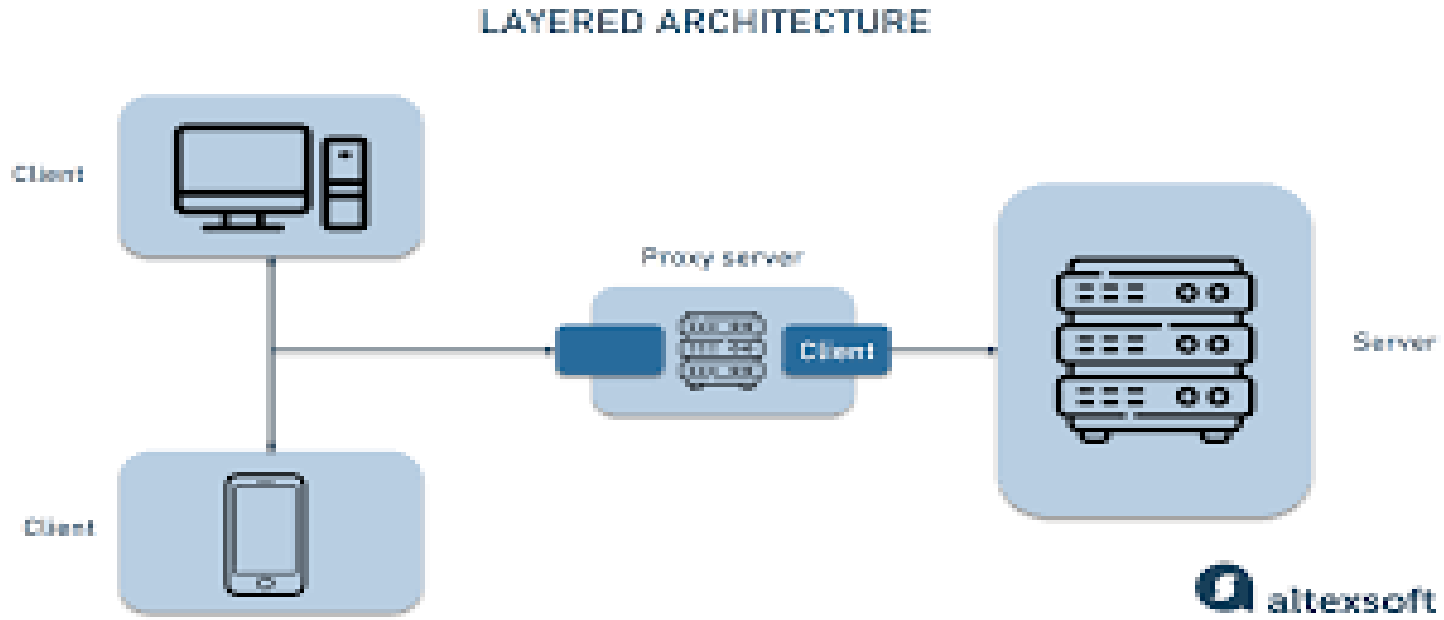


Stateless



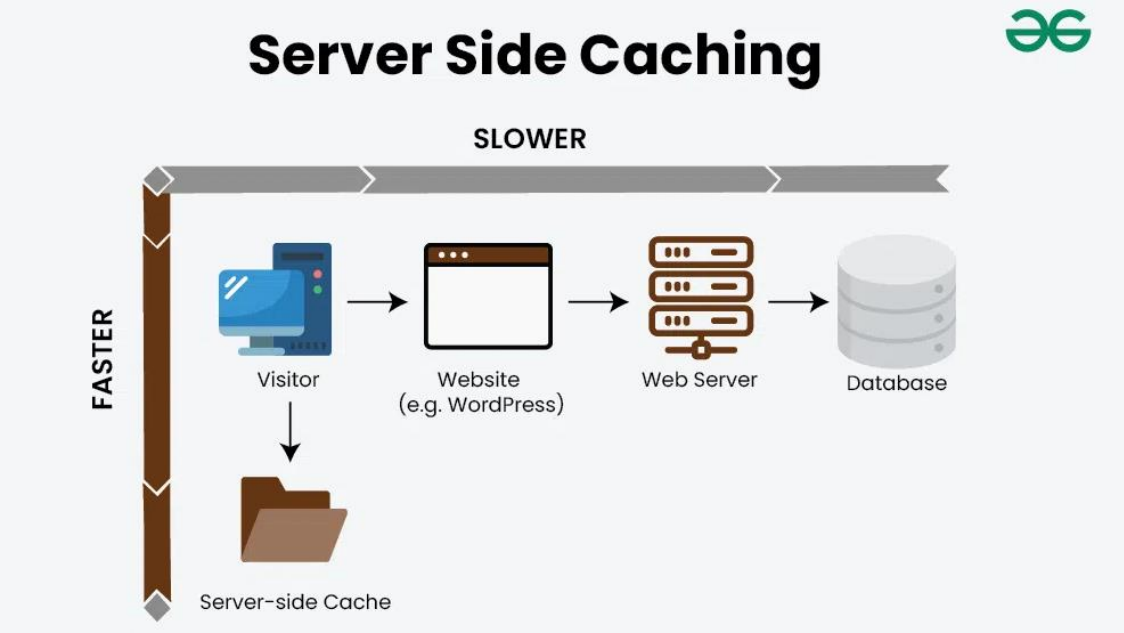
layered system architecture

- The client can connect to other authorized intermediaries between the client and server, and it will still receive responses from the server.



