Core Module-4

(60 hours)

Python Programming

(60 hours)

In this section, we will discuss:

- Introduction to Python.
- History.
- Features
- Setting Up Path
- Basic Syntax Variable
- Data Types Operator
- Conditional Statement
- Looping
- Control Statement

In this session, we will discuss:

- String Manipulation
- Lists
- Tuple
- Functions and Methods
- Dictionaries
- Functions
- Modules
- Input and Output
- Exception Handling
- Object Oriented Programming

Introduction to Python

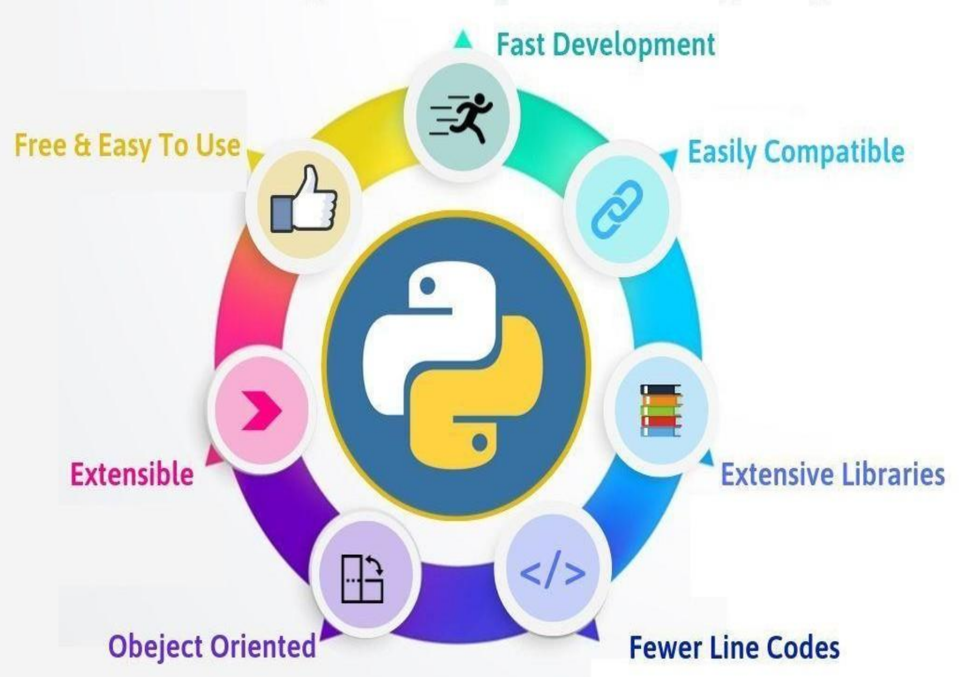


Image source: https://images.app.goo.gl/hHjqBxYkqezSBY41A

Introduction to Python

Characteristics of Python

- Simple Syntax
- GUI Programming.
- Scalable
- Free and Open Source
- Variety of Usage and Application.
- Interpreted and Interactive
- Object Oriented

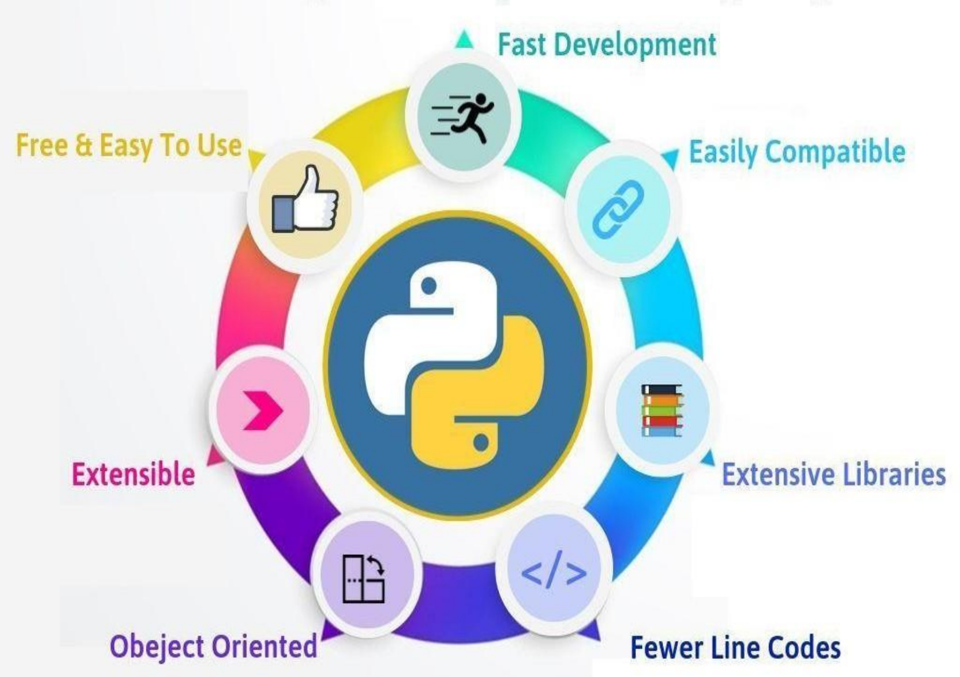
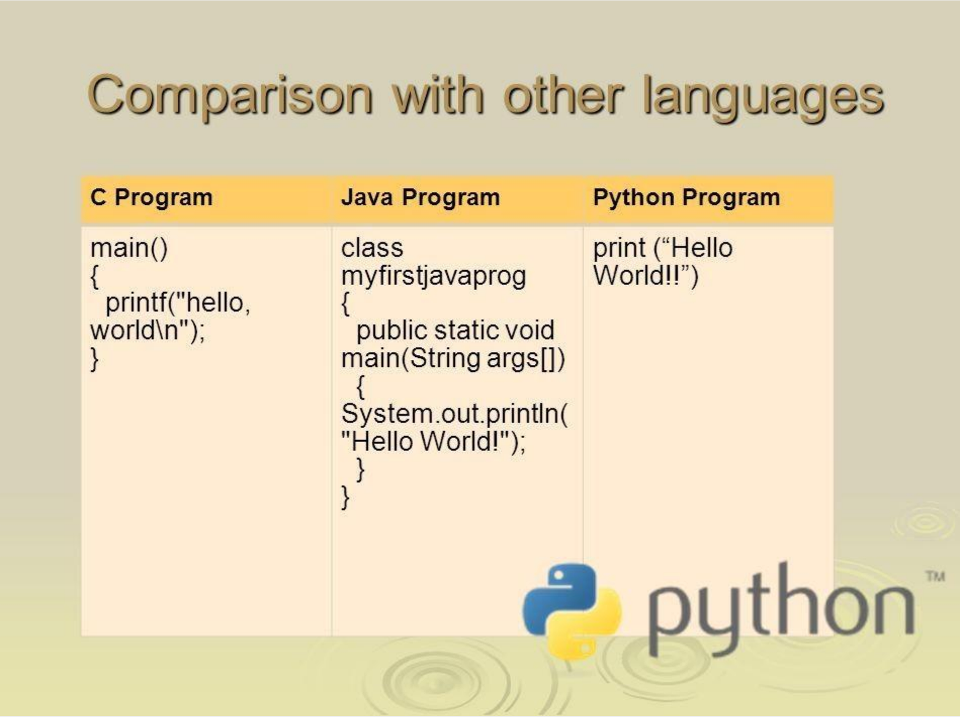


Image source: https://images.app.goo.gl/hHjqBxYkqezSBY41A

Introduction to Python

Advantages over other languages

- Simple code
- It is easy to understand
- It is Free
- It Needs Less Coding
- All Kinds of Businesses Can Afford it
- It is one of the most Trending Language.



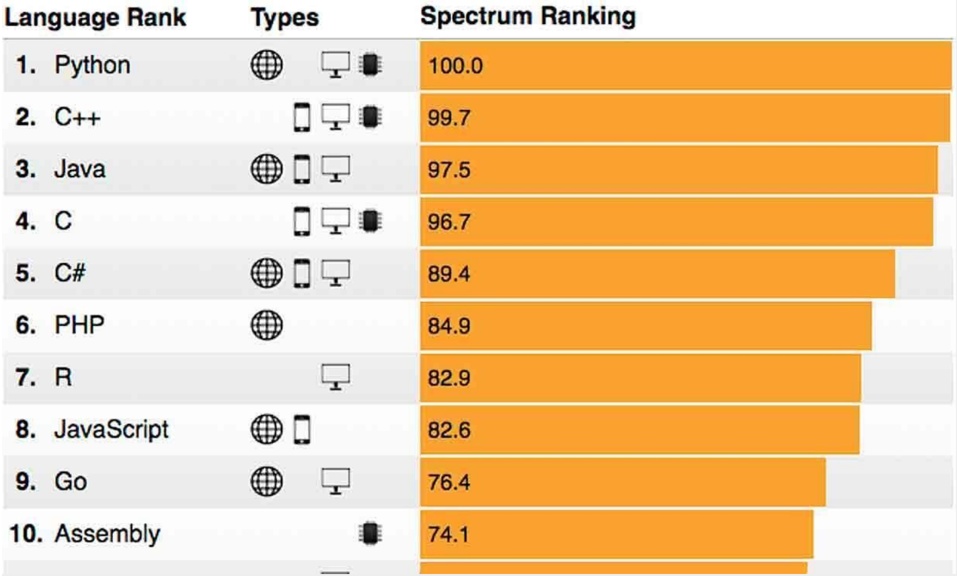
Comparison with other languages

C Program	Java Program	Python Program

main()
{
printf("hello, world\n");
}
	class myfirstjavaprog
	{
	public static void main(String args[])
	{
	System.out.println("Hello World!");
	}
	}
		print (“Hello World!!”)

Image Source: https://images.app.goo.gl/2K2vApdtZx7vBbVn7

History



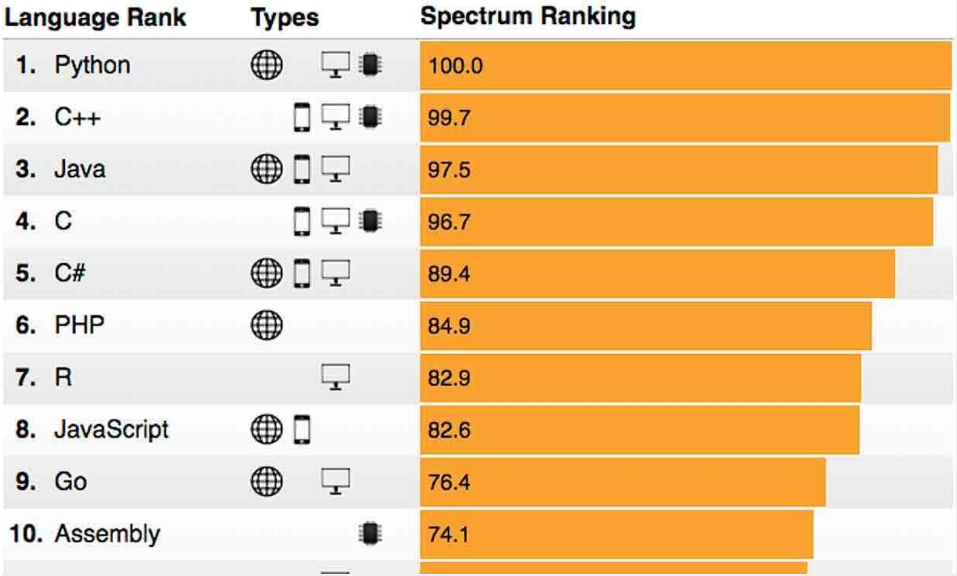
Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▪	100.0
2. C++	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▪	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▪	74.1

Image Source : https://images.app.goo.gl/VtzTMjduwUVfx1r57

History

Python Timeline/History and IEEE rankings

- Python was conceptualized by Guido Van Rossum in the late 1980s.
- Rossum Published the first version of Python code(0.9.0) in February 1991 at CWI(Centrum Wiskunde & Informatica) in Netherland, Amsterdam.
- Python is Derived from ABC Programming Language that had been developed at the CWI.



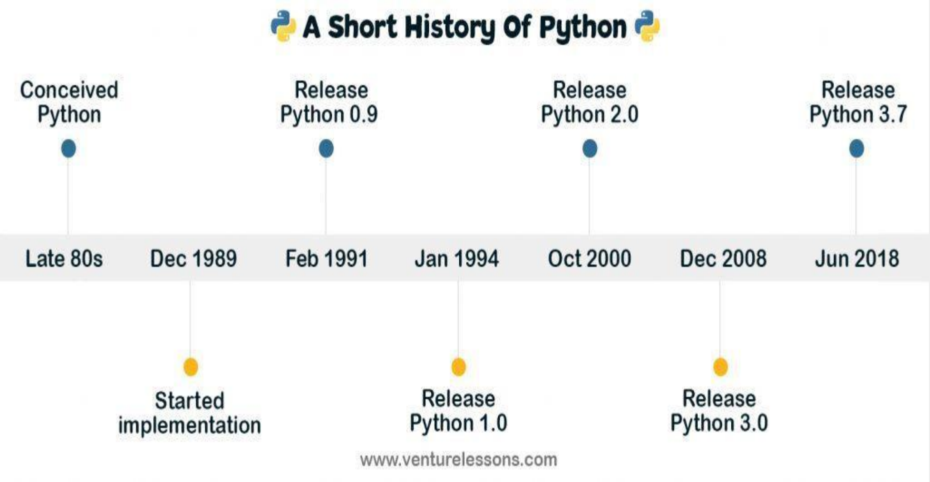
Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥 ▣	100.0
2. C++	📱 🖥 ▣	99.7
3. Java	🌐 📱 🖥	97.5
4. C	📱 🖥 ▣	96.7
5. C#	🌐 📱 🖥	89.4
6. PHP	🌐	84.9
7. R	🖥	82.9
8. JavaScript	🌐 📱	82.6
9. Go	🌐 🖥	76.4
10. Assembly	▣	74.1

Image Source : https://images.app.goo.gl/VtzTMjduwUVfx1r57

History

Python Timeline/History and IEEE rankings (Contd..)

- Rossum choose the name “Python” , since he was a big fan of Monty Python’s Flying Circus.
- Python is now maintained by a core development team at the institute, although Rossum still holds a vital role in directing its progress.



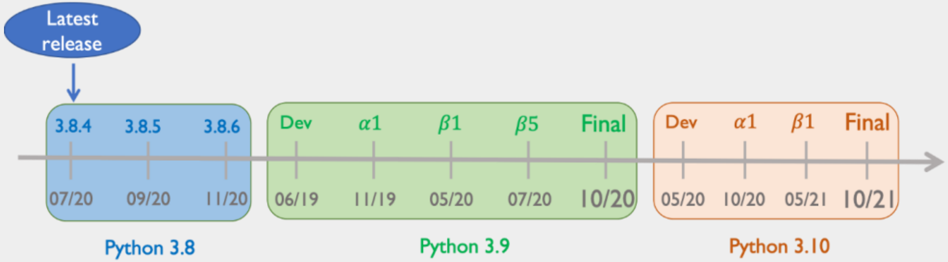
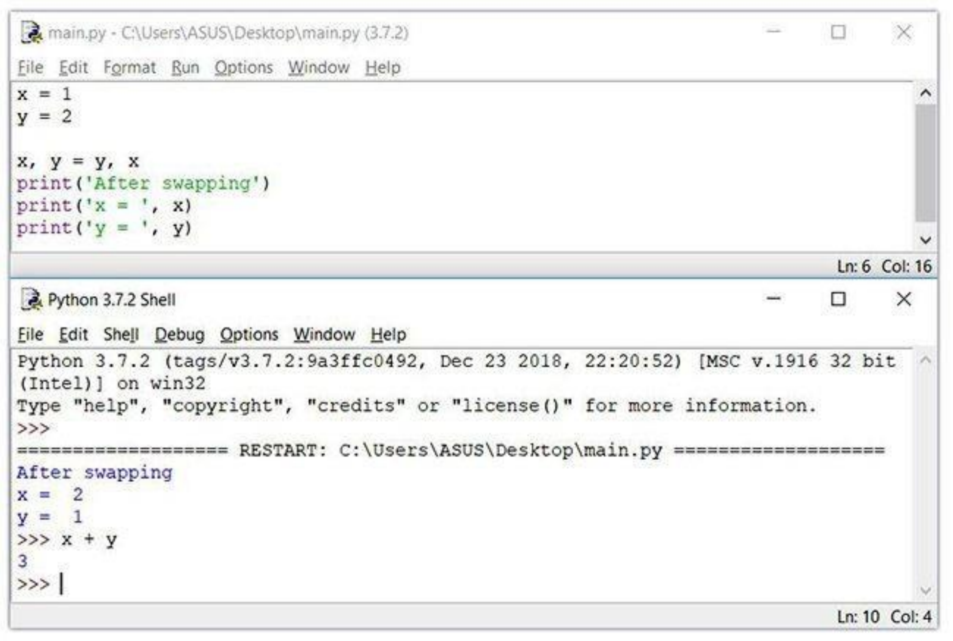


Image Source: https://images.app.goo.gl/61fVvm834STCtJW89

Features



main.py - C:\Users\ASUS\Desktop\main.py (3.7.2)
File Edit Format Run Options Window Help
Ln: 6 Col: 16

```
x = 1
y = 2

x, y = y, x
print('After swapping')
print('x = ', x)
print('y = ', y)
```

Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Ln: 10 Col: 4

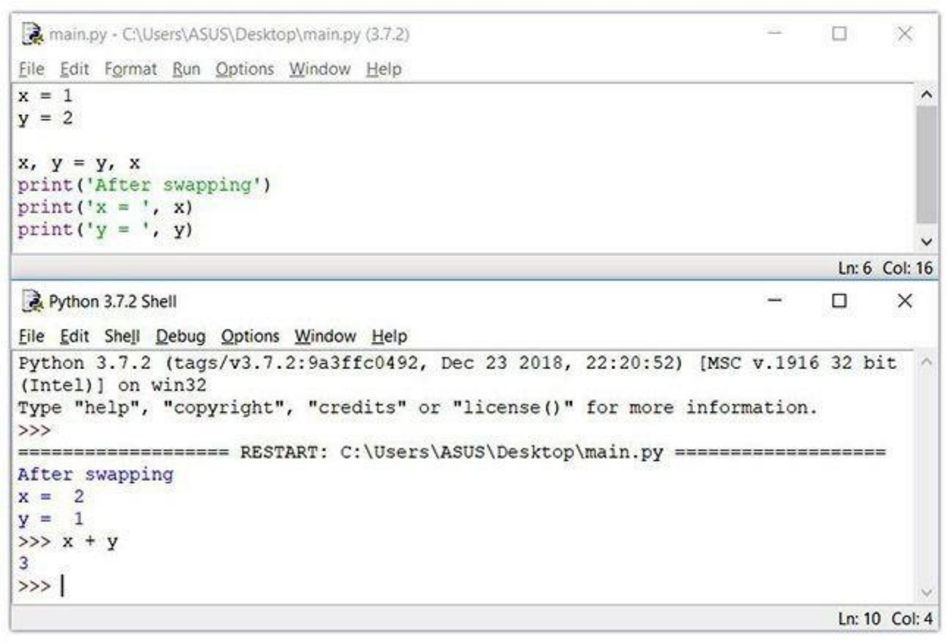
```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\ASUS\Desktop\main.py ===================
After swapping
x =  2
y =  1
>>> x + y
3
>>> |
```

Image source :
https://cdn.programiz.com/sites/tutorial2program/files/python-idle.JPG

Features

Easy to Code and Understand

- Python has simpler syntax when compared to C, C++, Java and other programming languages.
- This enables any newbie to quickly pick up the basics of Python.
- Also, despite being a high-level language, Python code looks very short much readable due to its English like commands. In short, it is a developer-friendly language.


main.py - C:\Users\ASUS\Desktop\main.py (3.7.2)
File Edit Format Run Options Window Help
Ln: 6 Col: 16
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
Ln: 10 Col: 4

```
x = 1
y = 2

x, y = y, x
print('After swapping')
print('x = ', x)
print('y = ', y)
```