Cacheability

- RESTful web services support caching, which is the process of storing some responses on the client or on an intermediary to improve server response time.



Code on demand

- In REST architectural style, servers can temporarily extend or customize client functionality by transferring software programming code to the client.

5 ADVANTAGES OF REST APIS

- Flexibility
- Scalability
- Security
- Ease of use
- Interoperability

Raspberry-Pi a computer

- **What is a Raspberry Pi?**
- Raspberry pi is the name of the "credit card-sized computer board" developed by the Raspberry pi foundation, based in the U.K. It gets plugged in a TV or monitor and provides a fully functional computer capability.
- It is aimed at imparting knowledge about computing to even younger students at the cheapest possible price. Although it is aimed at teaching computing to kids, but can be used by everyone willing to learn programming, the basics of computing, and building different projects by utilizing its versatility.
- Raspberry Pi is developed by Raspberry Pi Foundation in the United Kingdom. The Raspberry Pi is a series of powerful, small single-board computers.
- Raspberry Pi is launched in 2012 and there have been several iterations and variations released since then.

Raspberry-Pi a computer

- Various versions of Raspberry Pi have been out till date. All versions consist of a Broadcom system on a chip (SoC) with an integrated ARM-compatible CPU and on-chip graphics processing unit (GPU).
- The original device had a single-core Processor speed of device ranges from 700 MHz to 1.2 GHz and a memory range from 256 MB to 1 GB RAM.
- To store the operating system and program memory Secure Digital (SD) cards are used. Raspbian OS which is a Linux operating system is recommended OS by Raspberry Pi Foundation. Some other third party operating systems like RISC OS Pi. Diet Pi, Kali, Linux can also be run on Raspberry Pi.

Used:

- It also provides a set of general purpose input/output pins allowing you to control electronic components for physical computing and explore the Internet of Things (IOT).

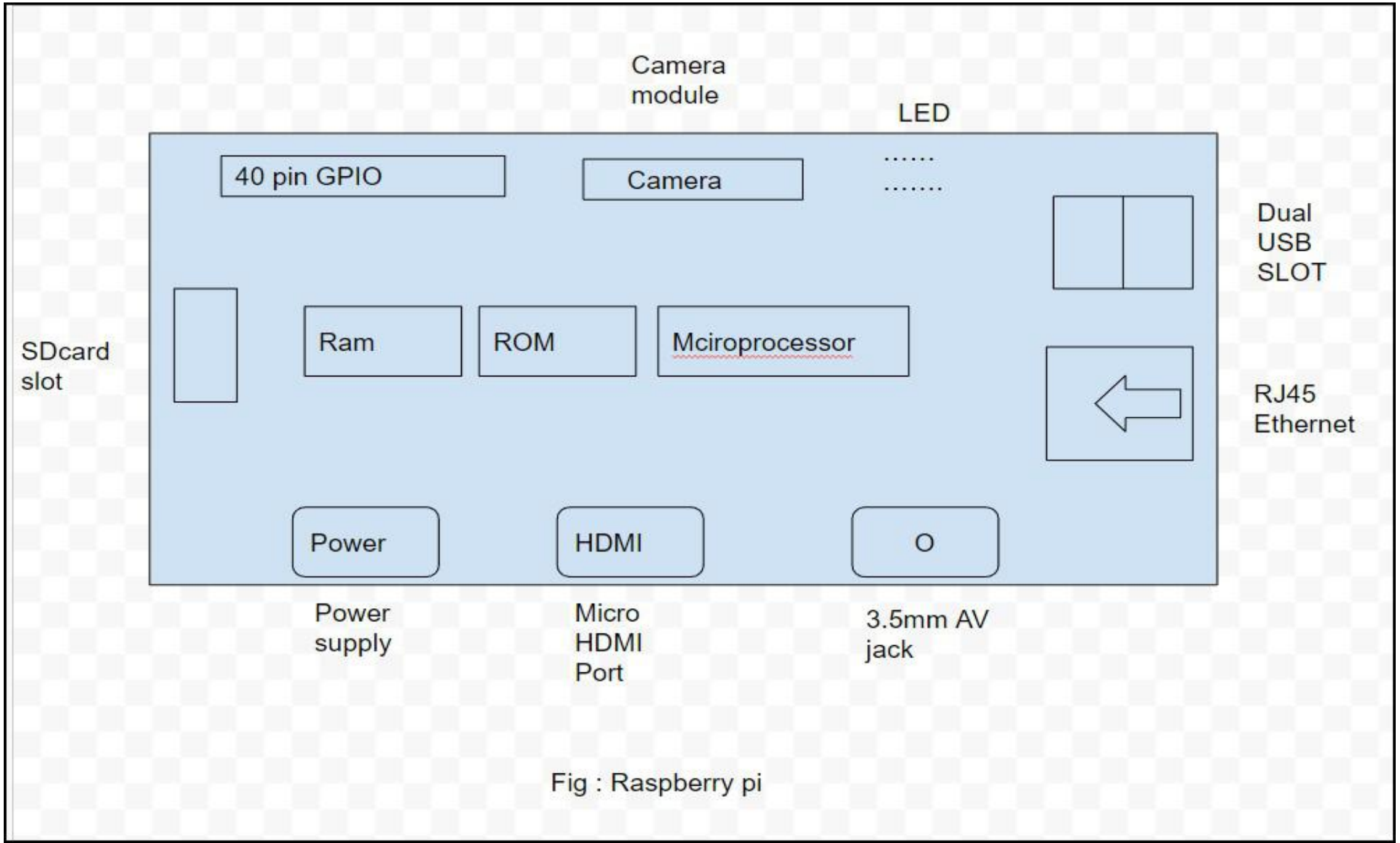
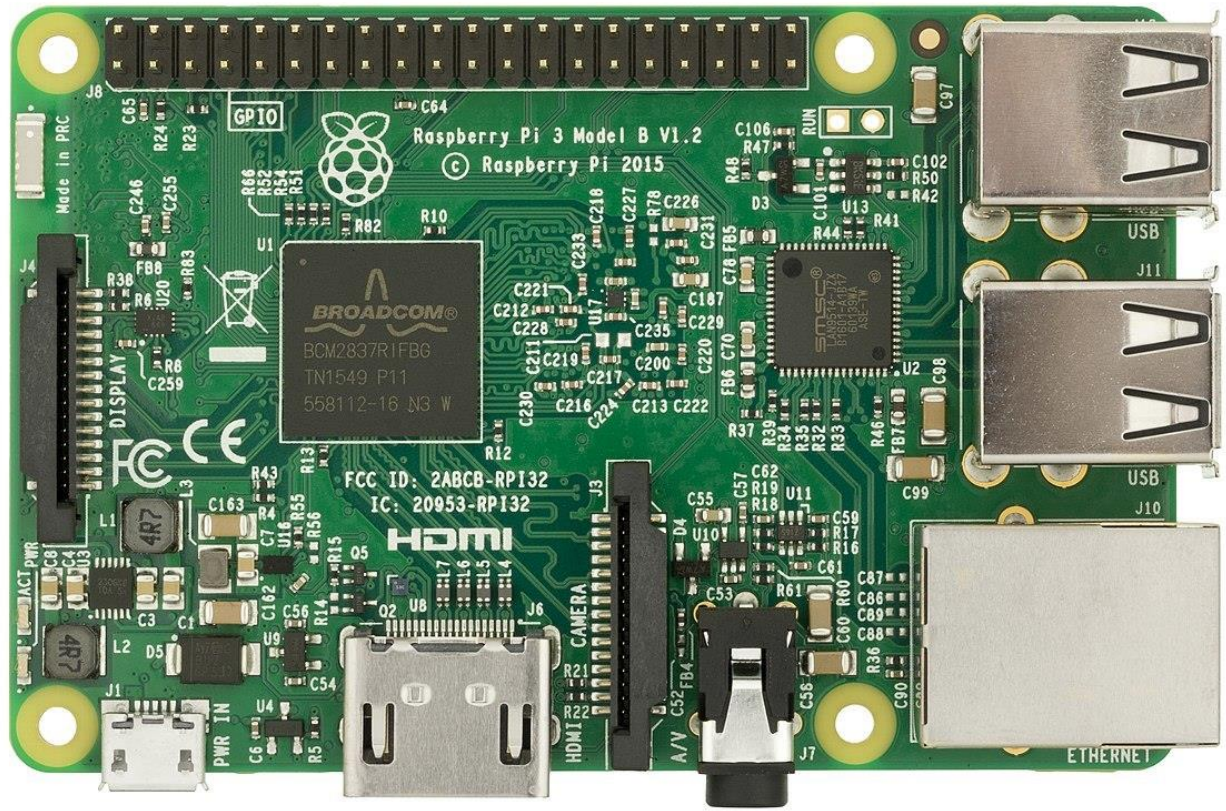