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: C:\Users\ASUS\Desktop\main.py ===================
After swapping
x =  2
y =  1
>>> x + y
3
>>> |
```

Image source :
https://cdn.programiz.com/sites/tutorial2program/files/python-idle.JPG

edunet foundation

Features

Expressive Language

- Python is very expressive when compared to other languages.
- By expressive, we mean, in Python a single line of code performs a lot more than what multiple lines can perform in other languages.
- In simple it means that fewer lines of code are required to write a program in Python.

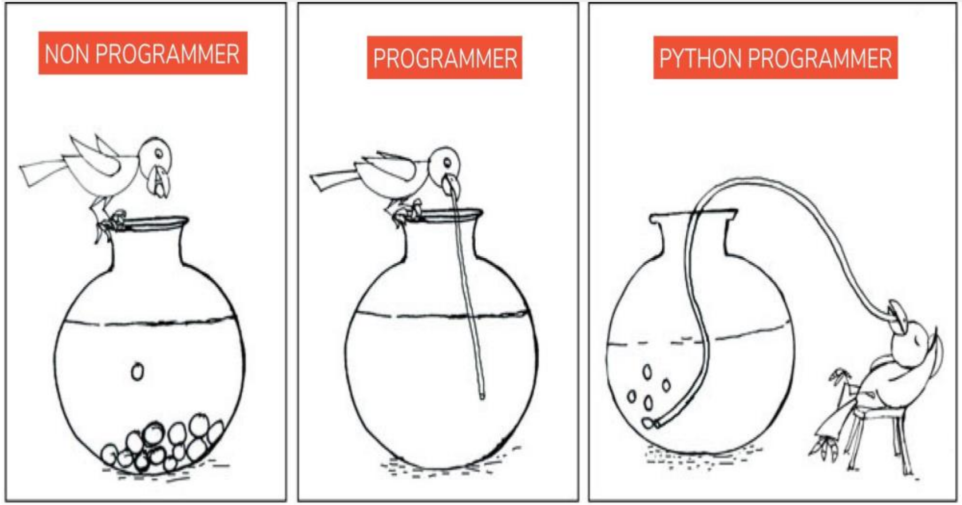


Image Source: https://images.app.goo.gl/Ftdt9TC6dtaVkXHg6

Features

Object Oriented

- Python is a multi-paradigm programming language. Meaning, it supports different programming approach.
- One of the popular approach to solve a programming problem is by creating objects.
- This is known as Object-Oriented
- Programming (OOP).

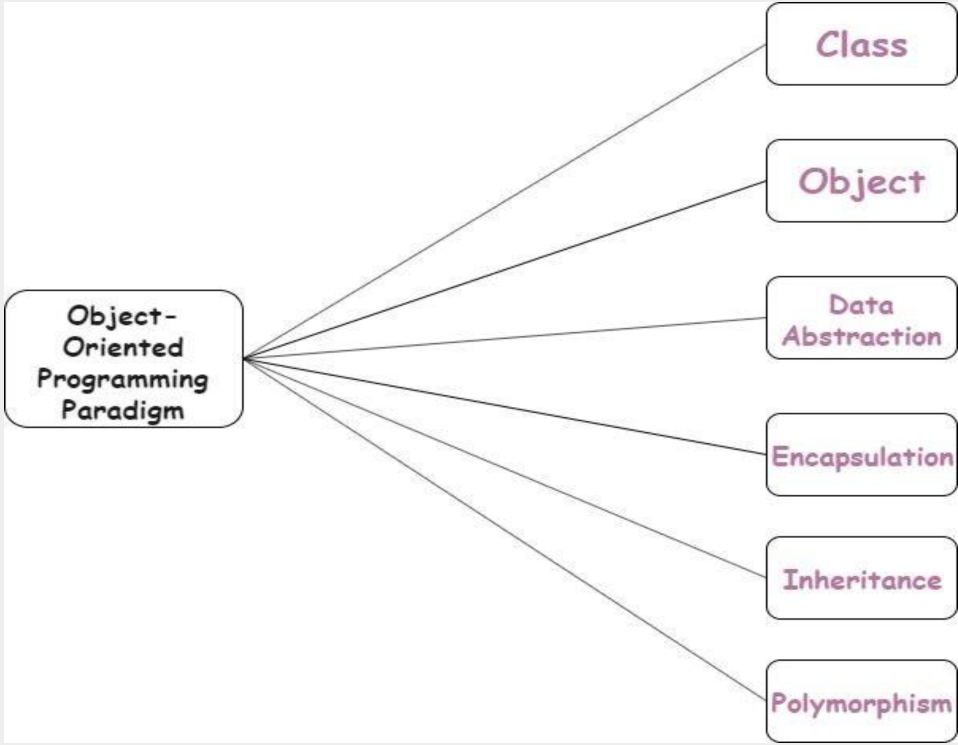


Image Source: https://images.app.goo.gl/iP8uaejcdLcDbJCH6

Features

Object Oriented (Contd..)

- Python is an “object-oriented programming language.”
- This means that almost all the code is implemented using a special construct called classes.

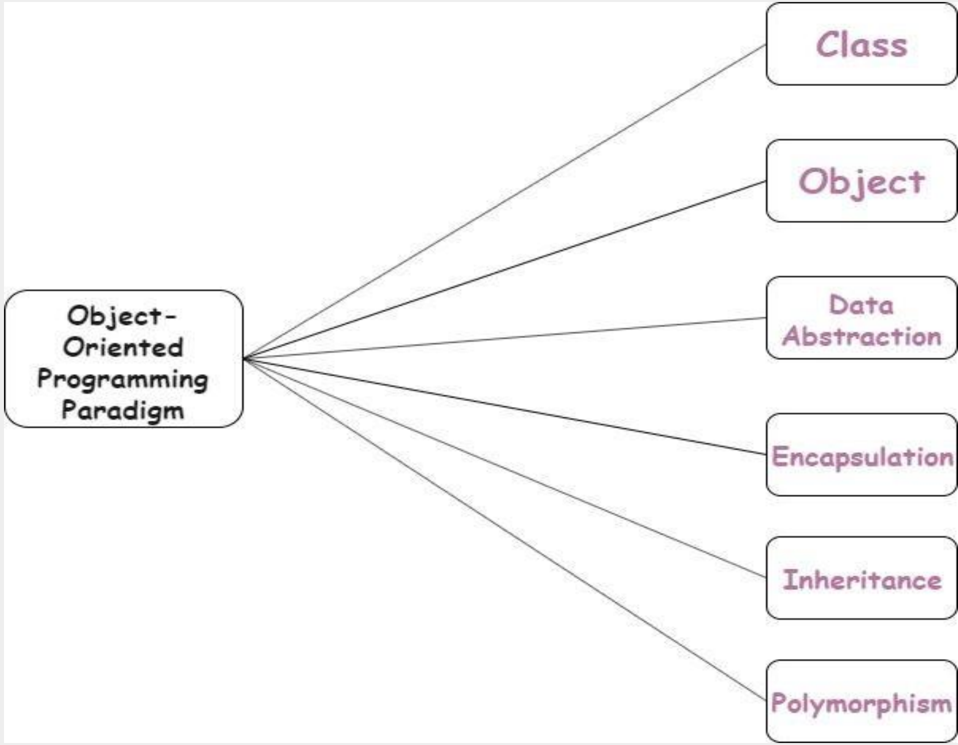


Image Source: https://images.app.goo.gl/iP8uaejcdLcDbJCH6

Features

Extensible Language

- In case you want to write a part of your Python code in C++ or Java etc, then you can do it.
- Since Python is an extensible language, it lets you do this with ease.

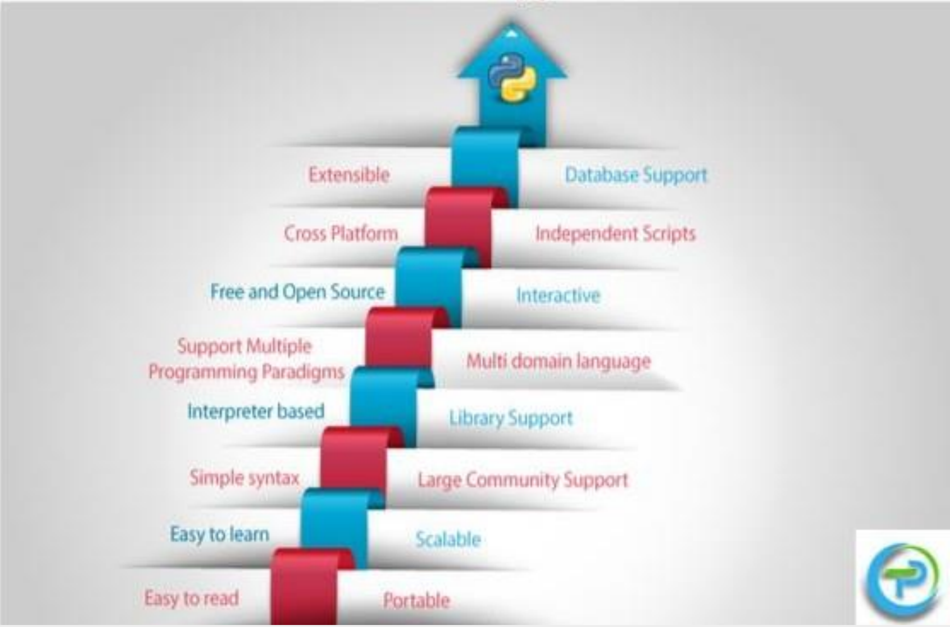
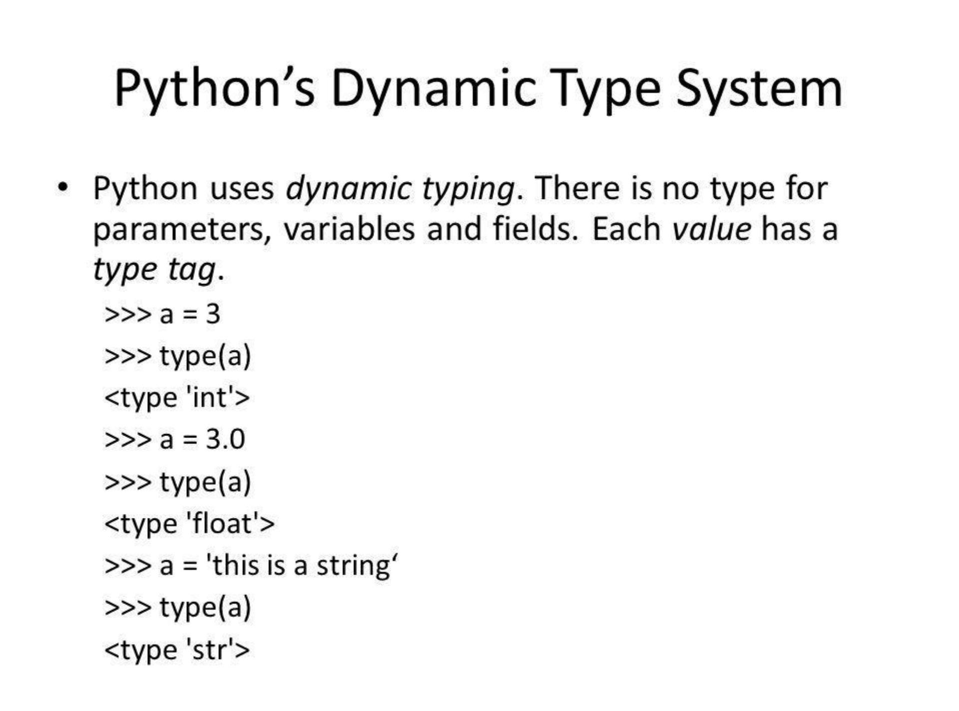


Image Source: https://techvidvan.com/tutorials/features-of-python/

Features

Dynamically typed Programming Language

- Python is a dynamically typed language.
- This means, whenever a variable is declared, the programmer need not mention its data type.
- Rather, the type of the variable is decided during run time.



Python’s Dynamic Type System

- Python uses *dynamic typing*. There is no type for parameters, variables and fields. Each *value* has a *type tag*.

```
>>> a = 3
>>> type(a)
<type 'int'>
>>> a = 3.0
>>> type(a)
<type 'float'>
>>> a = 'this is a string‘
>>> type(a)
<type 'str'>
```

Image Source: https://techvidvan.com/tutorials/features-of-python/

Features

Use of Interpreter

- Python installation interprets and executes the code line by line at a time.
- Python interpreter offers some pretty cool features:
- Interactive editing
- History substitution
- Code completion on systems with support for readline

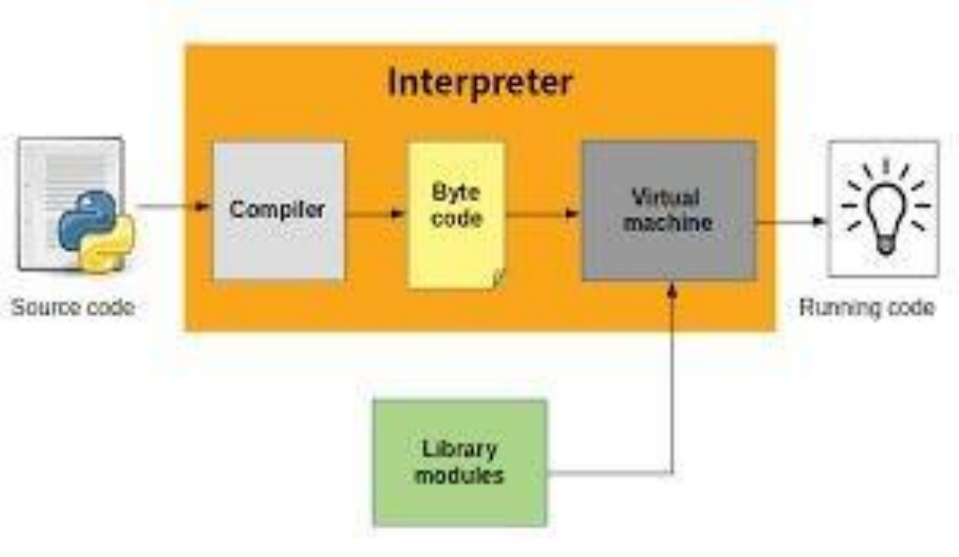


Image Source: https://images.app.goo.gl/zFfSFFXMyPCLGPZcA

Features

Free & Open Source

- Python language is freely available
- i.e without any cost.
- It is open and available to anyone.
- Anyone can freely distribute it,read the source code and edit it.
- Pythons license is administered by the Python Software Foundation.

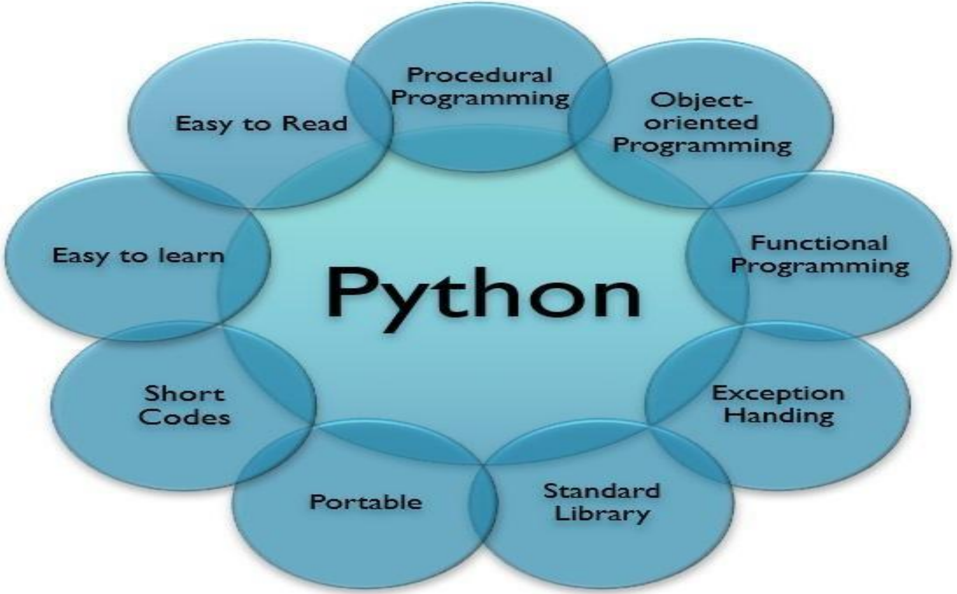


Image Source: https://images.app.goo.gl/WLD1cUM2GXMk9E5Z6

Features

Cross Platform Language

- Python can run equally well on variety of platform-
- Windows
- Linux/Unix
- Macintosh
- Smart Phones etc
- We can also say that Python is a portable language.

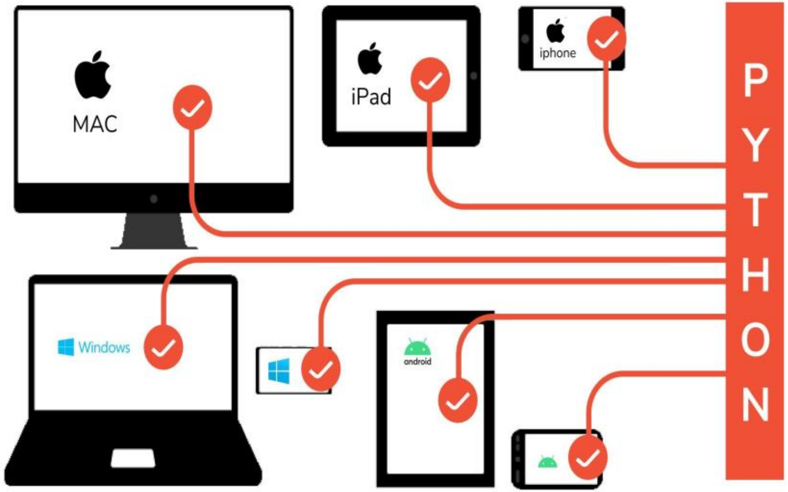
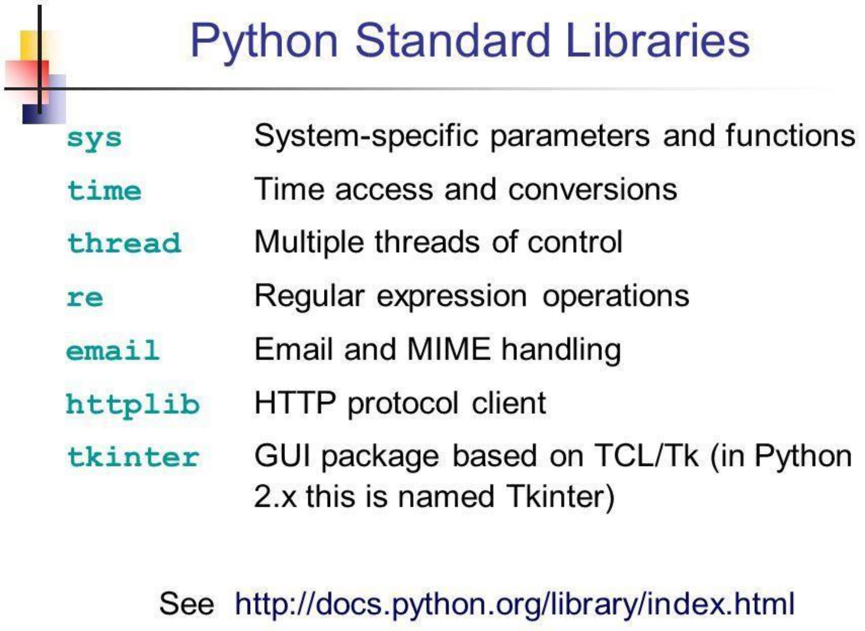


Image Source: https://images.app.goo.gl/WLD1cUM2GXMk9E5Z6

Features

Large Standard Library

- Python has a large standard library and this helps save the programmers time as you don't have to write your own code for every single logic.
- There are libraries for expressions, unit-testing, web browsers, databases, CGI, image manipulation etc.



Python Standard Libraries

`sys`	System-specific parameters and functions
`time`	Time access and conversions
`thread`	Multiple threads of control
`re`	Regular expression operations
`email`	Email and MIME handling
`httplib`	HTTP protocol client
`tkinter`	GUI package based on TCL/Tk (in Python 2.x this is named Tkinter)

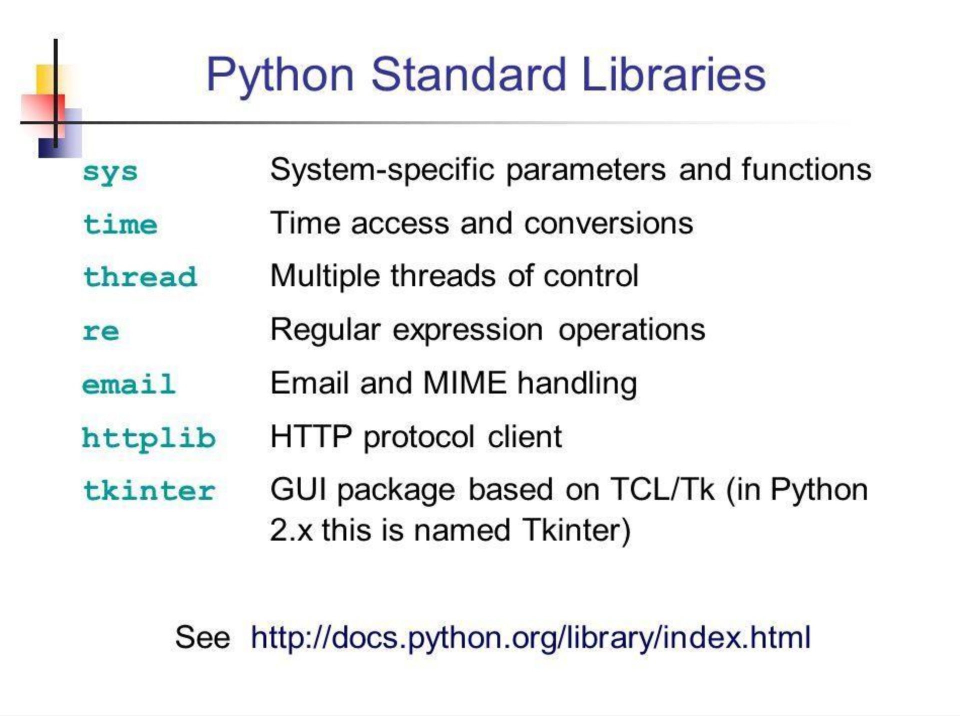
See http://docs.python.org/library/index.html

Image Source: https://docs.python.org/3/library/index.html

Features

Large Standard Library (Contd..)

- Python provide rich set of module and functions for rapid application development
- Python’s bulk of the library is very portable and cross-platform compatible on UNIX, Windows and Macintosh.



Python Standard Libraries

`sys`	System-specific parameters and functions
`time`	Time access and conversions
`thread`	Multiple threads of control
`re`	Regular expression operations
`email`	Email and MIME handling
`httplib`	HTTP protocol client
`tkinter`	GUI package based on TCL/Tk (in Python 2.x this is named Tkinter)

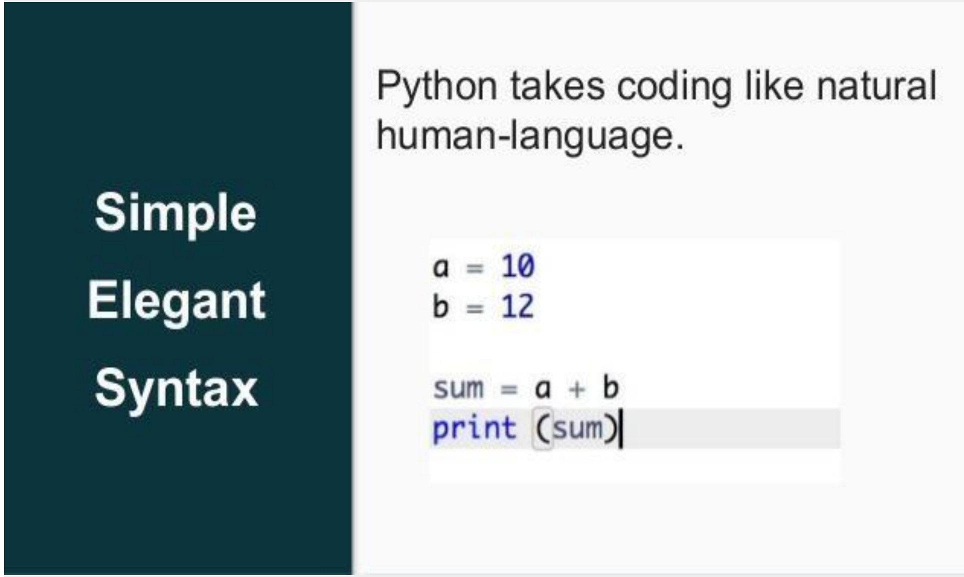
See http://docs.python.org/library/index.html

Image Source: https://docs.python.org/3/library/index.html

Features

Elegant Syntax

- Python Elegant Syntax means it is more capable to expressing the code's purpose than many other languages.
- Python can easily test even small portion of code.
- Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.



Simple Elegant Syntax

Python takes coding like natural human-language.

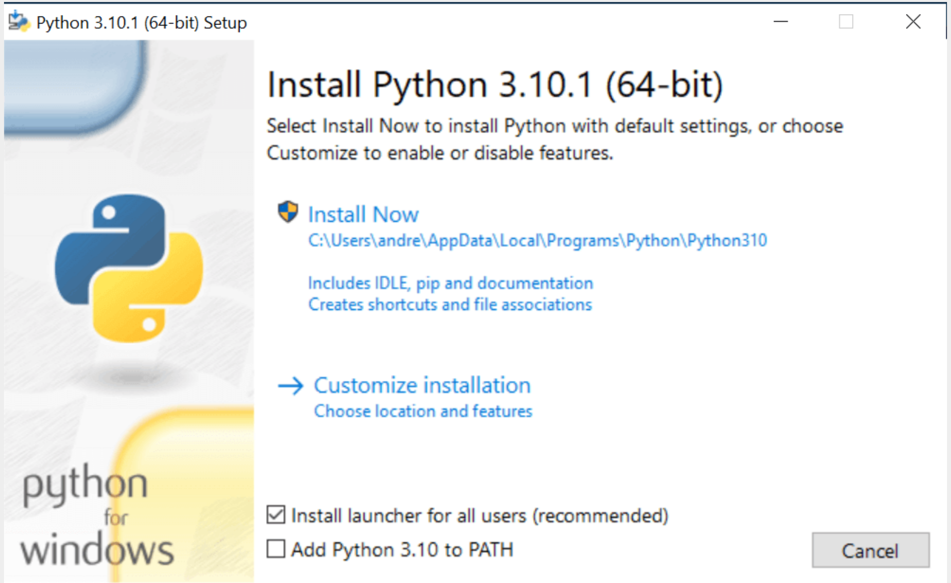
```
a = 10
b = 12

sum = a + b
print (sum)
```

Image Source: https://images.app.goo.gl/ZPuTEDwaKeEAcv8h6

Setting Up Path

Installation of Python IDLE

- List of Hardware/Software Requirements:
- Laptop/Computer with Windows/Linux OS-Ubuntu 18.04LTS
- Python Software
- Installation Steps:
- Install Python software in the system.
- Open the browser and type the python.org/downloads.



Python 3.10.1 (64-bit) Setup

Install Python 3.10.1 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\andre\AppData\Local\Programs\Python\Python310

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

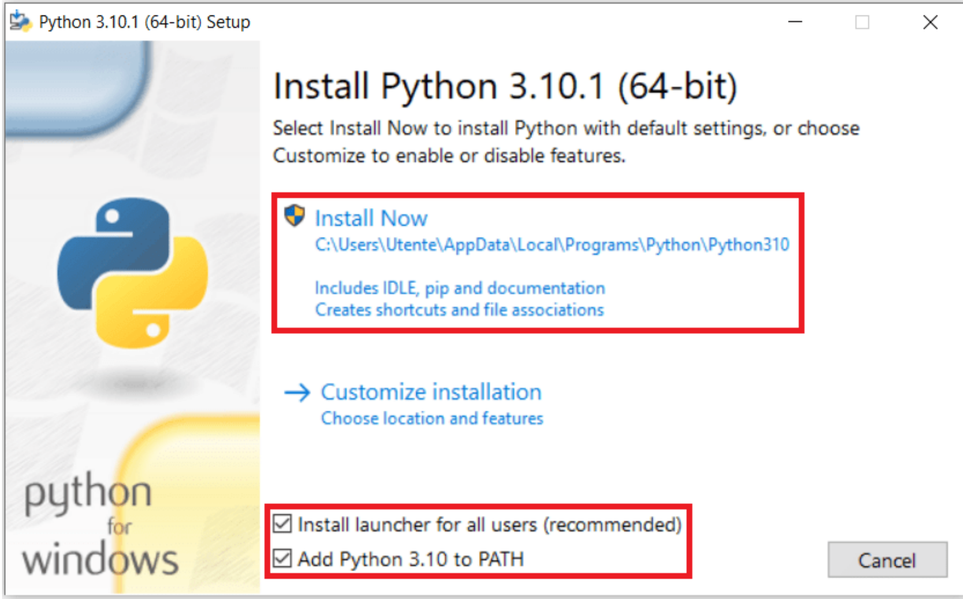
☑ Install launcher for all users (recommended)
□ Add Python 3.10 to PATH

Image Source: https://www.geeksforgeeks.org/how-to-install-python-on-windows/

Setting Up Path

Installation of Python in Windows(Contd..)

- Choice either Windows x86-64 executable installer for 64-bit or Windows x86 executable installer for 32-bit.
- After downloading a file this page will appear.
- Install the software
- Select Add Python 3.10 to Path



Python 3.10.1 (64-bit) Setup

Install Python 3.10.1 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

Install Now
C:\Users\Utente\AppData\Local\Programs\Python\Python310

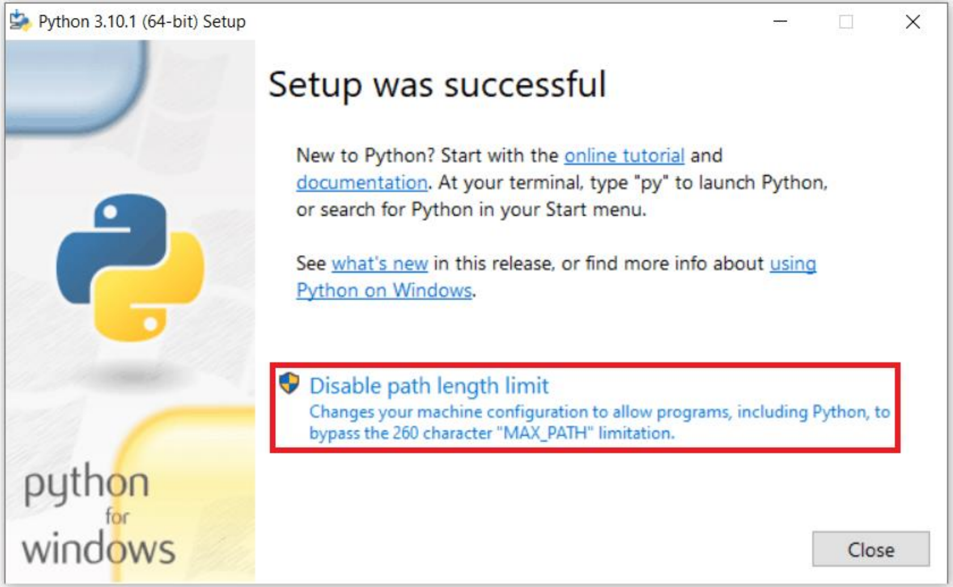
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.10 to PATH

Image Source: https://www.geeksforgeeks.org/how-to-install-python-on-windows/

Setup Successful



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

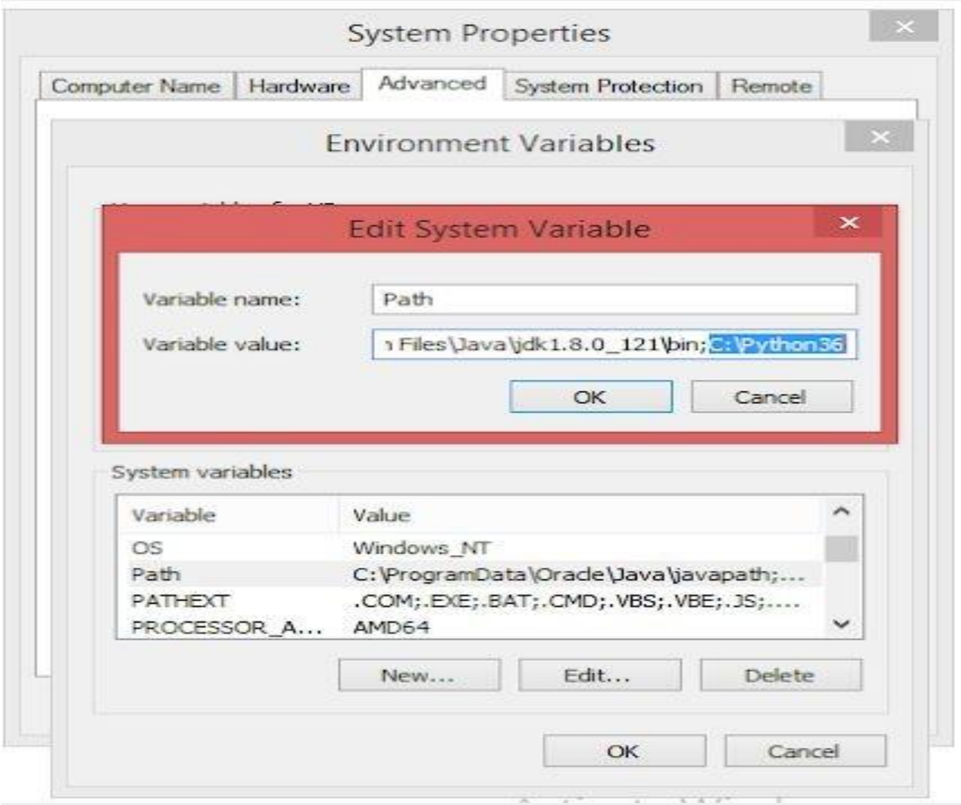
Image Source: https://www.geeksforgeeks.org/how-to-install-python-on-windows/

Setting Up Path

How to set Python Path in Windows

- To permanently modify the default
- environment variables :

- My Computer > Properties > Advanced System Settings > Environment Variables > Edit



Variable name: Path

Variable value: ı Files\Java\jdk1.8.0_121\bin;C:\Python36

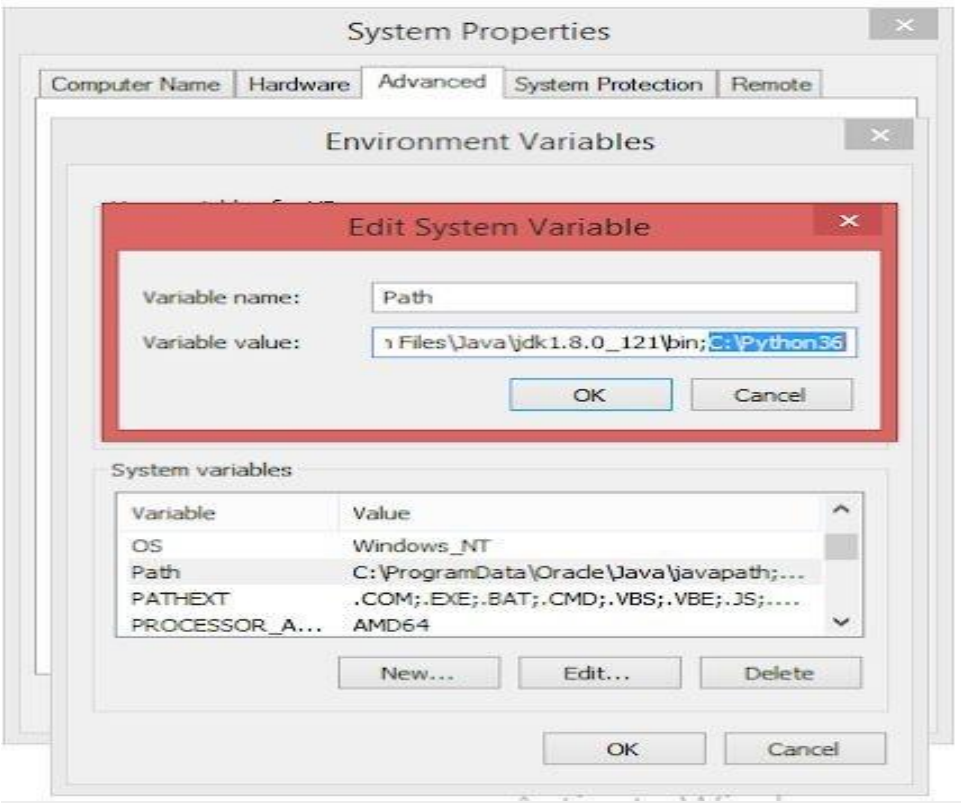
Variable	Value
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Image Source: https://net-informations.com/python/intro/path.htm#google_vignette

Setting Up Path

How to set Python Path in Windows(Contd..)

- Right-click 'My Computer'.
- Select 'Properties' at the bottom of the Context Menu.
- Select 'Advanced system settings'
- Click 'Environment Variables...' in the Advanced Tab
- Under 'System Variables': Click Edit
- Add python path to the end of the list (the paths are separated by semicolons(;))



Variable name: Path

Variable value: ı Files\Java\jdk1.8.0_121\bin;C:\Python36

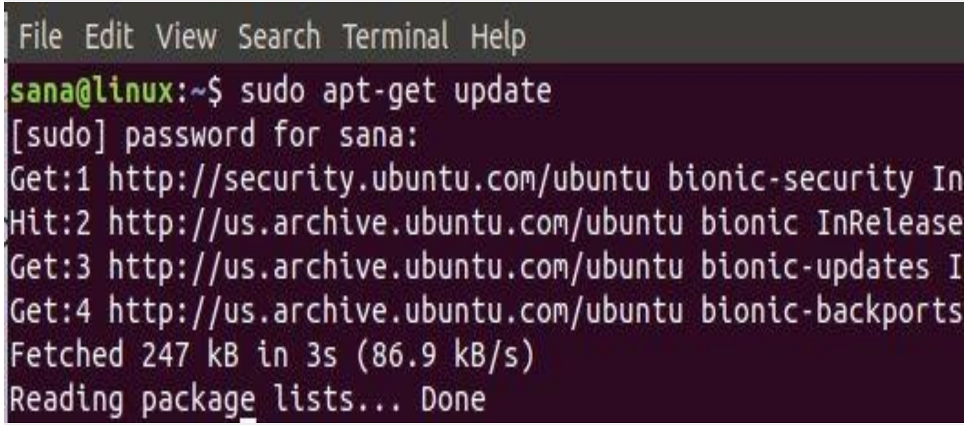
Variable	Value
OS	Windows_NT
Path	C:\ProgramData\Oracle\Java\javapath;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PROCESSOR_A...	AMD64

Image Source: https://net-informations.com/python/intro/path.htm#google_vignette

Setting Up Path

Installation of Python in Linux

- Linux is an open source Operating System. There are many Linux based operating systems. Popular are Ubuntu, Fedora, Linux Mint, Debian.
- Open the terminal or Command prompt from your linux based OS. Type the following commands.
- If you are using Ubuntu 16.0 or newer version, then you can easily install Python 3.6 or Python 2.7 by typing the following commands



```
sana@linux:~$ sudo apt-get update
[sudo] password for sana:
Get:1 http://security.ubuntu.com/ubuntu bionic-security In
Hit:2 http://us.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://us.archive.ubuntu.com/ubuntu bionic-updates I
Get:4 http://us.archive.ubuntu.com/ubuntu bionic-backports
Fetched 247 kB in 3s (86.9 kB/s)
Reading package lists... Done
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

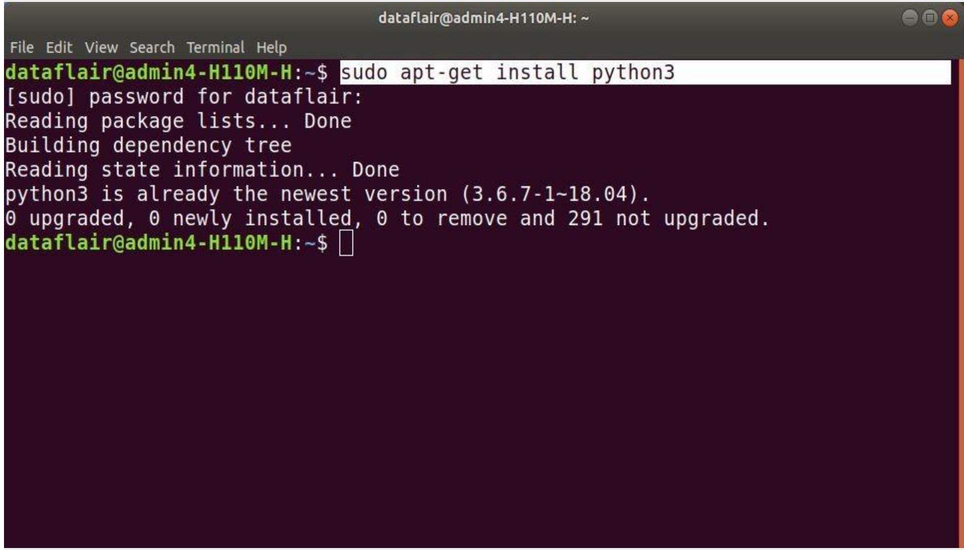
Setting Up Path

Installation of Python in Linux (Contd..)

- $ sudo apt-get update
- $ sudo apt-get install python3

or

- $ sudo apt-get update
- $ sudo apt-get install python2.7



dataflair@admin4-H110M-H: ~
File Edit View Search Terminal Help

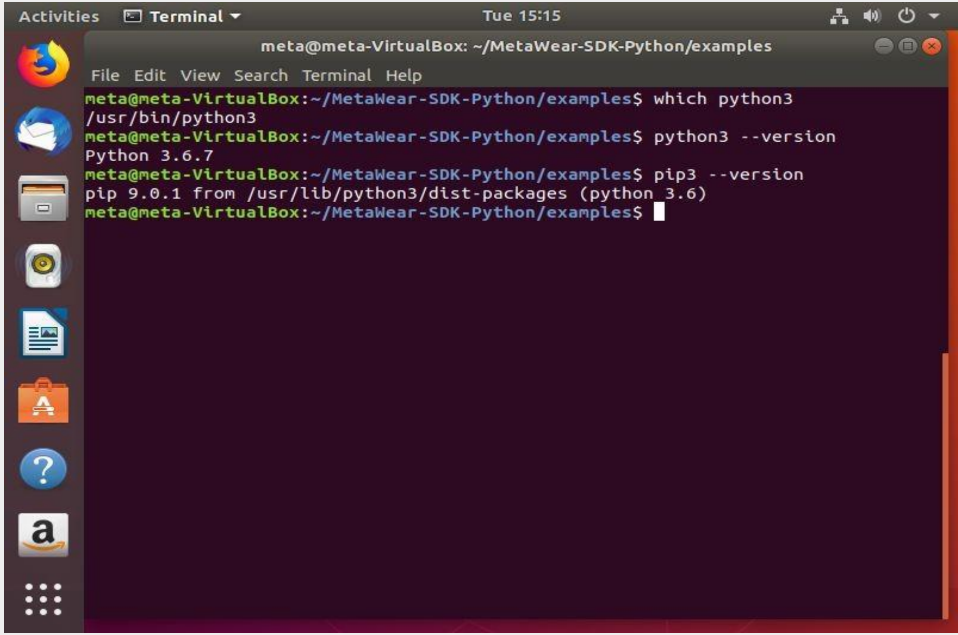
```
dataflair@admin4-H110M-H:~$ sudo apt-get install python3
[sudo] password for dataflair:
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.6.7-1~18.04).
0 upgraded, 0 newly installed, 0 to remove and 291 not upgraded.
dataflair@admin4-H110M-H:~$
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

Setting Up Path

Installation of Python in Linux (Contd..)

- You can check it is installed or not by
- Type the following commands in terminal.
- For Python3:-
- $ python3 --version
- For Python2:-
- $ python2 --version



Activities
Terminal
Tue 15:15
meta@meta-VirtualBox: ~/MetaWear-SDK-Python/examples
File Edit View Search Terminal Help

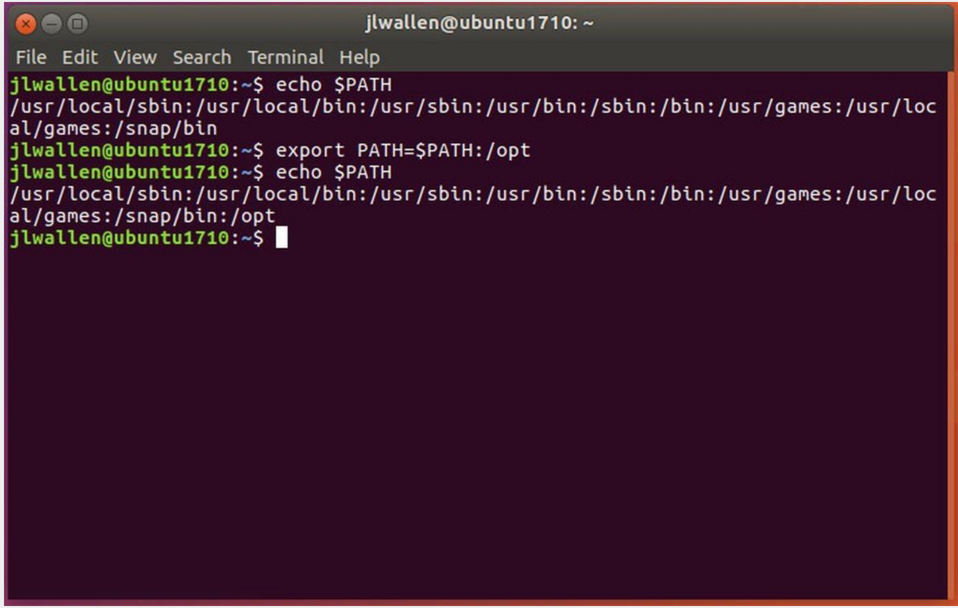
```
meta@meta-VirtualBox:~/MetaWear-SDK-Python/examples$ which python3
/usr/bin/python3
meta@meta-VirtualBox:~/MetaWear-SDK-Python/examples$ python3 --version
Python 3.6.7
meta@meta-VirtualBox:~/MetaWear-SDK-Python/examples$ pip3 --version
pip 9.0.1 from /usr/lib/python3/dist-packages (python 3.6)
meta@meta-VirtualBox:~/MetaWear-SDK-Python/examples$
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

Setting Up Path

How to Set Python Path in Linux

- The steps for adding Path in Linux are fairly straight forward, just follow the steps outlined below.
- In the csh shell, type the following sentence:
- PATH "$PATH:/usr/local/bin/python" and press Enter.