Fig : Raspberry pi

Raspberry Pi model -

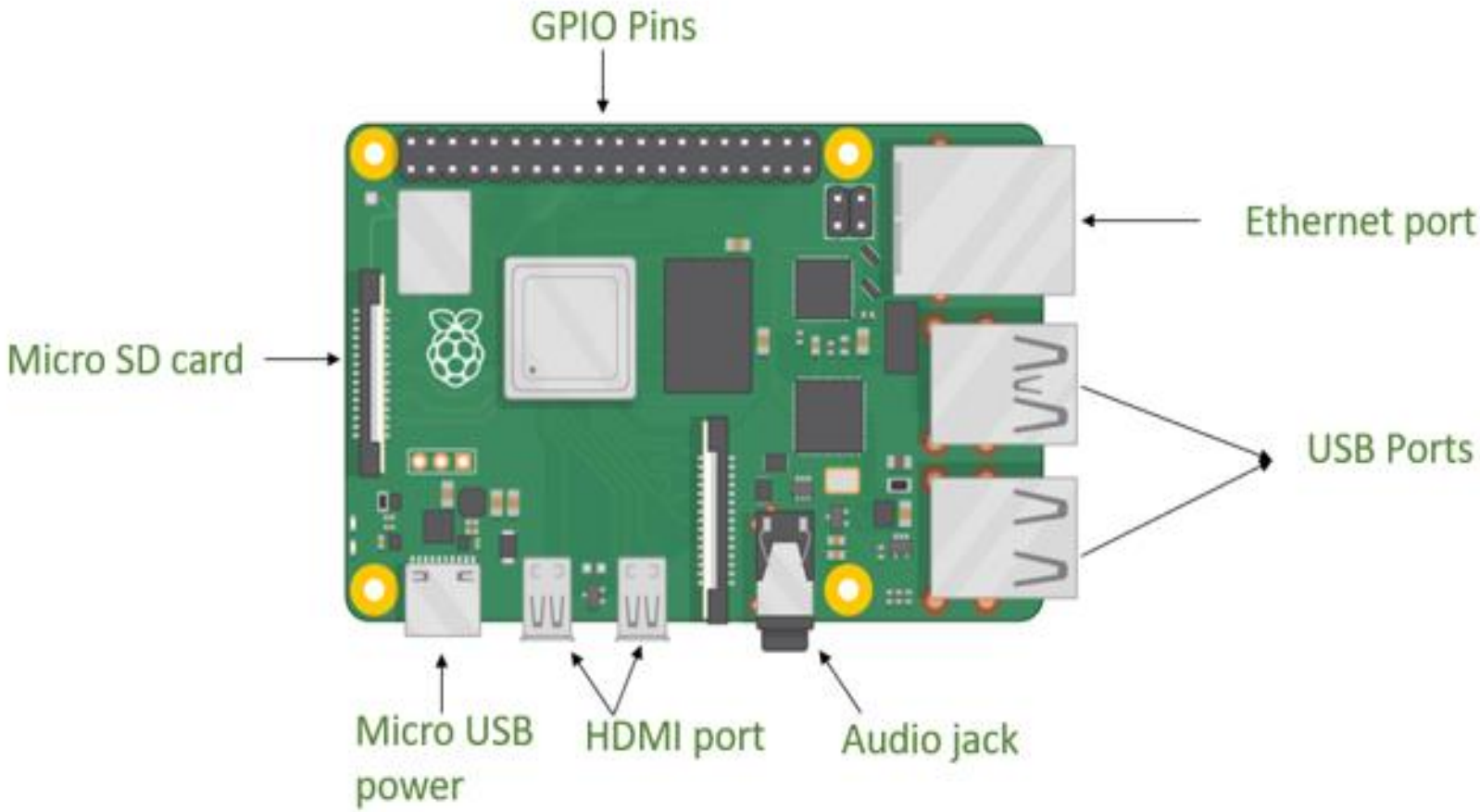
- There have been many generations of raspberry Pi from Pi 1 to Pi 4.
  There is generally a model A and model B.
  Model A is a less expensive variant and it trends to have reduce RAM and dual cores such as USB and Ethernet.
- **List of Raspberry pi models and releases year:**
- pi 1 model B - 2012
- pi 1 model A - 2013
- pi 1 model B+ -2014
- pi 1 model A+ - 2014
- Pi 2 Model B - 2015
- Pi 3 Model B- 2016
- Pi 3 Model B+ -2018
- Pi 3 Model A+ -2019
- Pi 4 Model A - 2019
- Pi Model B - 2020
- Pi 400 - 2021