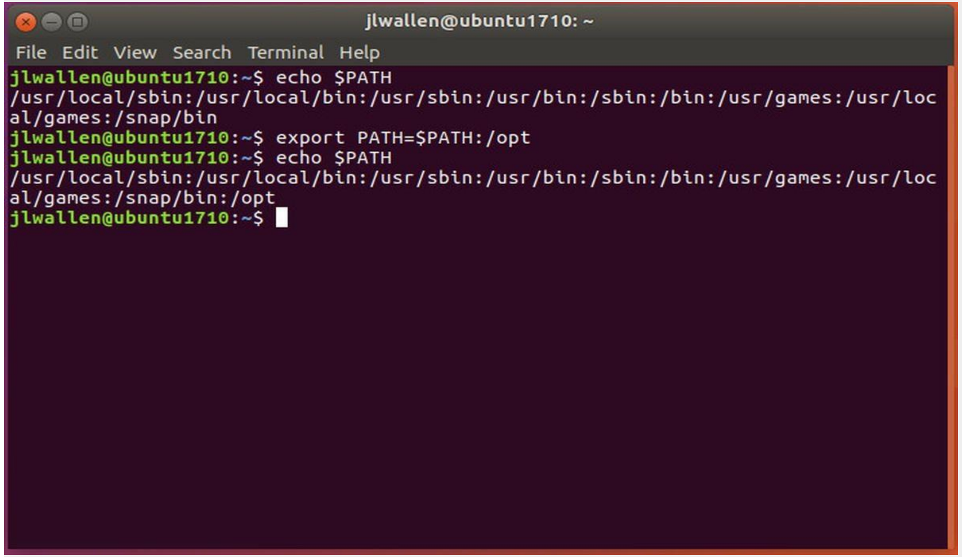
```
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
jlwallen@ubuntu1710:~$ export PATH=$PATH:/opt
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:/opt
jlwallen@ubuntu1710:~$ 
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

Setting Up Path

open up the bash shell and type the following phrase

- export PATH=”$PATH:/usr/local/bin/python” and press Enter.
- If you have access to either sh or ksh shell, then open up the terminal and type the following,
- PATH=”$PATH:/usr/local/bin/python” and press Enter.



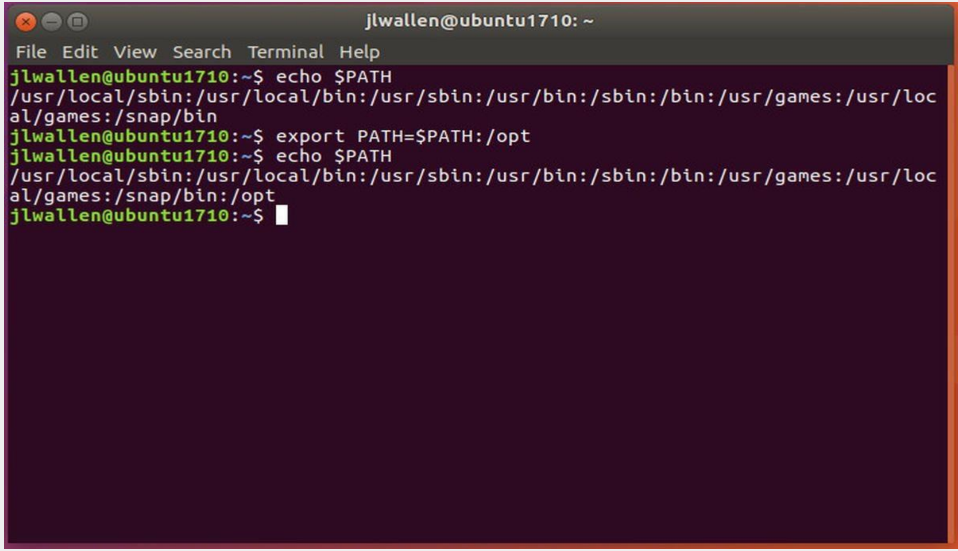
```
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
jlwallen@ubuntu1710:~$ export PATH=$PATH:/opt
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:/opt
jlwallen@ubuntu1710:~$
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

Setting Up Path

How to Set Python Path in Linux (Contd..)

- One of the most important things to note
- when you are adding Path to Python in Unix or Linux is that,
- /usr/local/bin/python is the default path of the Python directory.



```
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin
jlwallen@ubuntu1710:~$ export PATH=$PATH:/opt
jlwallen@ubuntu1710:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:/opt
jlwallen@ubuntu1710:~$ 
```

Image Source: https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

edunet foundation

Basic Syntax Variable

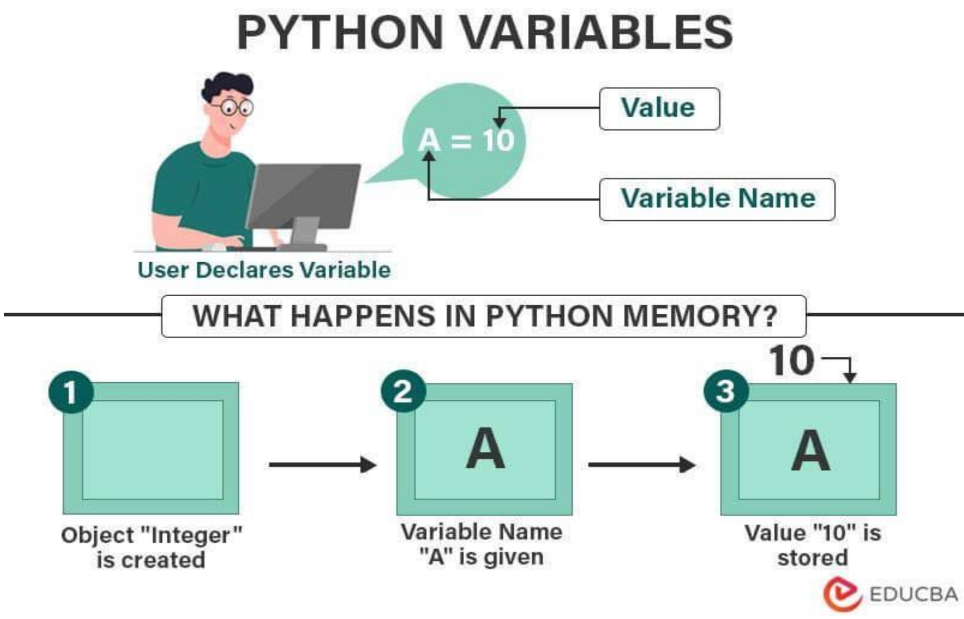


Image Source: https://images.app.goo.gl/pLedhKeCYfNXcyLy7

Basic Syntax Variable

Variables

- A variable is a container for a value.
- It can be assigned a name, you can use it to refer to it later in the program.
- Based on the value assigned, the interpreter decides its data type.

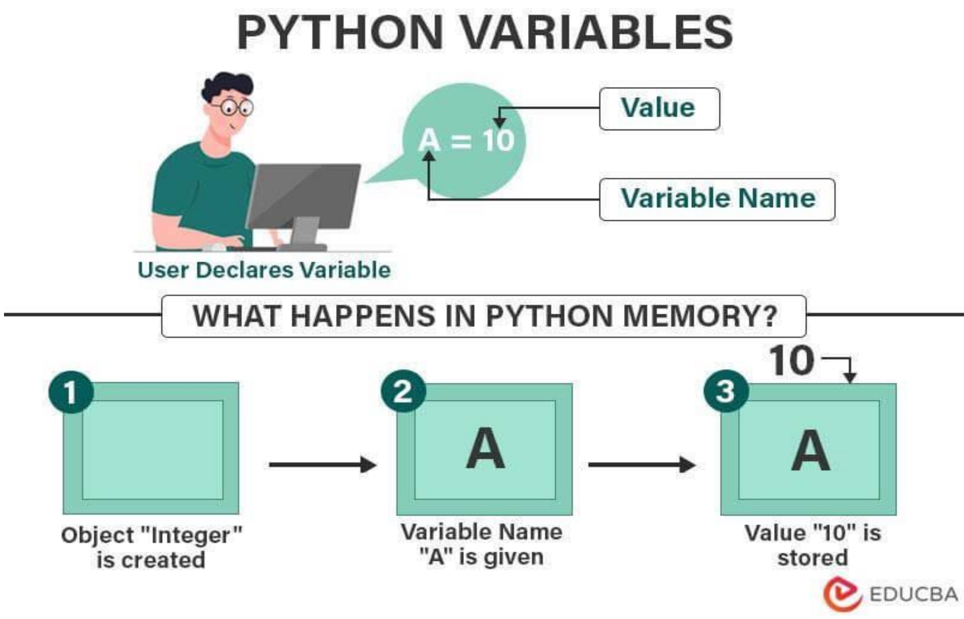
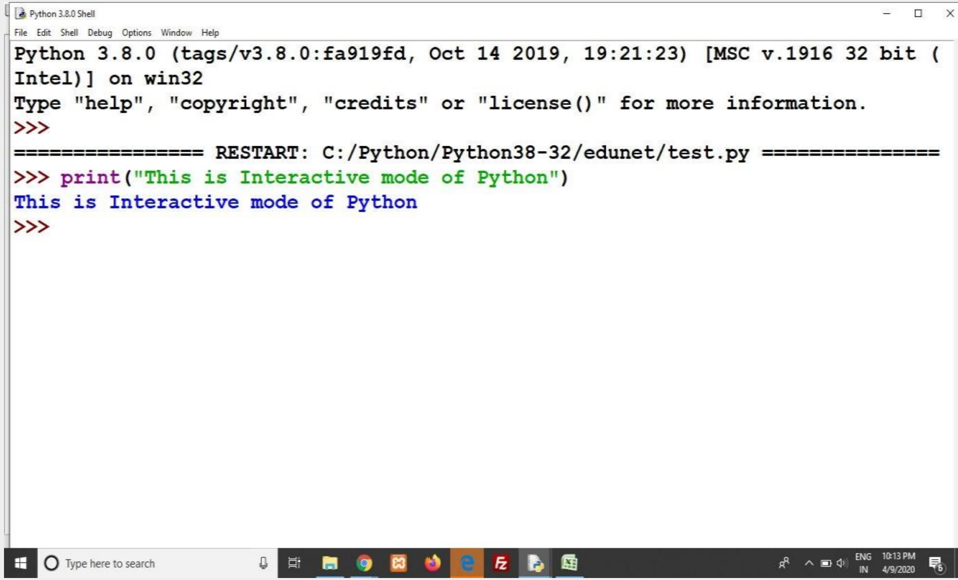


Image Source: https://images.app.goo.gl/pLedhKeCYfNXcyLy7

Basic Syntax Variable

Interactive Mode Programming

- In Python, the code executes via the Python Shell, which comes with Python Installation.
- To access the Python shell, open the terminal of your operating system and then type "python". Press the enter key and the Python shell will appear.

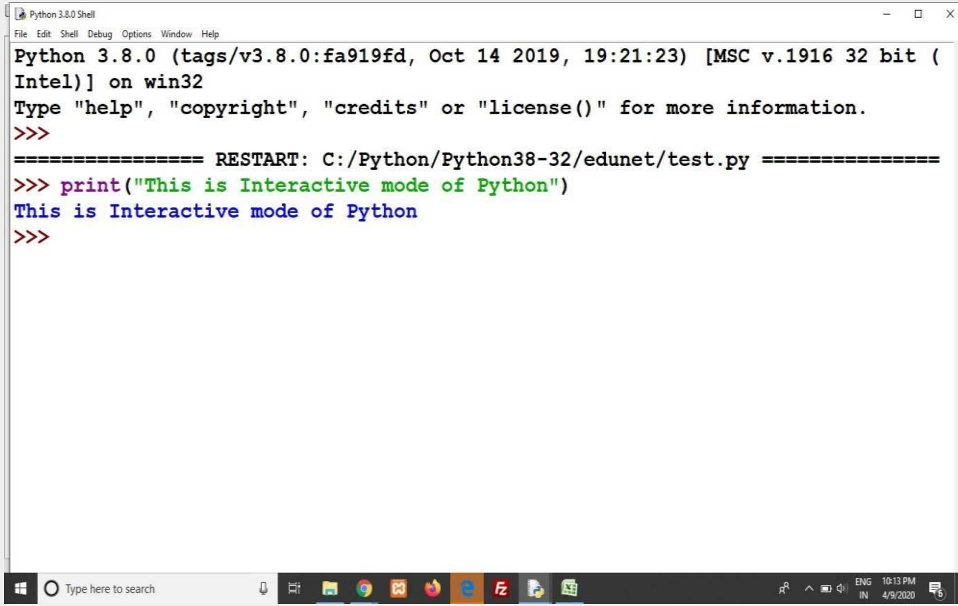


```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python/Python38-32/edunet/test.py ===============
>>> print("This is Interactive mode of Python")
This is Interactive mode of Python
>>>
```

Basic Syntax Variable

Interactive Mode Programming (Contd..)

- The indicates that the Python shell is ready to >>> execute and send your commands to the Python interpreter.
- The result is immediately displayed on the Python shell as soon as the Python interpreter interprets the command.

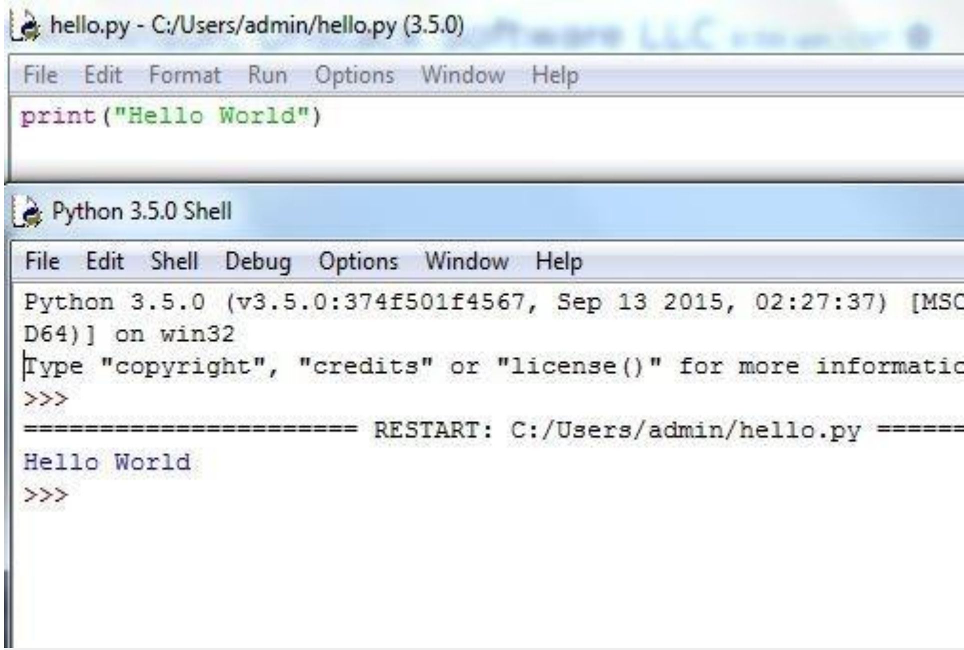


```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python/Python38-32/edunet/test.py ===============
>>> print("This is Interactive mode of Python")
This is Interactive mode of Python
>>>
```

Basic Syntax Variable

Script Mode Programming

- Script mode is used to work with lengthy codes.
- In Script mode, You write your code in a text file then save it with a .py extension.
- you can run your code by clicking”Run” then “Run module” or simply press F5.
- You can use any text editor to wite the code like- Sublime, Atom, Notepad++, etc.


hello.py - C:/Users/admin/hello.py (3.5.0)
File Edit Format Run Options Window Help
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help

```
print("Hello World")
```

```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:27:37) [MSC
D64)] on win32
Type "copyright", "credits" or "license()" for more informatic
>>>
====================== RESTART: C:/Users/admin/hello.py ======
Hello World
>>>
```

Basic Syntax Variable

Python Identifiers

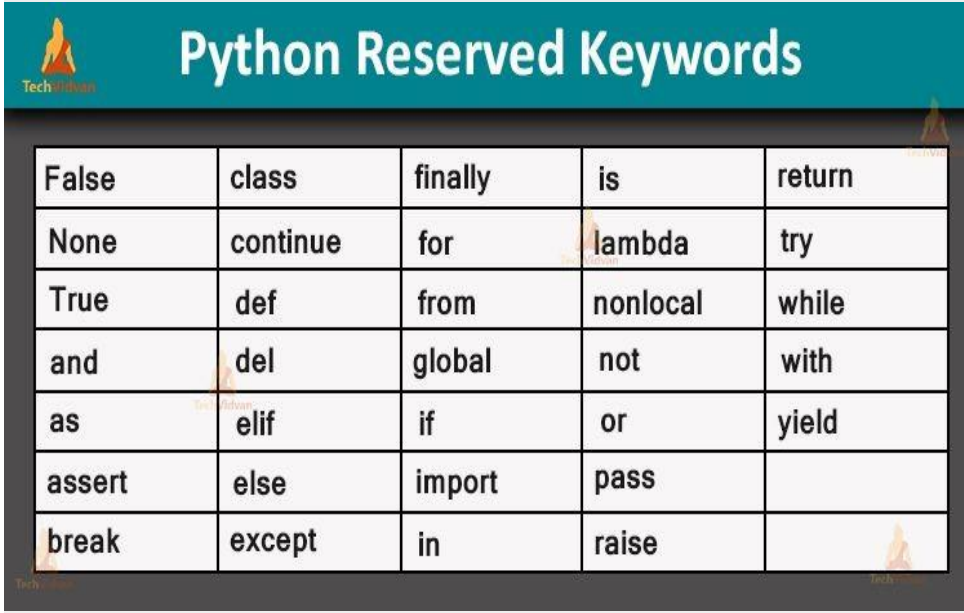
- A Python identifier is a name used to find a variable, function, class, module or other object.
- An identifier begin with a letter A to Z or a to z or an underscore(_) followed by zero or more letters and digits (0 to 9).
- Python doesn’t permit punctuation character such as @, $ and % within identifiers.
- Python is a case sensitive Programming Language.

Image Source: https://www.google.com/Python-Identifiers.jpg

Basic Syntax Variable

Reserved Keywords

- There are reserved words and cannot use them as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters only.



Python Reserved Keywords

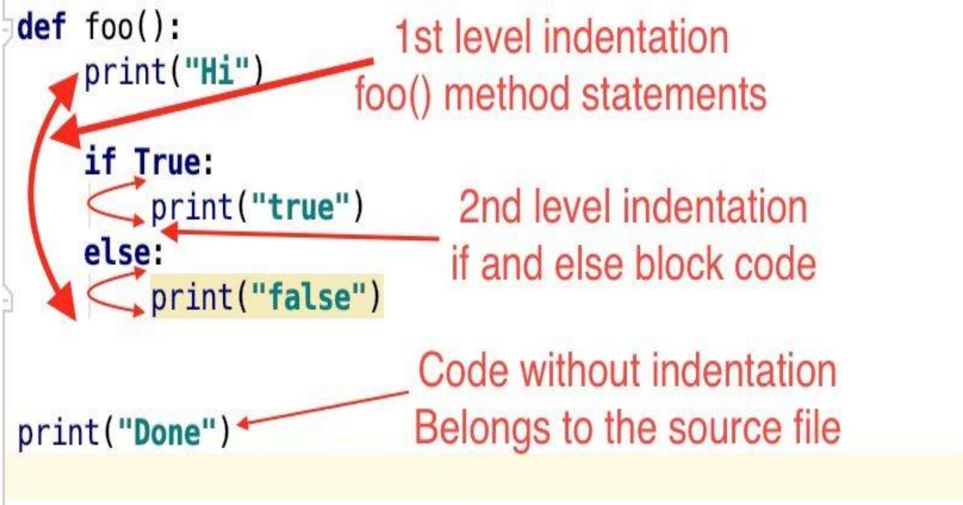
False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Image Source: https://images.app.goo.gl/Tz4Vm1dpgUkWGpWn9

Basic Syntax Variable

Lines and Indentation

- Python provides no braces to indicate blocks or code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statement within the block must be indented the same amount.



```
def foo():
    print("Hi")

    if True:
        print("true")
    else:
        print("false")


print("Done")
```

1st level indentation
foo() method statements

2nd level indentation
if and else block code

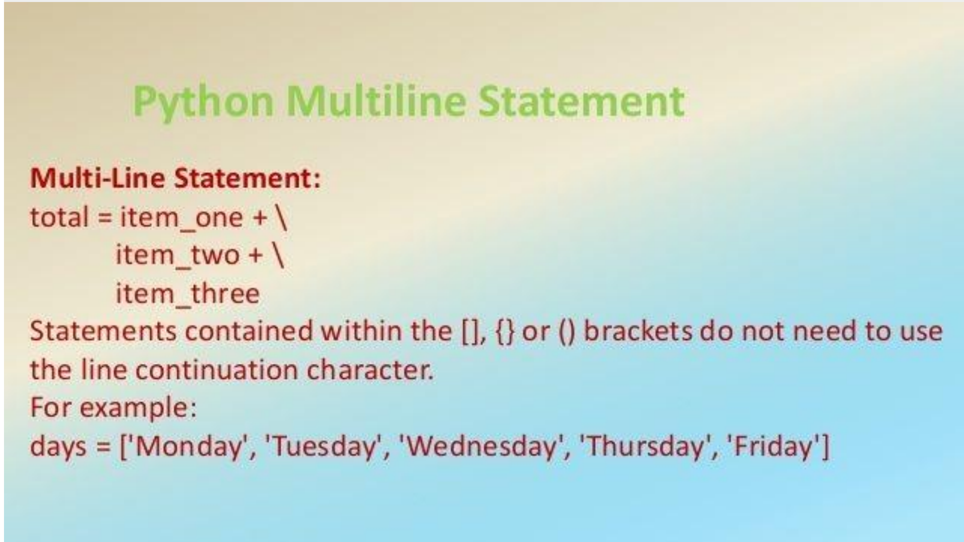
Code without indentation
Belongs to the source file

Image Source: https://images.app.goo.gl/rBAU8eBd2GsUXYtZ9

Basic Syntax Variable

Multi Line Statement

- Python is to be able to easily print across multiple lines.
- Statements contained within the [],{} or () brackets do not need to use the line continuation character.



Python Multiline Statement

Multi-Line Statement:

```
total = item_one + \
        item_two + \
        item_three
```

Statements contained within the [], {} or () brackets do not need to use the line continuation character.

For example:

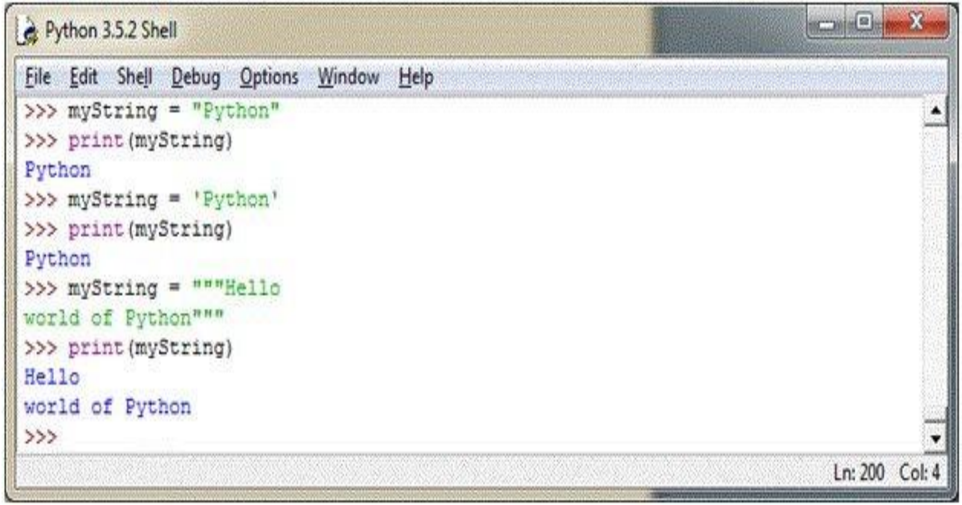
```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Image Source: https://www.slideshare.net/slideshow/fundamentals-of-python-language/124292692

Basic Syntax Variable

Quotation in Python

- Python accepts single (‘), double (“), and triple (''' or " "") quotes to denote string literals, as long as the same type of quote starts and ends the strings.
- The triple quotes are used to span the string across multiple lines.



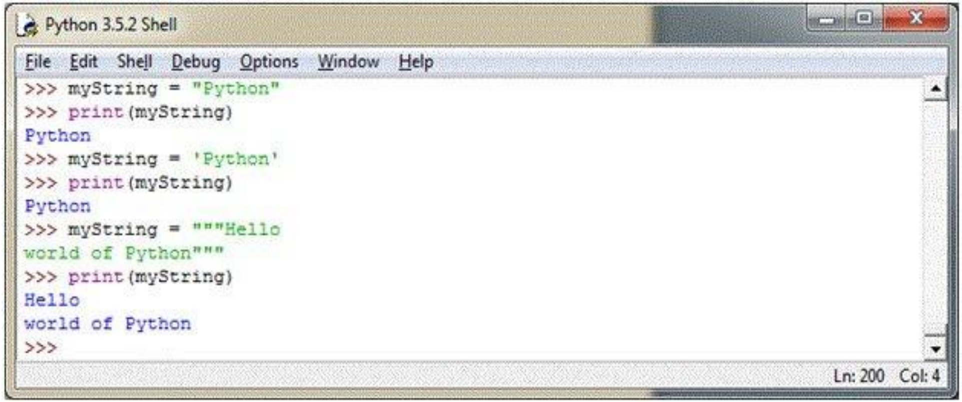
```
>>> myString = "Python"
>>> print(myString)
Python
>>> myString = 'Python'
>>> print(myString)
Python
>>> myString = """Hello
world of Python"""
>>> print(myString)
Hello
world of Python
>>>
```

Image Source: https://images.app.goo.gl/PFnJJHof1xDqqssV6

Basic Syntax Variable

Quotation in Python (Contd..)

- For example, all the followings are legal-
- word = ’word’
- sentence = “This is a sentence”
- Paragraph = “””This is a paragraph. It is made up of multiple lines and sentences.”””



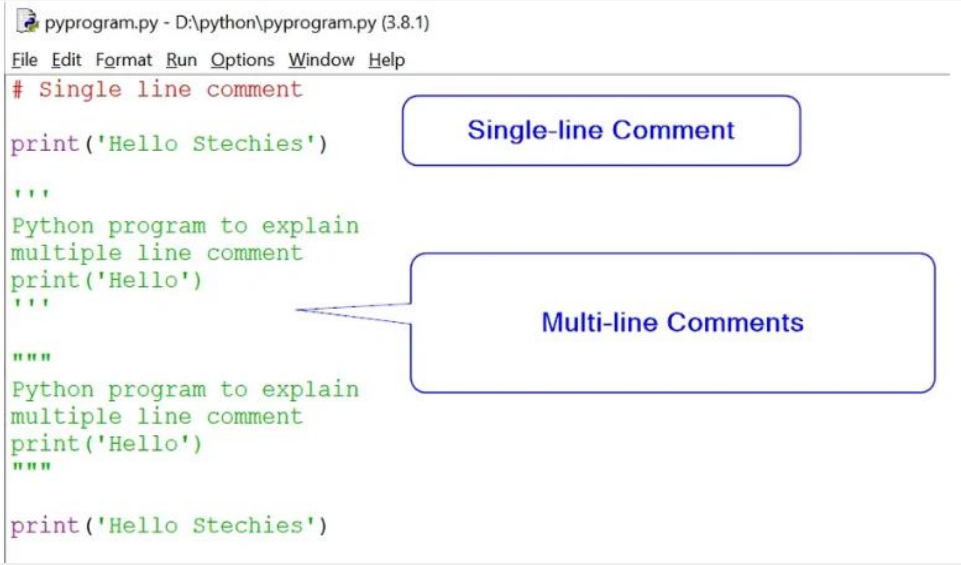
```
>>> myString = "Python"
>>> print(myString)
Python
>>> myString = 'Python'
>>> print(myString)
Python
>>> myString = """Hello
world of Python"""
>>> print(myString)
Hello
world of Python
>>>
```

Image Source: https://images.app.goo.gl/PFnJJHof1xDqqssV6

Basic Syntax Variable

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.



pyprogram.py - D:\python\pyprogram.py (3.8.1)
File Edit Format Run Options Window Help

```
# Single line comment

print('Hello Stechies')

'''
Python program to explain
multiple line comment
print('Hello')
'''

"""
Python program to explain
multiple line comment
print('Hello')
"""

print('Hello Stechies')
```

Single-line Comment

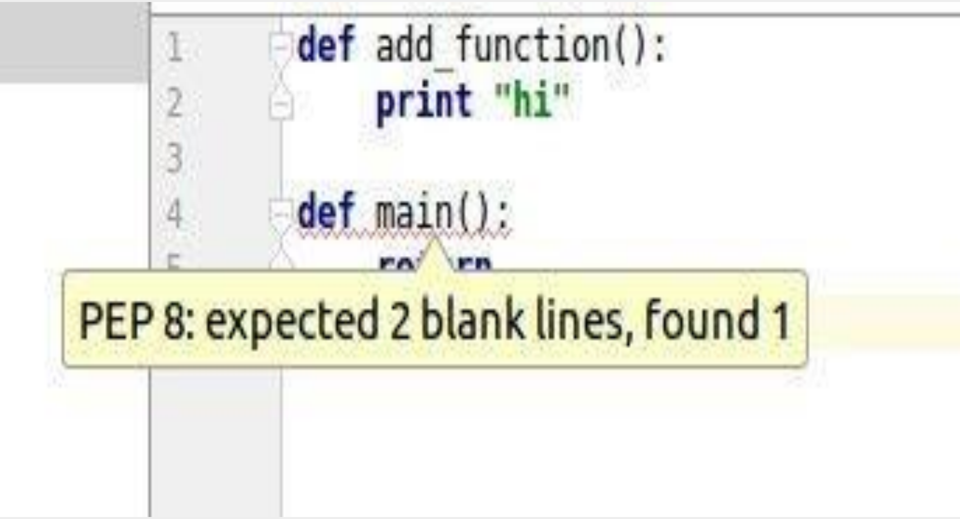
Multi-line Comments

Image Source: https://images.app.goo.gl/AgSoNnFVLUZArQFs5

Basic Syntax Variable

Using Blank Lines

- A line containing only whitespaces, possibly with a comment, is known as a blank line and Python totally ignores it.
- In an interactive interpreter session, you must enter physical line to terminate a multiple statement.



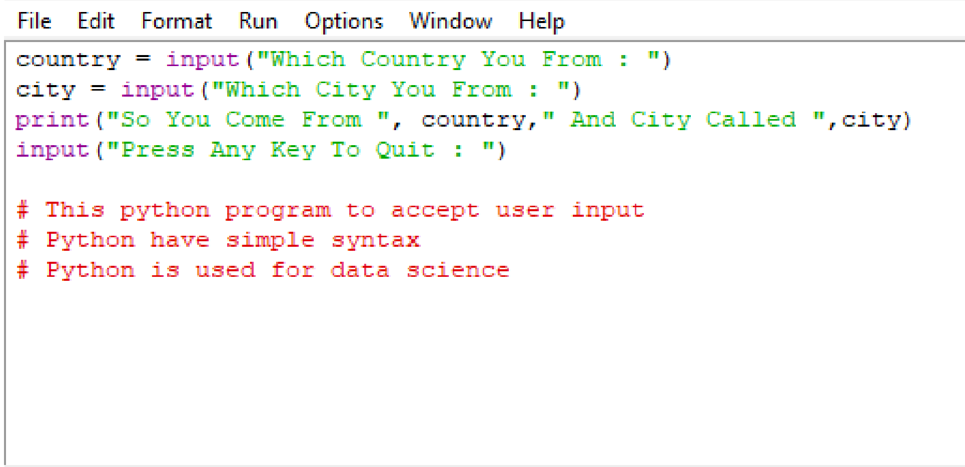
```
def add_function():
    print "hi"

def main():
    return
```

PEP 8: expected 2 blank lines, found 1

Basic Syntax Variable

Input from the User

- Builtin function - input()

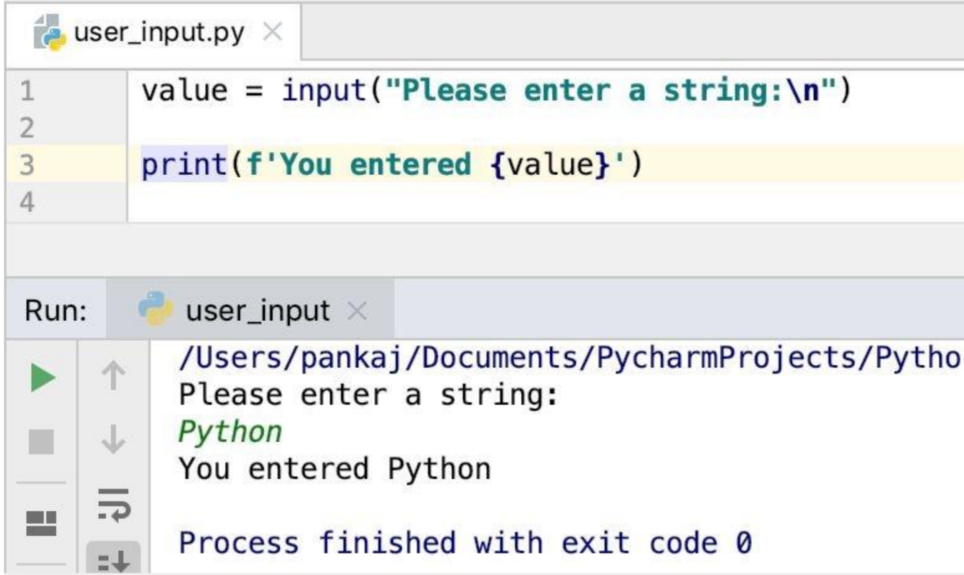


```
country = input("Which Country You From : ")
city = input("Which City You From : ")
print("So You Come From ", country," And City Called ",city)
input("Press Any Key To Quit : ")

# This python program to accept user input
# Python have simple syntax
# Python is used for data science
```

Basic Syntax Variable

Input from the User (Contd..)

- even when the user inputs an integer value, it will still be considered as a string.
- input()-input() interprets and evaluates the input entered by the user, which means if the user enters an integer, an integer will be returned and if the user enters a string, a string will be returned.



```
value = input("Please enter a string:\n")

print(f'You entered {value}')
```

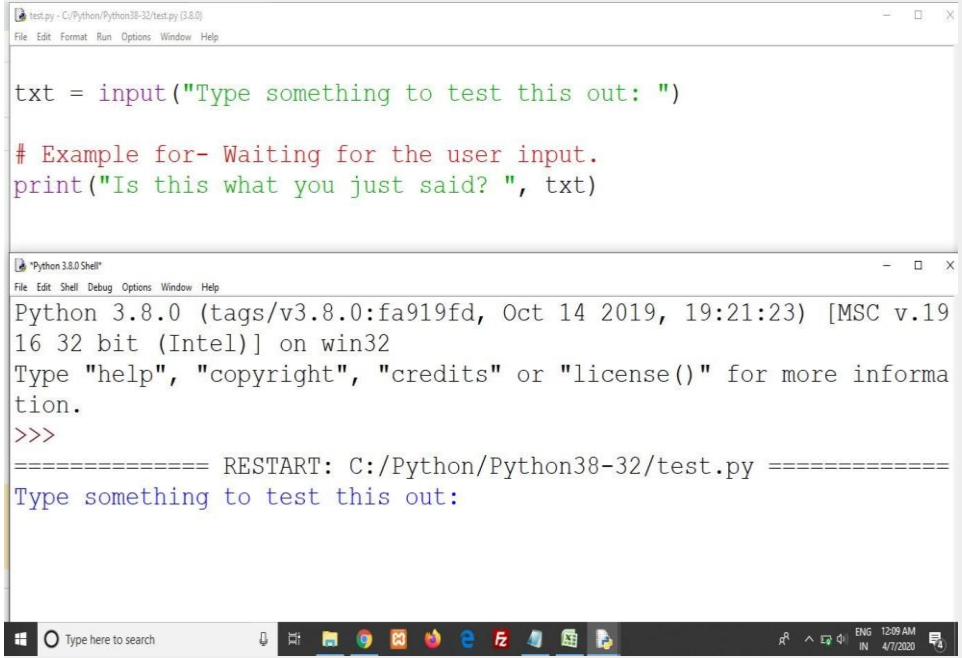
```
/Users/pankaj/Documents/PycharmProjects/Pytho
Please enter a string:
Python
You entered Python

Process finished with exit code 0
```

Image Source: https://images.app.goo.gl/UjRhrkMQH1wHB6XS8

Basic Syntax Variable

Waiting for the User

- The following line of the program displays the prompt, the statement saying “Press the Enter key to Exit”.
- Wait for the user to take action-
- raw_input(“/n/n Press the Enter key to Exit.”).
- Here, /n/n is used to create two new lines before displaying the actual line.



test.py - C:/Python/Python38-32/test.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG IN
12:09 AM 4/7/2020

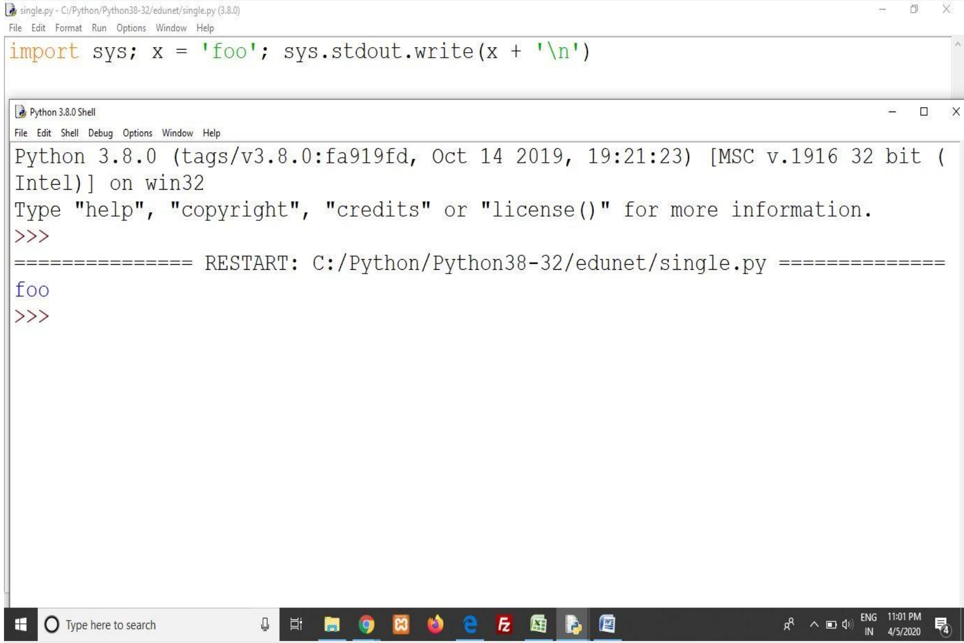
```
txt = input("Type something to test this out: ")

# Example for- Waiting for the user input.
print("Is this what you just said? ", txt)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
============== RESTART: C:/Python/Python38-32/test.py =============
Type something to test this out:
```

Basic Syntax Variable

Multiple Statement on a Single Line

- The Semicolon (;) allows multiple statements on the Single Line given that neither statement starts a new code block.
- Here is a simple snip using the semicolon-
- Import sys;x=”foo”sys.stdout.write(x+’\n’)



single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG IN 11:01 PM 4/5/2020

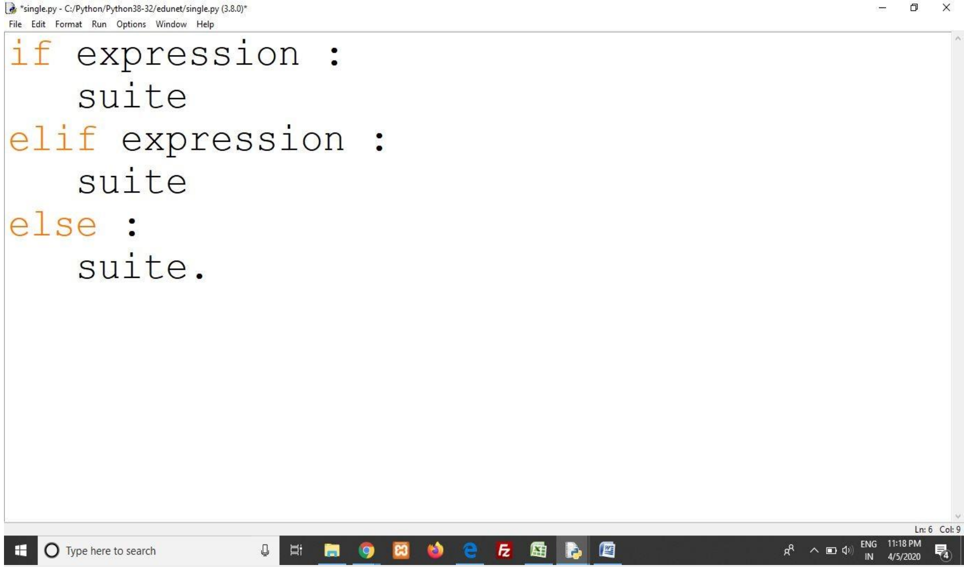
```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
=============== RESTART: C:/Python/Python38-32/edunet/single.py ==============
foo
>>> 
```

Basic Syntax Variable

Multiple Statement Groups as Suites

- A group of individual statements, which make a single code block are called Suites in Python.
- Compound or complex statements, such as if, while, def, and class require a header line and a suite.
- Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite.


single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Ln: 6 Col: 9
Type here to search
ENG IN
11:18 PM
4/5/2020

```
if expression :
   suite
elif expression :
   suite
else :
   suite.
```

Data Types Operator

Python
Operators
Arithmetic Operators
Relational Operators
Assignment Operators
Logical Operators
Bitwise Operators
Membership Operators
Identity Operators
Data Flair

Image Source: https://images.app.goo.gl/Lseqbjus8o1mXGJy8

Data Types Operator

Types of Operator

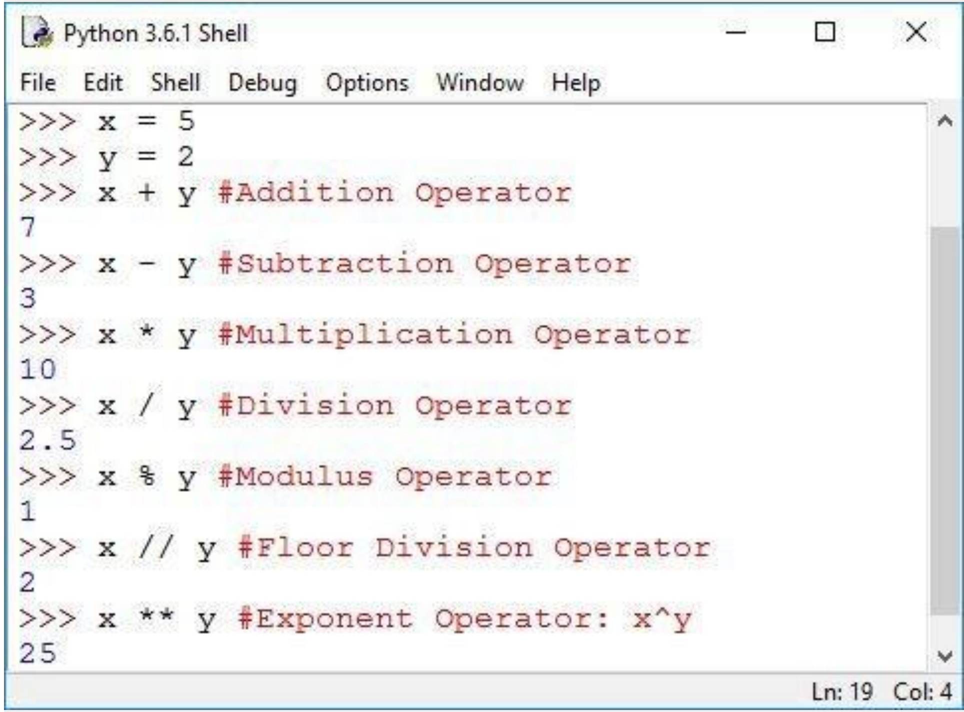
- Python Arithmetic Operator
- Python Comparison Operator
- Python Assignment Operator
- Python Bitwise Operator
- Python Logical Operator
- Python Membership Operator
- Python Identity Operator
- Python Operator Precedence

Image Source: https://images.app.goo.gl/Lseqbjus8o1mXGJy8

Data Types Operator

Python Arithmetic Operator

- Arithmetic Operators are used to perform mathematical operations like addition, subtraction, multiplication etc.