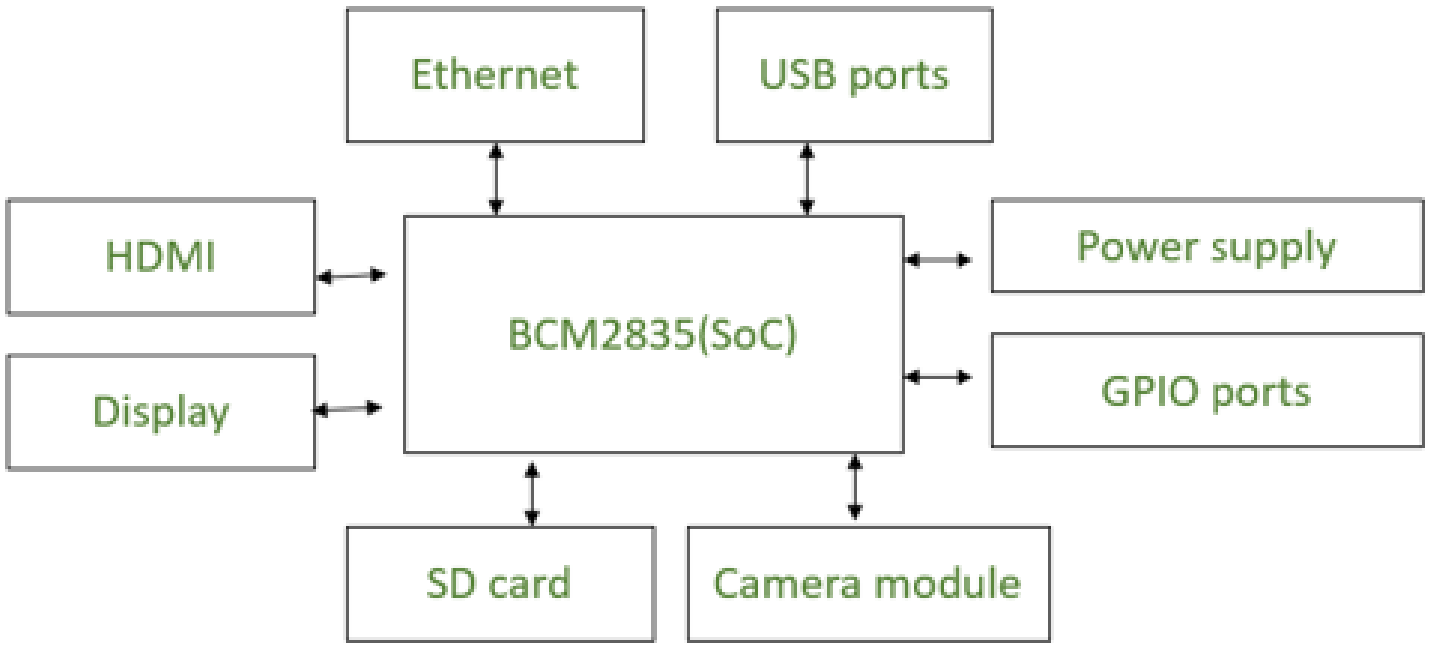

Raspberry Pi 3 Model B V1.2
© Raspberry Pi 2015
BROADCOM
BCM2837RIFBG
TN1549 P11
558112-16 N3 W
FCC ID: 2ABCB-RPI32
IC: 20953-RPI32
GPIO
DISPLAY
CAMERA
HDMI
USB
USB
ETHERNET
PWR IN
A/V
Made in PRC

- **Specs of the Computer: -** The computer has a quad-core ARM processor that doesn't support the same instruction as an X86 desktop CPU.
- It has 1GB of RAM, One HDMI port, four USB ports, one Ethernet connection, Micro SD slot for storage, one combined 3.5mm audio/video port, and a Bluetooth connection.
- It has got a series of input and output pins that are used for making projects like - home security cameras, Encrypted Door lock, etc.

- **Versatility of Raspberry Pi: -** It is indeed a versatile computer and can be utilized by people from all age groups, it can be used for watching videos on YouTube, watching movies, and programming in languages like Python, Scratch, and many more. As mentioned above it has a series of I/O pins that give this board the ability to interact with its environment and hence can be utilized to build really cool and interactive projects.

- **Examples of projects: -** It can be turned into a weather station by connecting some instruments to it for check the temperature, wind speed, humidity etc... It can be turned into a home surveillance system due to its small size; by adding some cameras to it the security network will be ready. If you love reading books it can also become a storage device for storing thousands of eBooks and also you can access them through the internet by using this device.


GPIO Pins
Ethernet port
Micro SD card
USB Ports
Micro USB power
HDMI port
Audio jack


Ethernet
USB ports
HDMI
BCM2835(SoC)
Power supply
Display
GPIO ports
SD card
Camera module

- **Processor:** Raspberry Pi uses Broadcom BCM2835 system on chip which is an ARM processor and Video core Graphics Processing Unit (GPU). It is the heart of the Raspberry Pi which controls the operations of all the connected devices and handles all the required computations.
- **HDMI:** High Definition Multimedia Interface is used for transmitting video or digital audio data to a computer monitor or to digital TV. This HDMI port helps Raspberry Pi to connect its signals to any digital device such as a monitor digital TV or display through an HDMI cable.

- **GPIO ports:** General Purpose Input Output ports are available on Raspberry Pi which allows the user to interface various I/P devices.
- **Audio output**: An audio connector is available for connecting audio output devices such as headphones and speakers.
- **USB ports:** This is a common port available for various peripherals such as a mouse, keyboard, or any other I/P device. With the help of a USB port, the system can be expanded by connecting more peripherals.