```
>>> x = 5
>>> y = 2
>>> x + y #Addition Operator
7
>>> x - y #Subtraction Operator
3
>>> x * y #Multiplication Operator
10
>>> x / y #Division Operator
2.5
>>> x % y #Modulus Operator
1
>>> x // y #Floor Division Operator
2
>>> x ** y #Exponent Operator: x^y
25
```

Image Source: https://images.app.goo.gl/D8YpxfkyZ1aHSaYq5

Data Types Operator

Python Comparison Operator

- Comparison Operators are used to compare values.
- It either returns True or False according to the condition

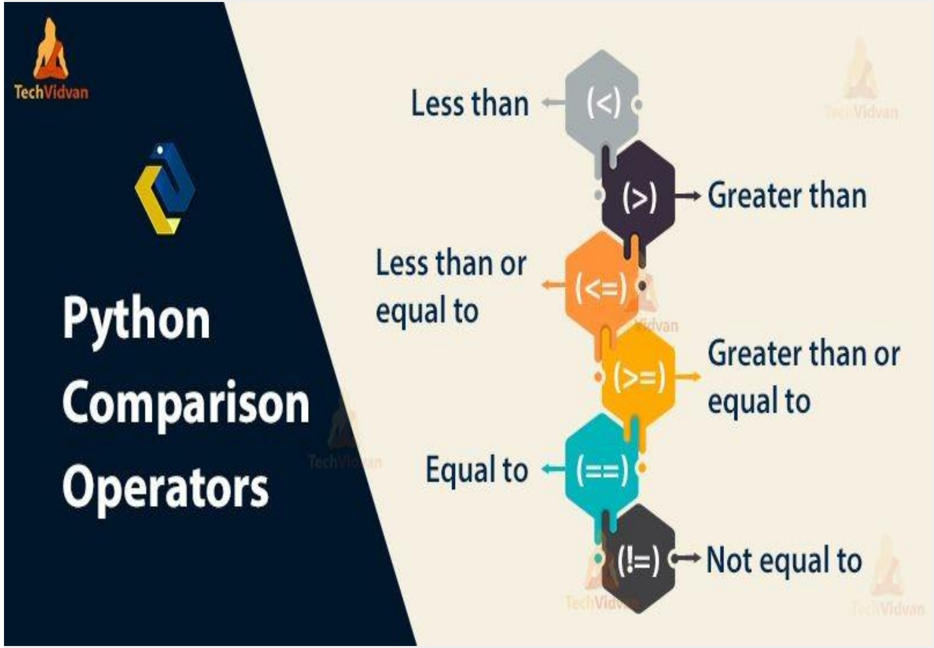


Image Source: https://images.app.goo.gl/P3ceUaPYPVn6WhUdA

Data Types Operator

Python Assignment Operator

- Assignment Operator are used to Python to assign values to variable.
- Equals (=) operator is the most commonly used assignment operator in Python.

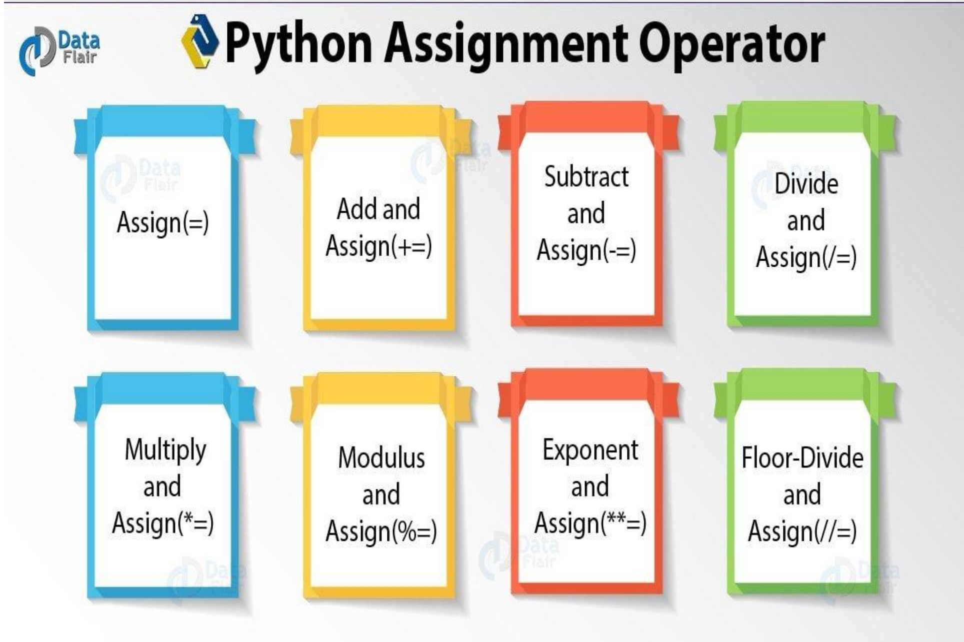
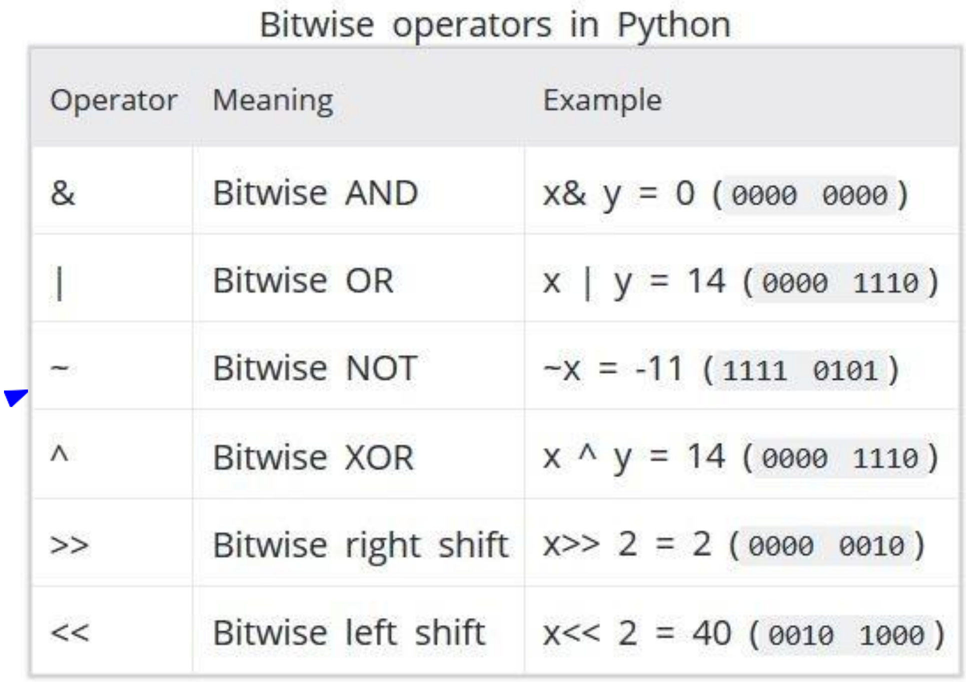


Image Source: https://images.app.goo.gl/mMu7mwghd7G5Wkwg7

Data Types Operator

Python Bitwise Operator

- Bitwise Operators act an Operand as if they were string of binary digits. It operates bit by bit hence, the name.
- For eg- 2 is 10 in binary and 7 is 111.
- Now In this table
- Let x=10 (0000 1010 in binary) and y=4 (0000
- 0100 in binary)



Bitwise operators in Python

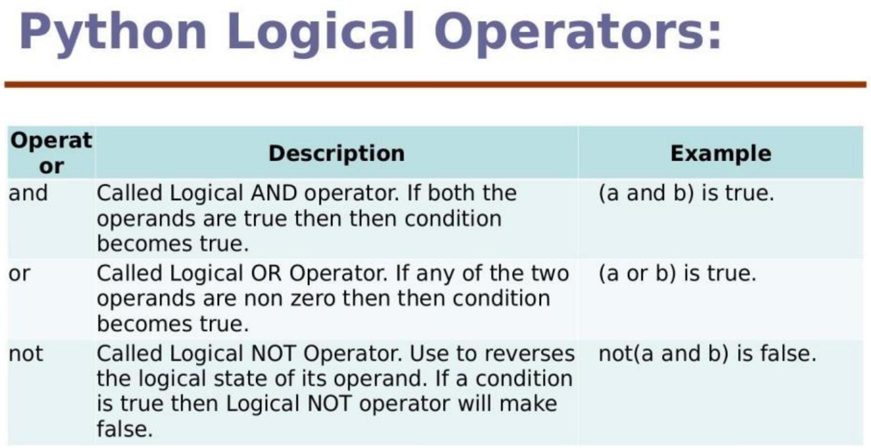
Operator	Meaning	Example
&	Bitwise AND	x& y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x>> 2 = 2 (0000 0010)
<<	Bitwise left shift	x<< 2 = 40 (0010 1000)

Image Source: https://images.app.goo.gl/pMuXXSQoJ7E9wCTk6

Data Types Operator

Python Logical Operator

- There are three types of Python Logical Operator-
  - And Operator
  - Or Operator
  - Not Operator



Python Logical Operators:

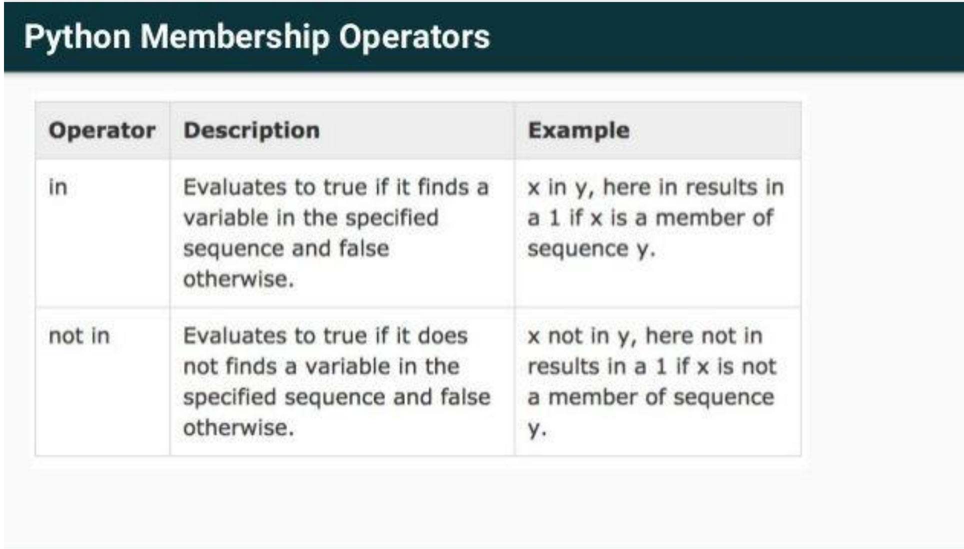
Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Image Source: https://images.app.goo.gl/Mw1wjPASLrZ8SBgYA

Data Types Operator

Python Membership Operator

- in and not in are the membership operators in Python.
- They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).



Python Membership Operators

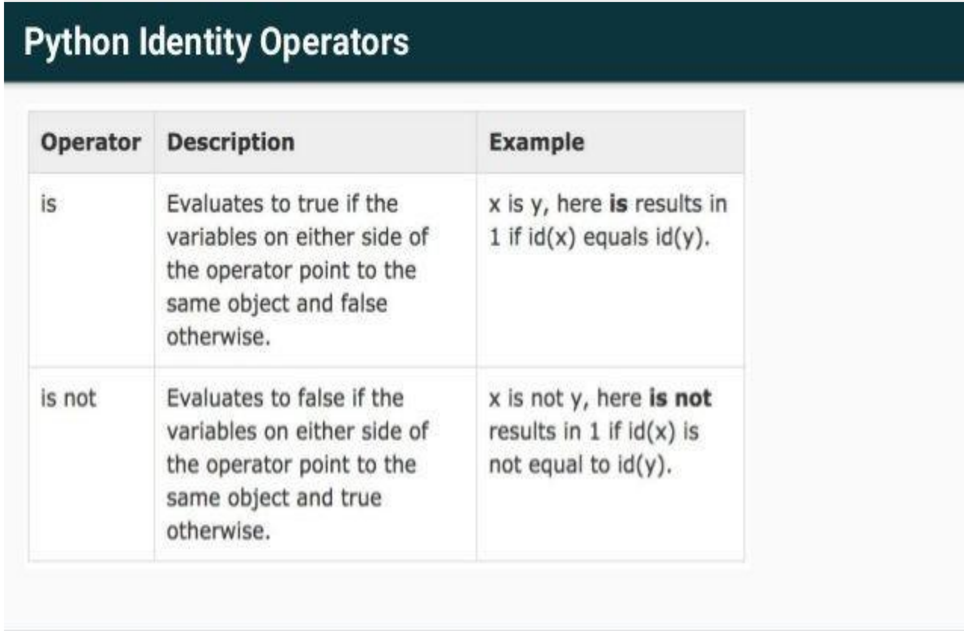
Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Image Source: https://images.app.goo.gl/3NA6GpPmD5b9ZRbb7

Data Types Operator

Python Identity Operator

- is and is not are the identity operators in Python.
- They are used 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.



Python Identity Operators

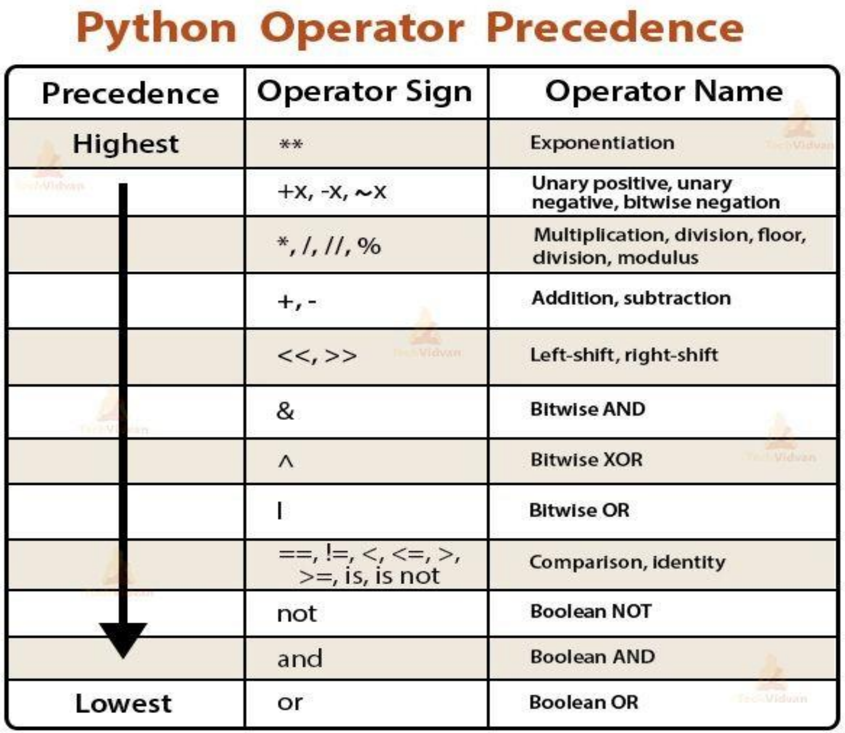
Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Image Source: https://images.app.goo.gl/Yj7kCuftbUTZo4aX6

Data Types Operator

Python Operators Precedence

- From the highest Precedence to the
- lowest down the table.
- Operators on the same row have the same Precedence.

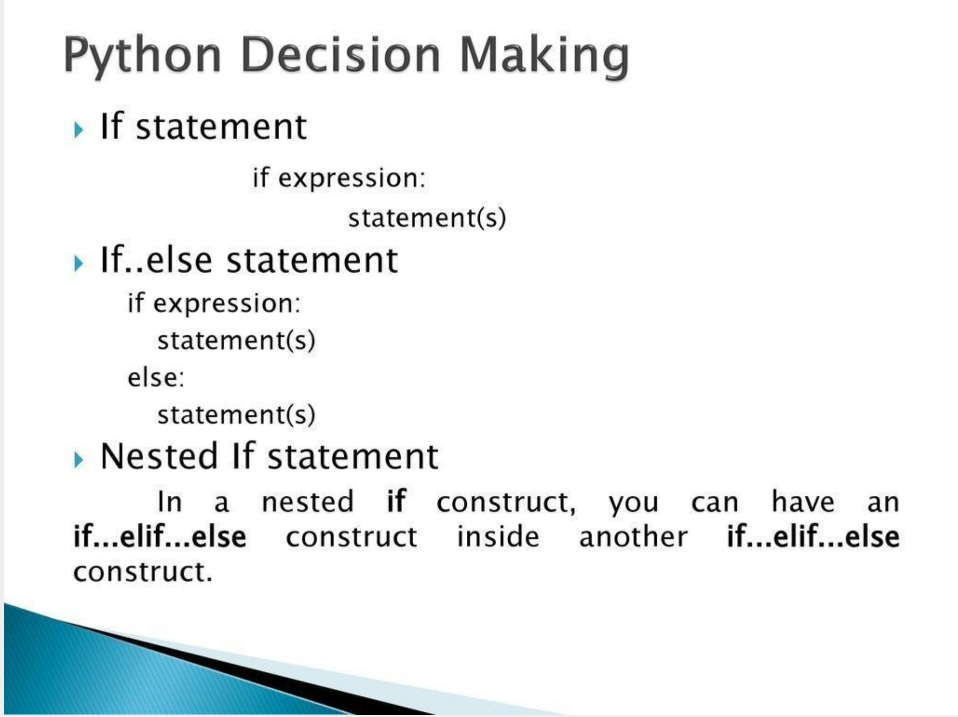


Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+x, -x, ~x	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

Image Source: https://images.app.goo.gl/7CS1HQToMXXx6Tet8

Conditional Statement



Python Decision Making

- If statement

```
if expression:
        statement(s)
```

- If..else statement

```
if expression:
   statement(s)
else:
   statement(s)
```

- Nested If statement

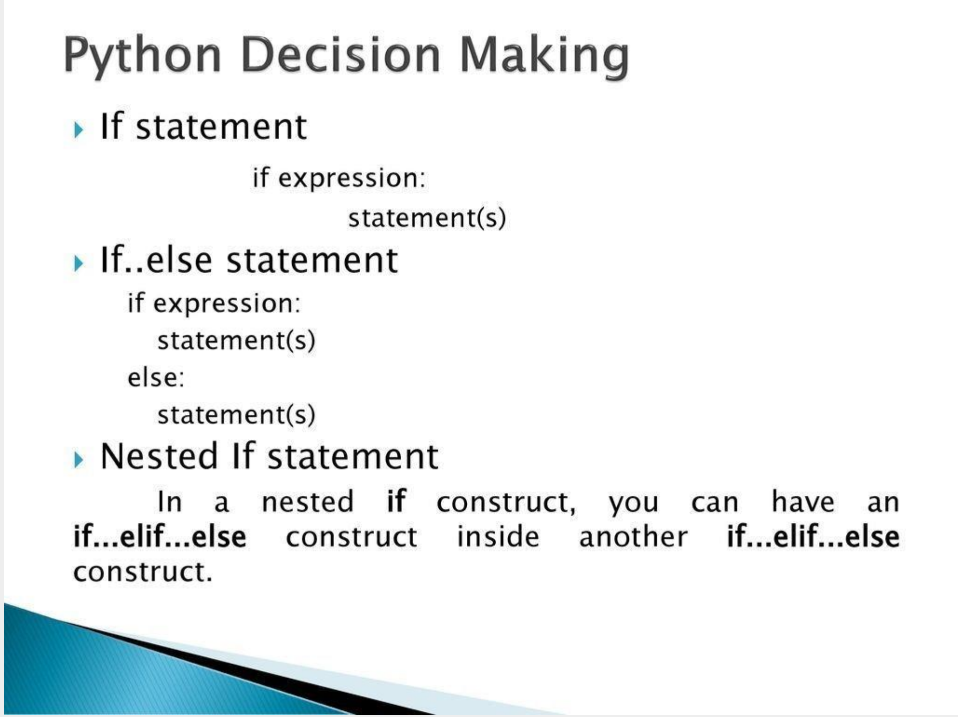
In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Conditional Statement

Statement And Description (Contd..)

- The basic Decision statements in Computer is Selection Structure.
- The Decision is described to computer as a conditional statement that can be answered True or False.
- Python language provide the following conditional (Decision Making) statements.

  If Statement
  If...else Statement
  If...elif...else Statement
  Nested if...else Statement



Python Decision Making

- If statement

```
        if expression:
                statement(s)
```

- If..else statement

```
 if expression:
   statement(s)
 else:
   statement(s)
```

- Nested If statement

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Conditional Statement

Statement And Description (Contd..)

- If statement-

The If statement is decision making statement.
It is used to control the flow of the statement and also used to test logically whether the condition is true or false.

- Syntax –

  if test expression:

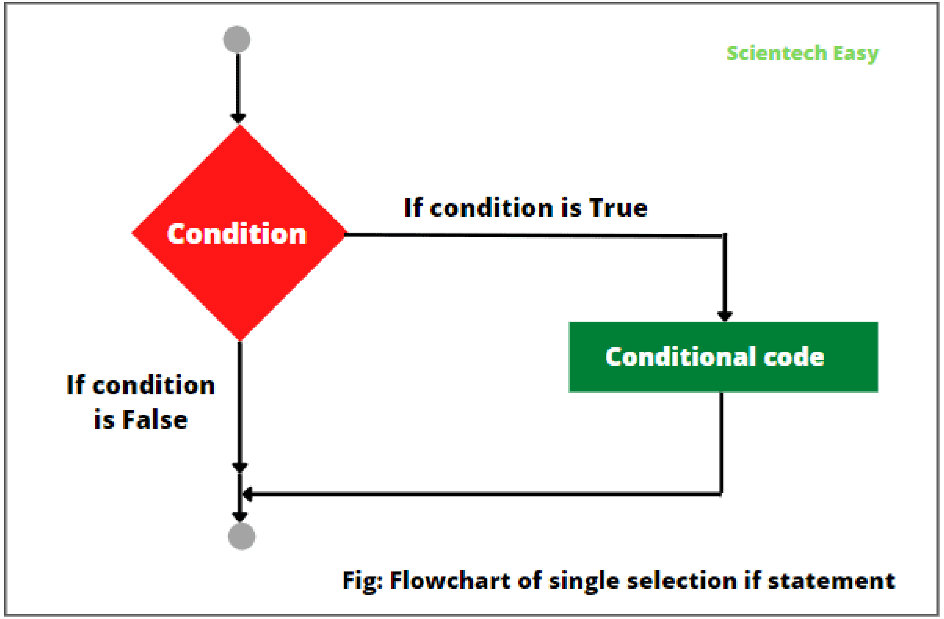


Fig: Flowchart of single selection if statement

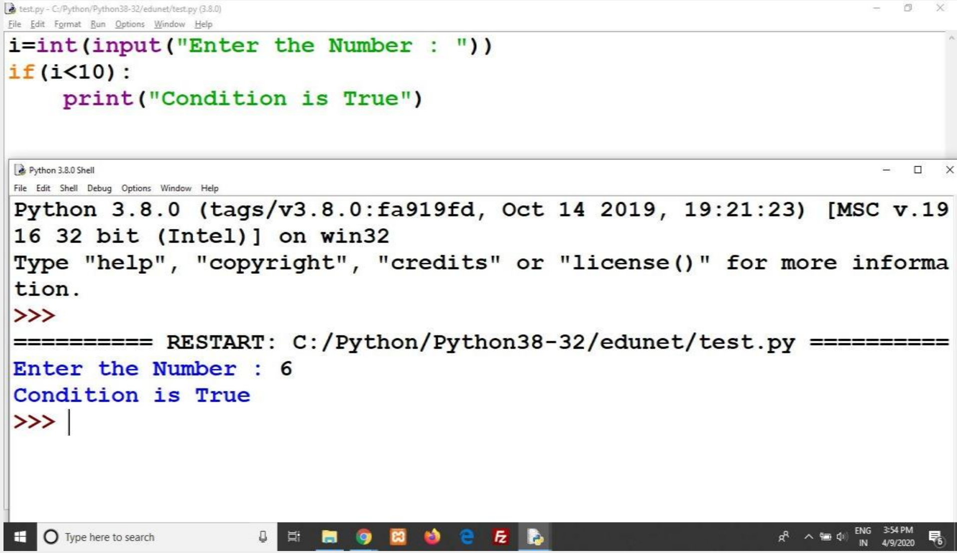
Image Source: https://images.app.goo.gl/6Wn2Zck8wQYJAfDG9

Conditional Statement

Statement And Description(Contd..)

- Example Program:

```
i=int(input("Enter the Number : "))
if(i<10):
    print("Condition is True")
```



test.py - C:/Python/Python38-32/edunet/test.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG IN
3:54 PM 4/9/2020

```
i=int(input("Enter the Number : "))
if(i<10):
    print("Condition is True")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========== RESTART: C:/Python/Python38-32/edunet/test.py ==========
Enter the Number : 6
Condition is True
>>> 
```

Conditional Statement

Statement And Description (Contd..)

- If..else statement-

The if..else statement is called alternative execution, in which there are two possibilities and the condition determines which one get executed.

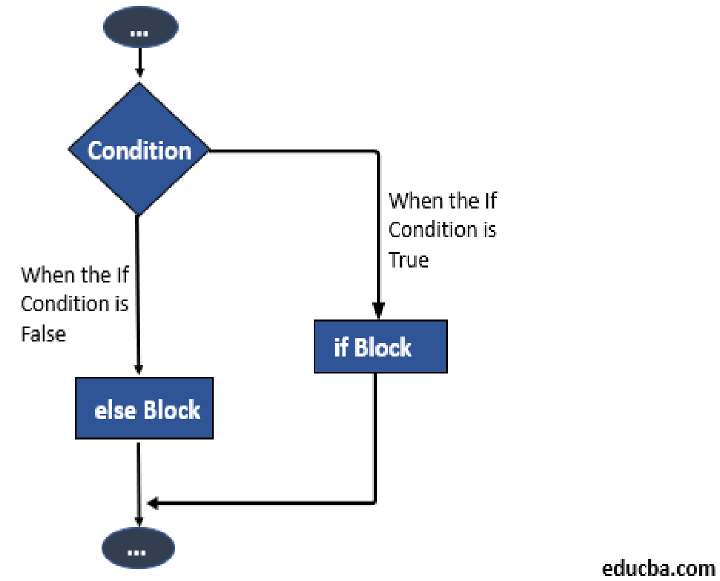


Image Source: https://images.app.goo.gl/qQHZX3JDoYjmG6sE6

Conditional Statement

Statement And Description (Contd..)

- Syntax-

```
if test expression:
    body of if
else:
    body of else
```

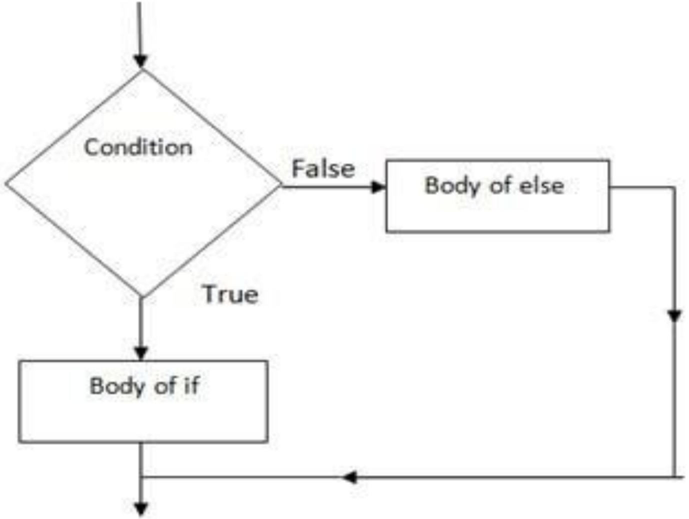
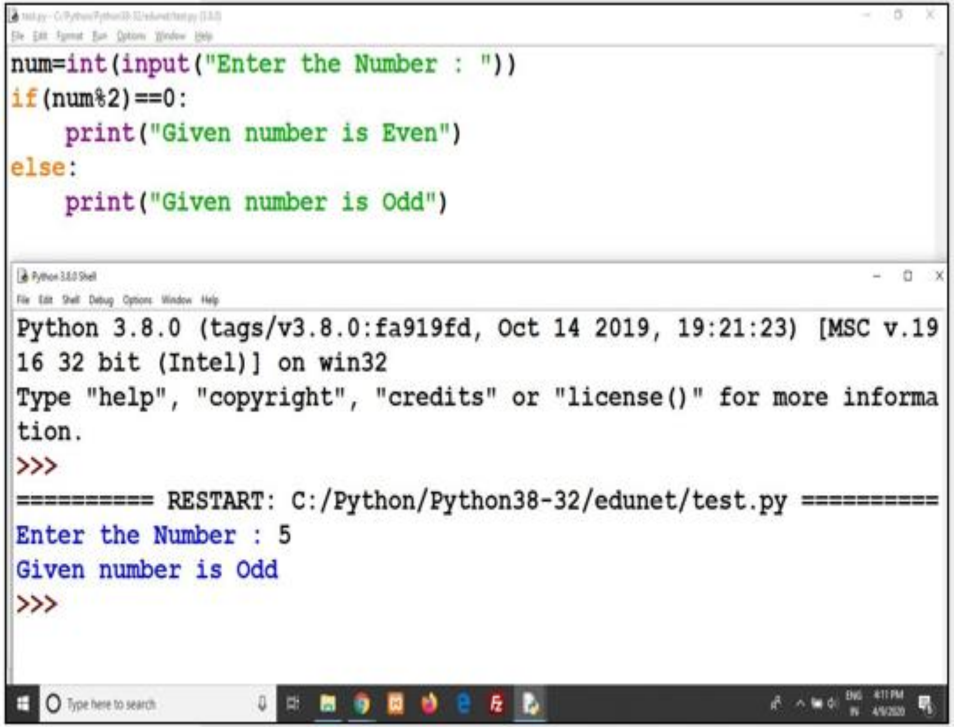


Image Source: https://images.app.goo.gl/Yj7kCuftbUTZo4aX6

Conditional Statement

```
num=int(input("Enter the Number : "))
if(num%2)==0:
    print("Given number is Even")
else:
    print("Given number is Odd")
```


test.py - C:/Python/Python38-32/edunet/test.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search

```
num=int(input("Enter the Number : "))
if(num%2)==0:
    print("Given number is Even")
else:
    print("Given number is Odd")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========== RESTART: C:/Python/Python38-32/edunet/test.py ==========
Enter the Number : 5
Given number is Odd
>>>
```

Conditional Statement

Statement And Description (Contd..)

- Elif Statement-

Elif is a keyword used in python in replacement of else if to place another condition in the program.
This is called chained conditional.
Chained conditions allows than two possibilities and need more than two branches.

Figure – elif condition Flowchart

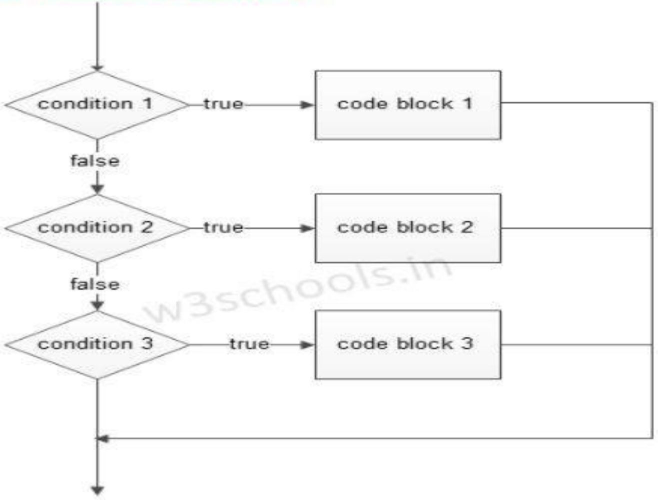


Image Source: https://images.app.goo.gl/einqW78QMyGTifcT8

Conditional Statement

Statement And Description (Contd..)

```
if expression:
        body of if
elif expression:
        body of elif
else:
        body of else
```

Figure – elif condition Flowchart

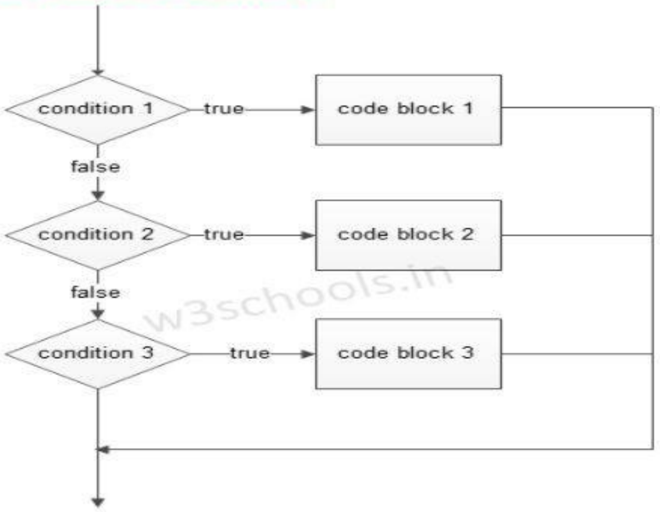
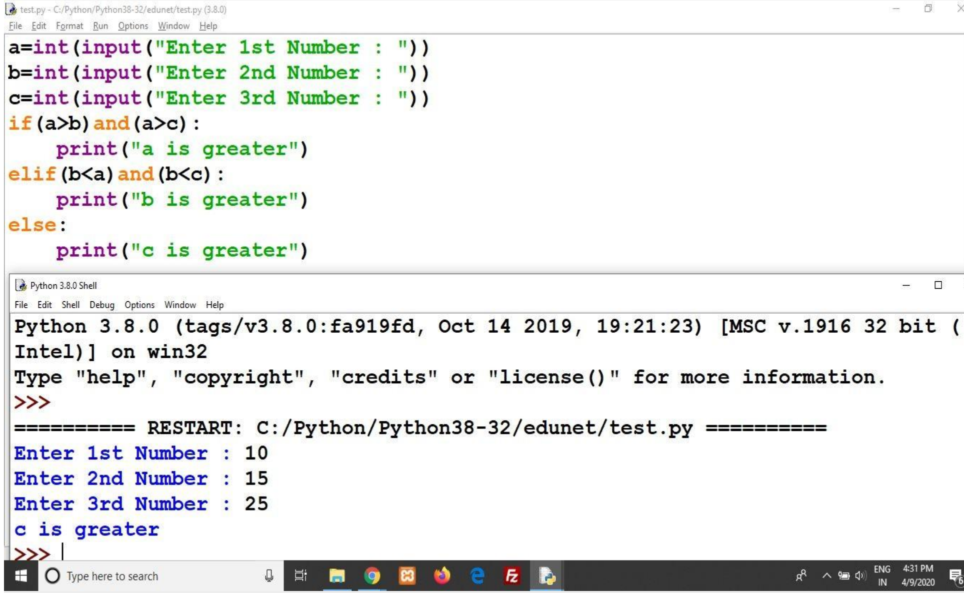


Image Source: https://images.app.goo.gl/einqW78QMyGTifcT8

Conditional Statement



test.py - C:/Python/Python38-32/edunet/test.py (3.8.0)
File Edit Format Run Options Window Help

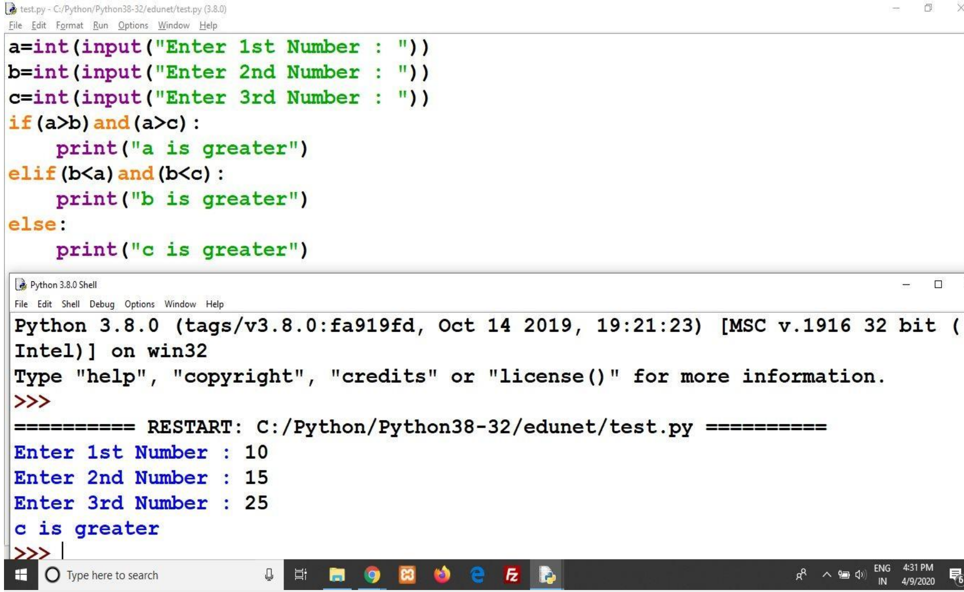
```
a=int(input("Enter 1st Number : "))
b=int(input("Enter 2nd Number : "))
c=int(input("Enter 3rd Number : "))
if(a>b)and(a>c):
    print("a is greater")
elif(b<a)and(b<c):
    print("b is greater")
else:
    print("c is greater")
```

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Python/Python38-32/edunet/test.py ==========
Enter 1st Number : 10
Enter 2nd Number : 15
Enter 3rd Number : 25
c is greater
>>> 
```

Conditional Statement

Statement And Description (Contd..)



test.py - C:/Python/Python38-32/edunet/test.py (3.8.0)
File Edit Format Run Options Window Help

```python
a=int(input("Enter 1st Number : "))
b=int(input("Enter 2nd Number : "))
c=int(input("Enter 3rd Number : "))
if(a>b)and(a>c):
    print("a is greater")
elif(b<a)and(b<c):
    print("b is greater")
else:
    print("c is greater")
```

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:/Python/Python38-32/edunet/test.py ==========
Enter 1st Number : 10
Enter 2nd Number : 15
Enter 3rd Number : 25
c is greater
>>> |
```

Conditional Statement

Statement And Description (Contd..)

- Nested if...else statement-

We can write an entire if..else statement in another if..else statement called nesting, and the statement is called nested if.
In a nested if construct, you can have an if..elif..else construct inside an if..elif..Else construct.

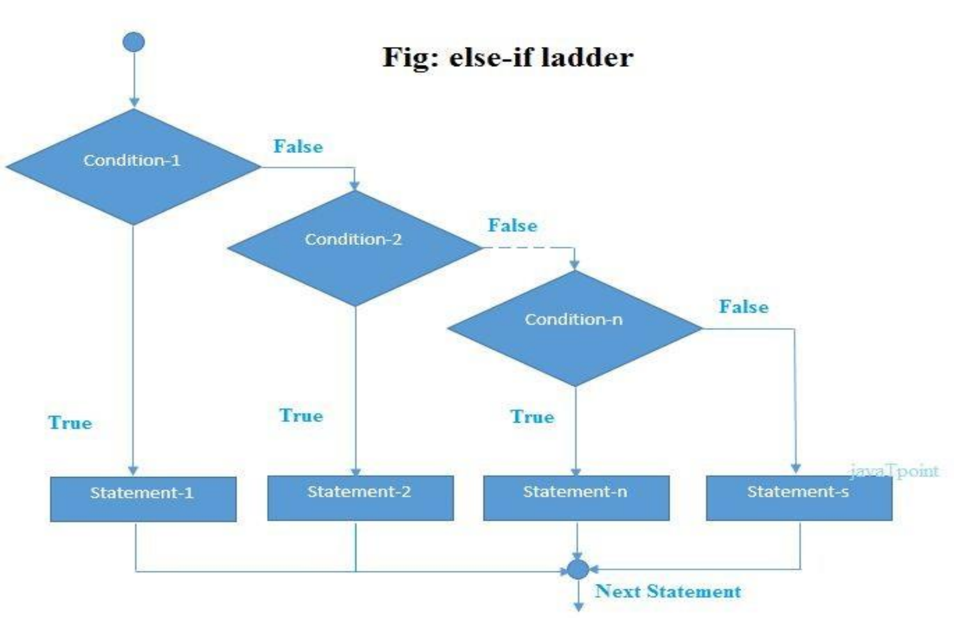


Image Source: https://images.app.goo.gl/yeUcQAudRfaBz9XC9

Conditional Statement

Statement And Description (Contd..)