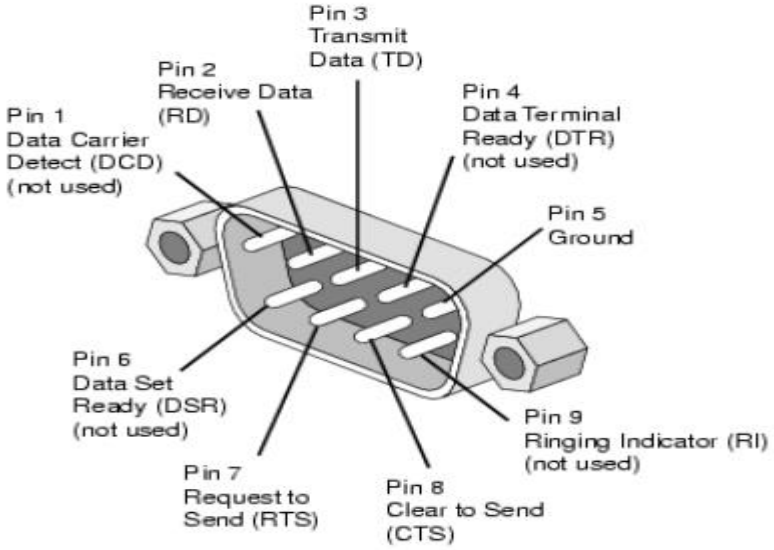
- **SD card: The** SD card slot is available on Raspberry Pi. An SD card with an operating system installed is required for booting the device.
- **Ethernet**: The ethernet connector allows access to the wired network, it is available only on the model B of Raspberry Pi.
- **Power supply:** A micro USB power connector is available onto which a 5V power supply can be connected.
- **Camera module:** Camera Serial Interface (CSI) connects the Broadcom processor to the Pi camera.
- **Display:** Display Serial Interface (DSI) is used for connecting LCD to Raspberry Pi using 15 15-pin ribbon cables. DSI provides a high-resolution display interface that is specifically used for sending video data.

Interface a Raspberry Pi with an Arduino

- A moving target, as technology changes –
- serial (RS-232), USB, I2C, SPI are common.
- Raspberry Pi does these, plus GPIO (Gen. Purp. Input/Output) – GPIB, CAMAC, VME/VXI, PCI cards (DAQ) for lab environ.

Serial Communication

- Most PCs have a DB9 male plug for RS-232 serial asynchronous communications
- In most cases, it is sufficient to use a 2- or 3-wire connect.
- ground (pin 5) and either or both receive and transmit (pins 2 and 3).
- Slow-ish (most common is 9600 bits/sec).



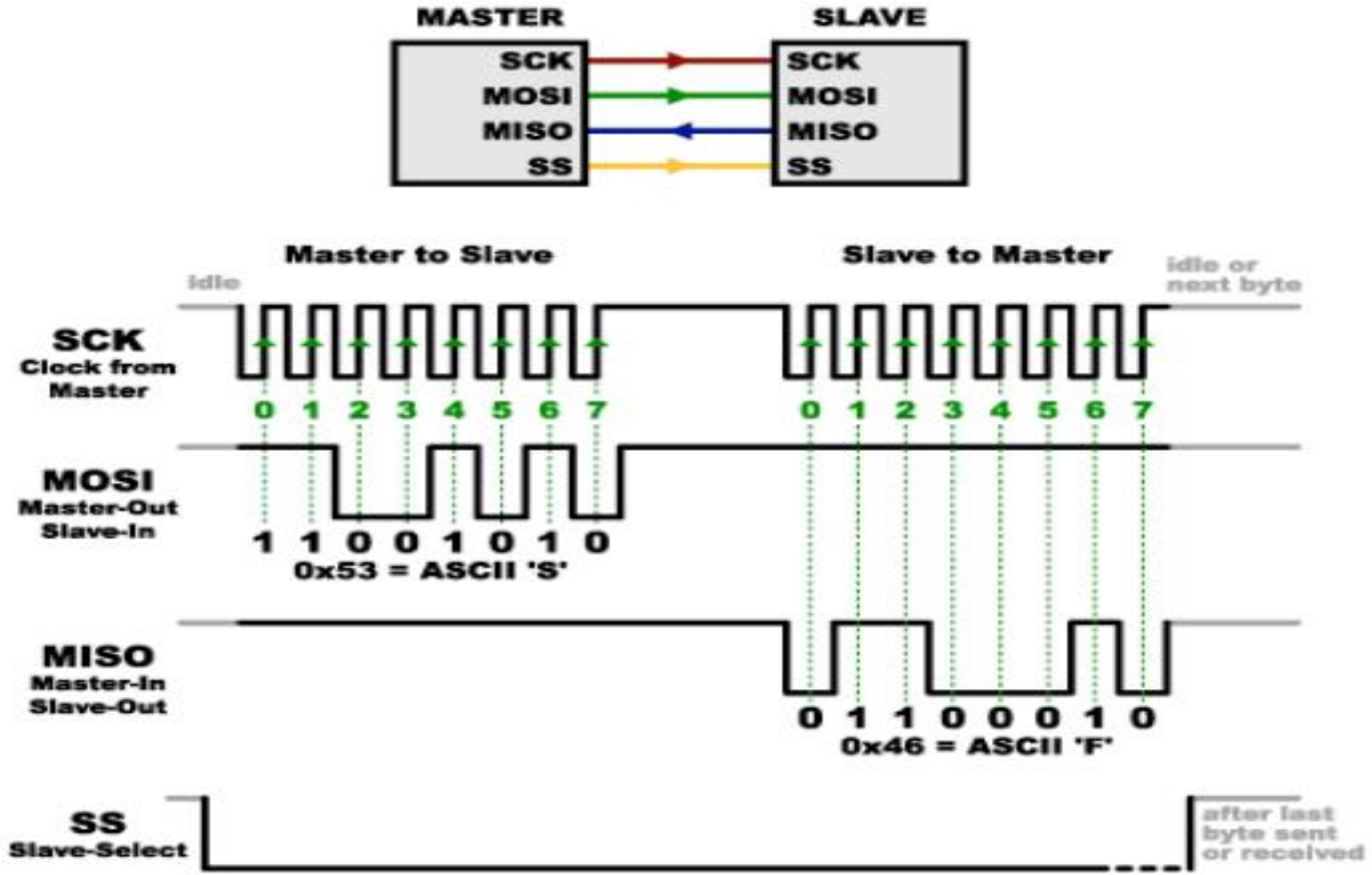
RS-232: most common implementation

- RS-232 is an electrical (physical) specification for communication
  - idle, or "mark" state is logic 1;
    - -5 to −15 V (usually about −12 V) on transmit
    - -3 to −25 V on receive
  - "space" state is logic 0;
    - +5 to +15 V (usually ~12 V) on transmit
    - +3 to +25 V on receive
  - the dead zone is from −3 V to +3 V (indeterminate state)
- Usually used in asynchronous mode, defined by parameters on prev. slide
  - so idles at −12; start jumps to +12; stop bit at −12
  - since each packet is framed by start/stop bits, guaranteed a transition at start
  - parity (if used) works as follows:
    - even parity guarantees an even number of ones in the train
    - odd parity guarantees an odd number of ones in the train
- UART: Universal Asynchronous Receiver/Transmitter
  - common term/label for a serial interface

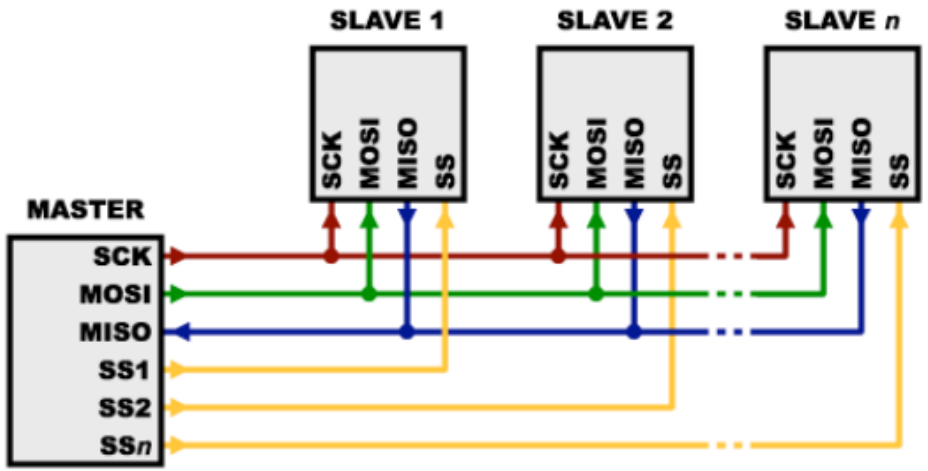
GPIB (IEEE-488)

- An 8-bit parallel bus allowing up to 15 devices connected to the same computer port
  - addressing of each machine (either via menu or dip-switches) determines who's who
  - can daisy-chain connectors, each cable 2 m or less in length
- Extensive handshaking controls the bus
  - computer controls who can talk and who can listen
- Many test-and-measurement devices equipped with GPIB
  - common means of controlling an experiment: positioning detectors, measuring or setting voltages/currents, etc.
- Can be reasonably fast (1 Mbit/sec)

SPI Scheme


MASTER
SLAVE
SCK
MOSI
MISO
SS
SCK
MOSI
MISO
SS
Master to Slave
Slave to Master
idle
idle or
next byte
SCK
Clock from
Master
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
MOSI
Master-Out
Slave-In
1 1 0 0 1 0 1 0
0x53 = ASCII 'S'
MISO
Master-In
Slave-Out
0 1 1 0 0 0 1 0
0x46 = ASCII 'F'
SS
Slave-Select
after last
byte sent
or received

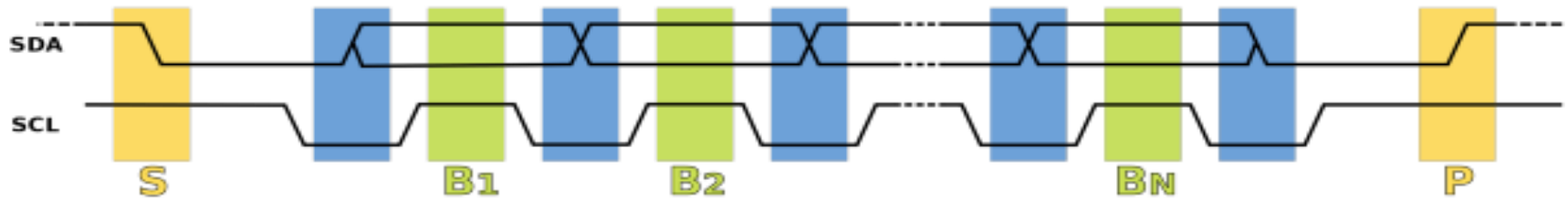
Multiple Devices



from sparkfun.com

I²C: Inter-Integrated Circuit

- Pronounced I-squared-C or I-two-C
- Two signal lines (plus ground):
  - clock (SCL)
  - data (SDA; bi-directional)



  - Starts when SDA pulled low while SCL still high
  - stoPs when SDA pulled high while SCL restored to high
  - data read/valid while SCL high (updated when SCL low)
  - data line can contain read/write and acknowledge bits

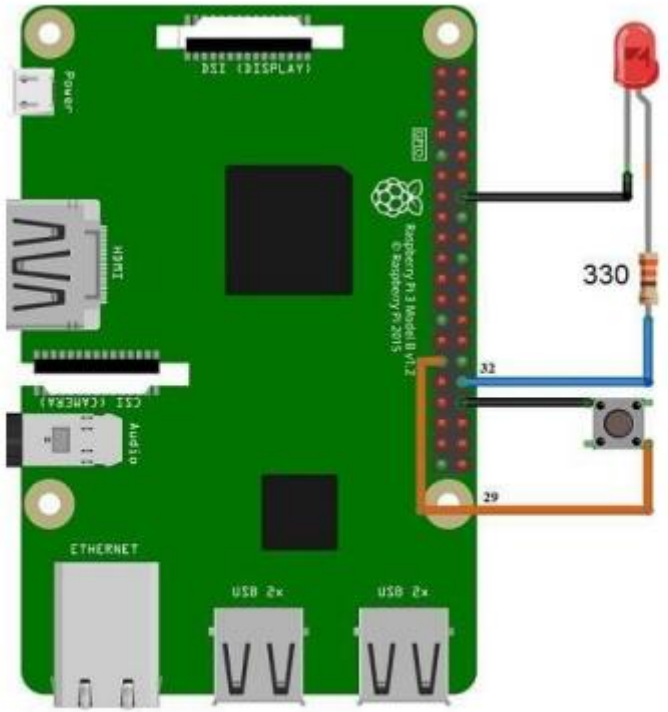
Raspberry Pi GPIO Access

- GPIO (General Purpose Input Output) pins can be used as input or output and allows raspberry pito connect with general purpose I/O devices.
- Raspberry pi 3 model B took out 26 GPIO pins on board.
- Raspberry pi can control many external I/O devices using these GPIO’s.
- These pins are a physical interface between the Pi and the outside world.

- We can program these pins according to our needs to interact with external devices.
- For example, if we want to read the state of a physical switch, we can configure any of the available GPIO pins as input and read the switch status to make decisions.
- We can also configure any GPIO pin as an output to control LED ON/OFF.
- Raspberry Pi can connect to the Internet using on-board Wi-Fi or Wi-Fi USB adapter.
- Once the Raspberry Pi is connected to the Internet then we can control devices, which are connected to the Raspberry Pi, remotely

3.3V PWR	1			2	5V PWR
I2C1 SDA	3			4	5V PWR
I2C1 SCL	5			6	GND
GPIO 4	7			8	UART0 TX
GND	9			10	UART0 RX
GPIO 17	11			12	GPIO 18
GPIO 27	13			14	GND
GPIO 22	15			16	GPIO 23
3.3V PWR	17			18	GPIO 24
SPI0 MOSI	19			20	GND
SPI0 MISO	21			22	GPIO 25
SPI0 SCLK	23			24	SPI0 CS0
GND	25			26	SPI0 CS1
Reserved	27			28	Reserved
GPIO 5	29			30	GND
GPIO 6	31			32	GPIO 12
GPIO 13	33			34	GND
GPIO 19	35			36	GPIO 16
GPIO 26	37			38	GPIO 20
GND	39			40	GPIO 21

Control LED with Push Button using Raspberry Pi



Control LED using Raspberry Pi Interfacing Diagram