```
If expression1:
      statement(s)
If expression2:
      statement(s)
Elif expression3:
      statement(s)
Else:
      statement(s)
```

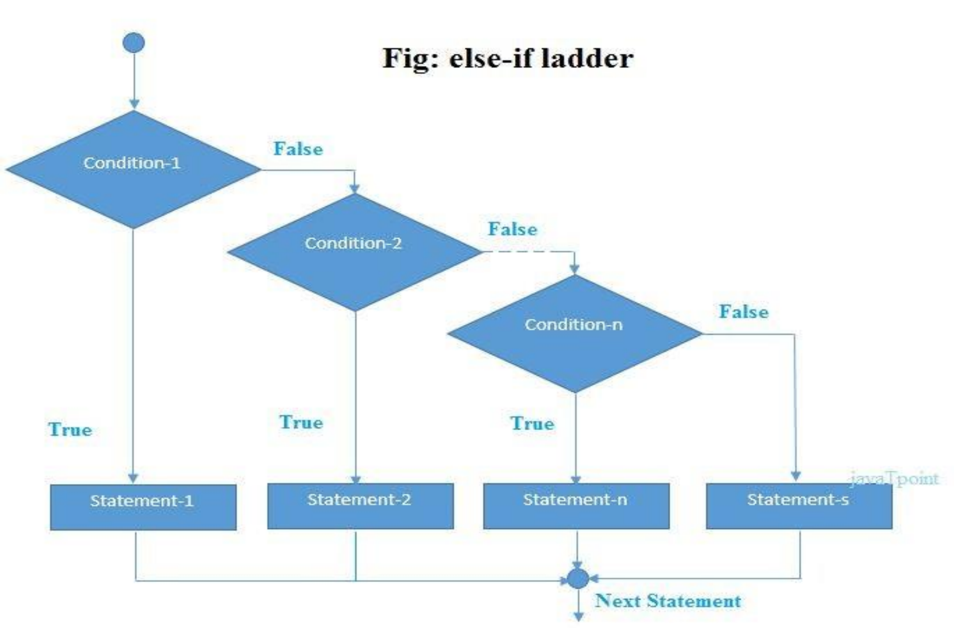


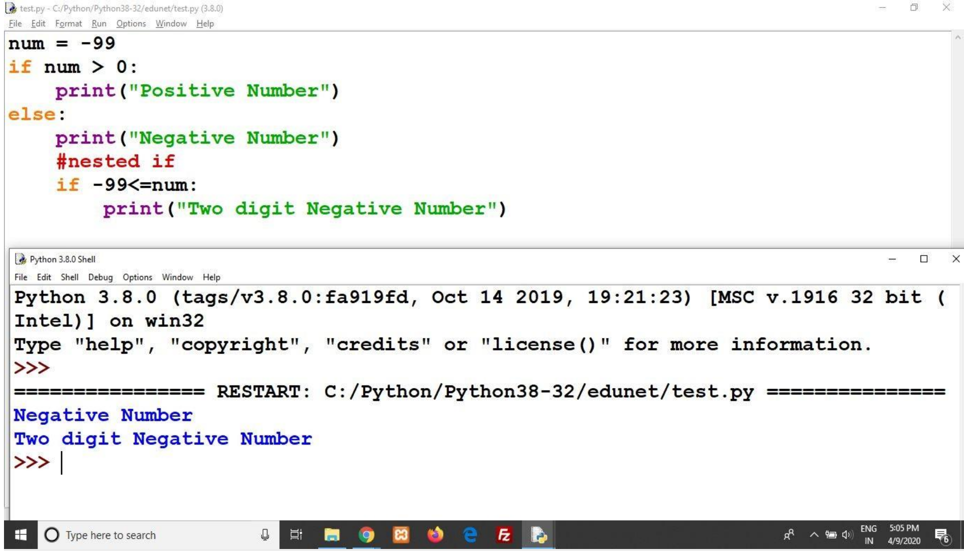
Image Source: https://images.app.goo.gl/yeUcQAudRfaBz9XC9

Conditional Statement

Statement And Description (Contd..)

- Example Program-

```
num = -99
if num > 0:
    print("Positive Number")
else:
    print("Negative Number")
    #nested if
    if -99<=num:
        print("Two digit Negative Number")
```


test.py - C:/Python/Python38-32/edunet/test.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG
IN
5:05 PM
4/9/2020

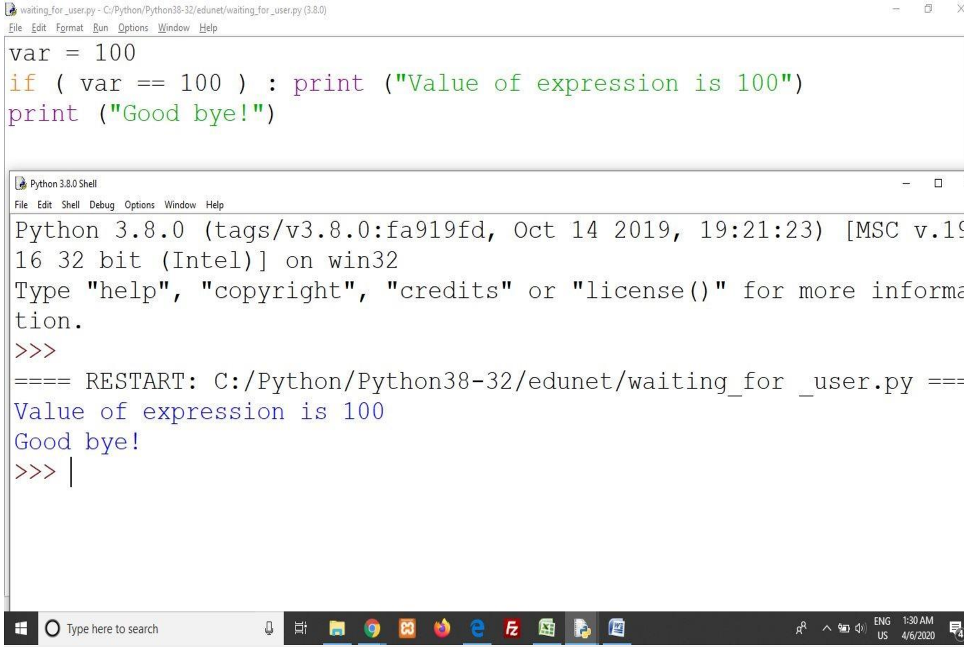
```
num = -99
if num > 0:
    print("Positive Number")
else:
    print("Negative Number")
    #nested if
    if -99<=num:
        print("Two digit Negative Number")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (
Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python/Python38-32/edunet/test.py ==============
Negative Number
Two digit Negative Number
>>> 
```

Conditional Statement

Single Statement Suites

- If the suite of an if clause consists only of a single line, it may go on the same line as the header statement.
- Here is an example of a one-line if clause −


waiting_for _user.py - C:/Python/Python38-32/edunet/waiting_for _user.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG US
1:30 AM 4/6/2020

```
var = 100
if ( var == 100 ) : print ("Value of expression is 100")
print ("Good bye!")
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
==== RESTART: C:/Python/Python38-32/edunet/waiting_for _user.py ===
Value of expression is 100
Good bye!
>>> 
```

Looping

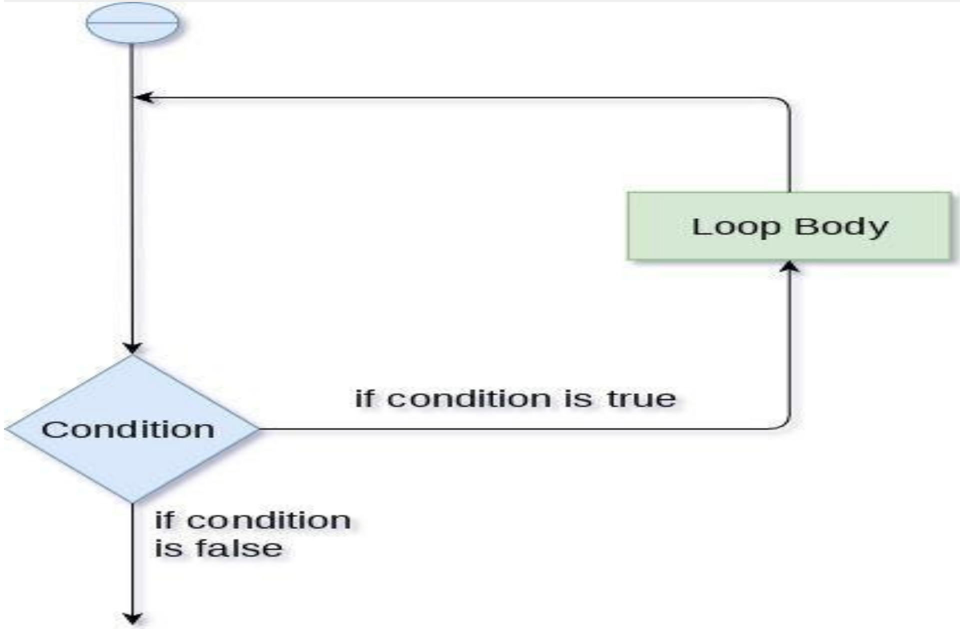


Image Source: https://images.app.goo.gl/4RUMqV2539YMXMQN7

Looping

Loop Types and Description

- The first statement in a function is executed first, followed by the second, and so on.
- . There may be a situation when you need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times.
- The following diagram illustrates a loop statement −

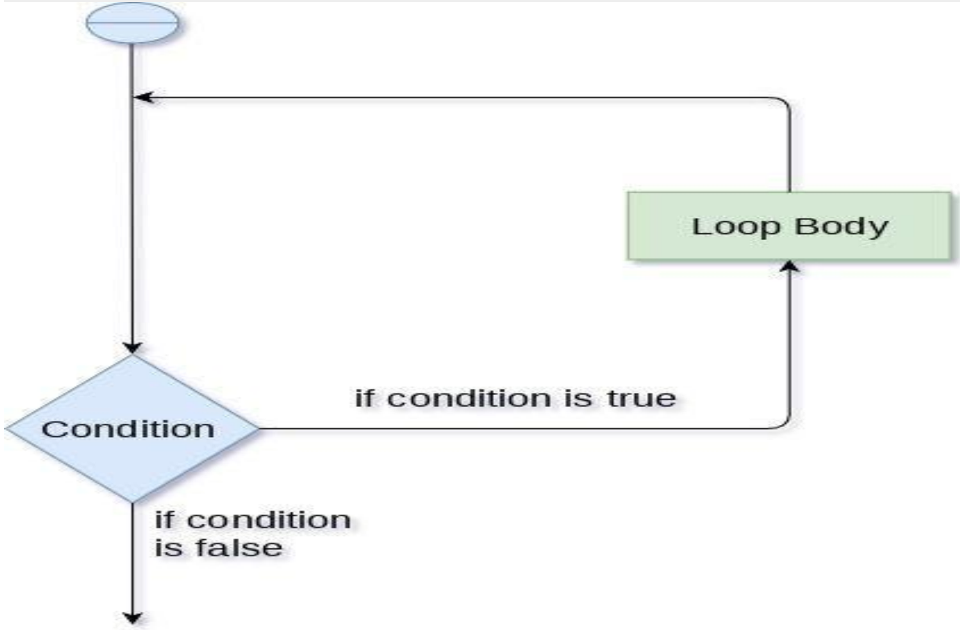


Image Source: https://images.app.goo.gl/4RUMqV2539YMXMQN7

Looping

Loop Structure

- Program statement are executed sequentially one after another. In some situations, a block of code needs of times.
- These are repetitive program codes, the computers have to perform to complete task,

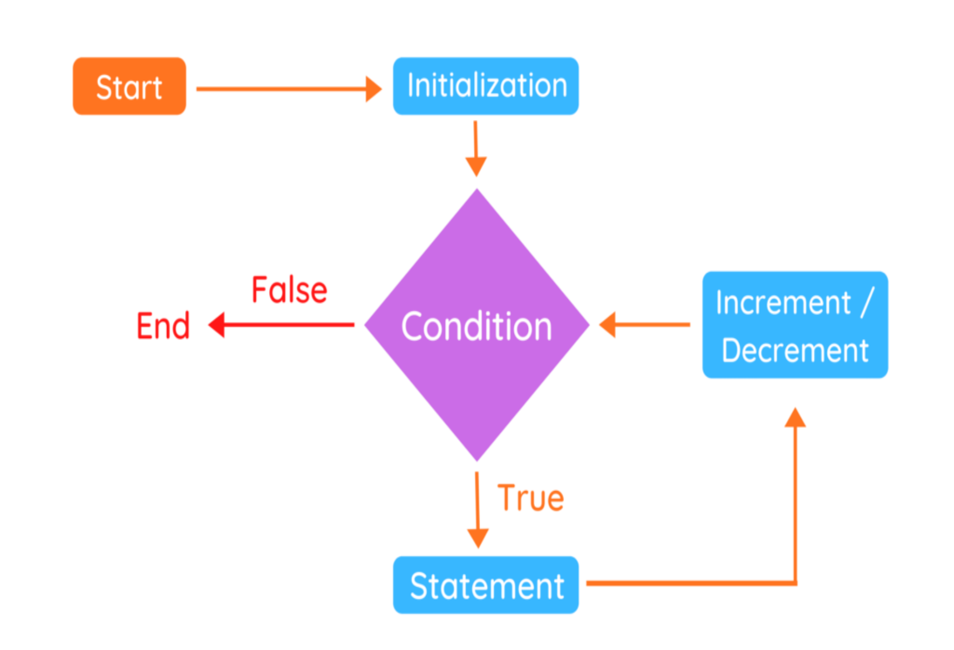


Image Source: https://images.app.goo.gl/h1FgsJh22U9bw7YR9

Looping

Loop Structure (Contd..)

- The following are the loop structures available in Python.
  While Statement
  For loop Statement
  Nested Loop Statement

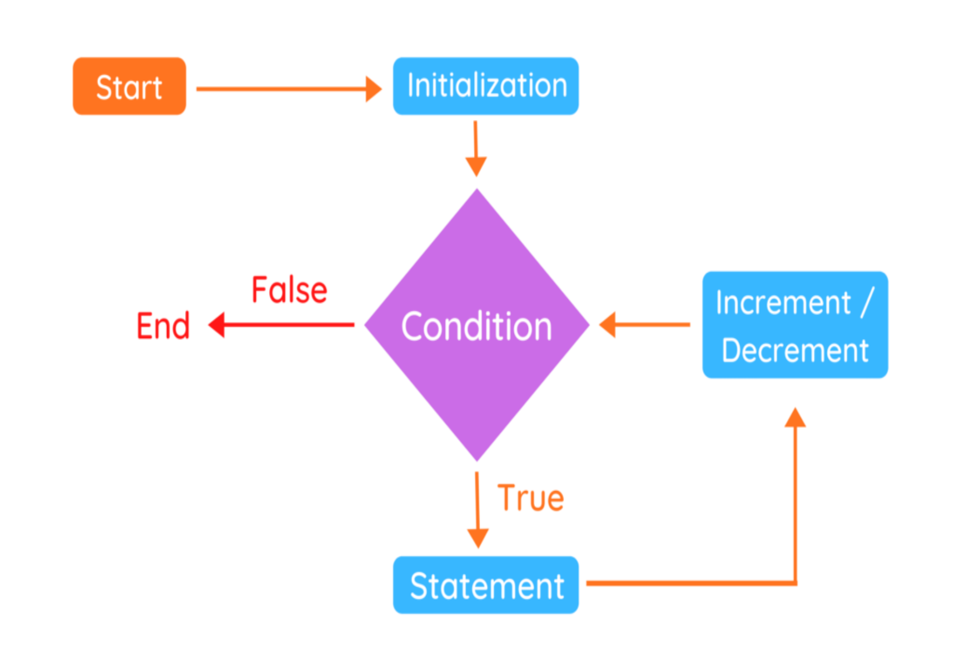


Image Source: https://images.app.goo.gl/h1FgsJh22U9bw7YR9

Looping

While Loop

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- It test the condition before executing the loop body.
- Syntax -

While expression:

        Statements(s)

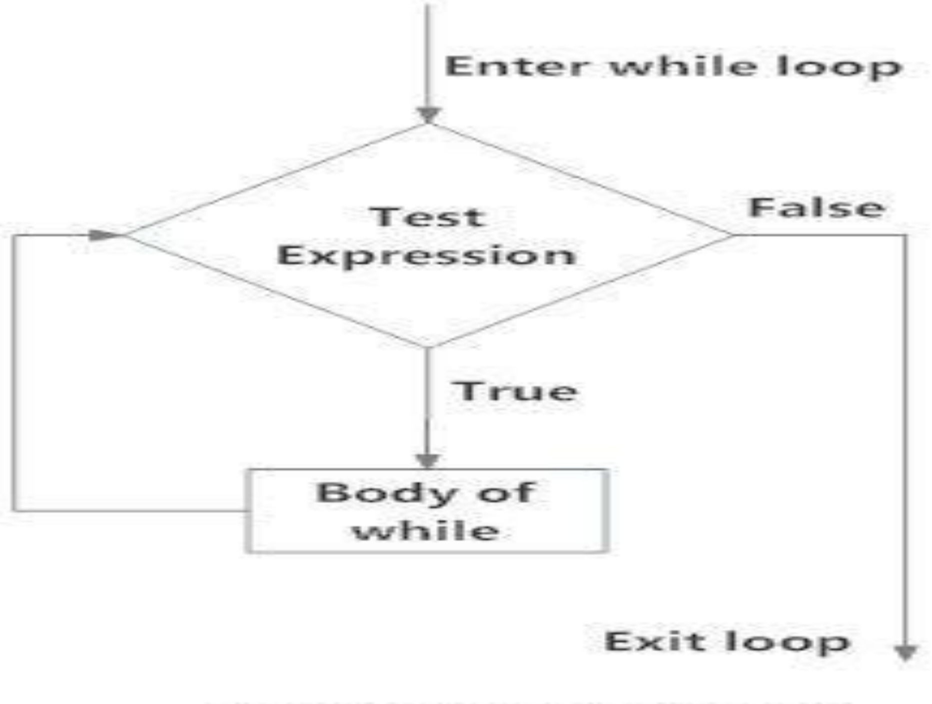


Fig: operation of while loop

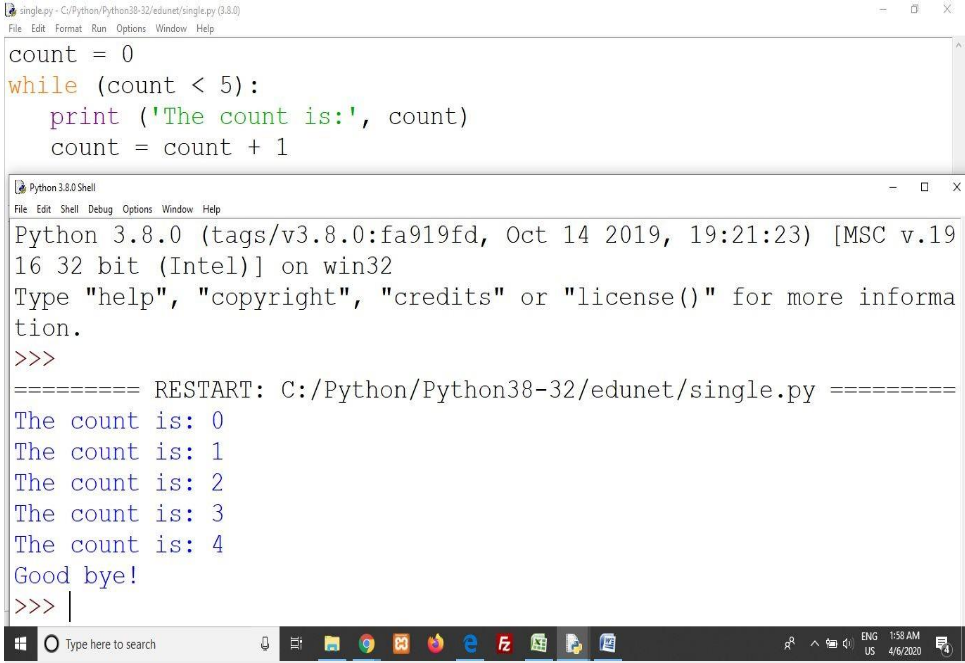
Image Source: https://images.app.goo.gl/TYVJXYyhmei968UR7

Looping

While Loop (Contd..)

- Example Program-

```
count=0
while(count<5):
    print(“The count is:”,count)
    count=count+1
```


single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG US
1:58 AM
4/6/2020

```
count = 0
while (count < 5):
   print ('The count is:', count)
   count = count + 1
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>> 
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
Good bye!
>>> 
```

Looping

For Loop

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- Syntax-

```
for iterating_var in sequence:
        statements(s)
```

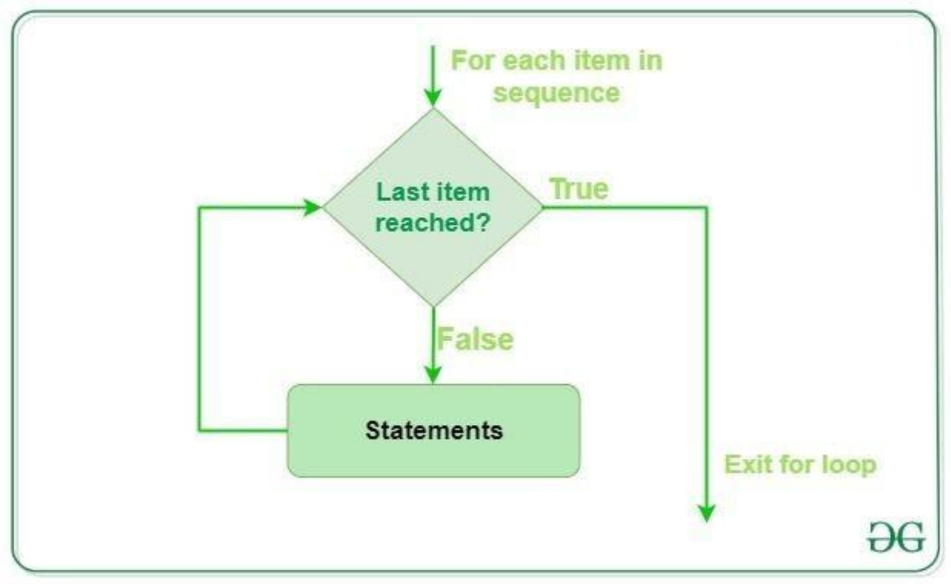


Image Source: https://images.app.goo.gl/jkyK2o4AqeCqvGmh7

Looping

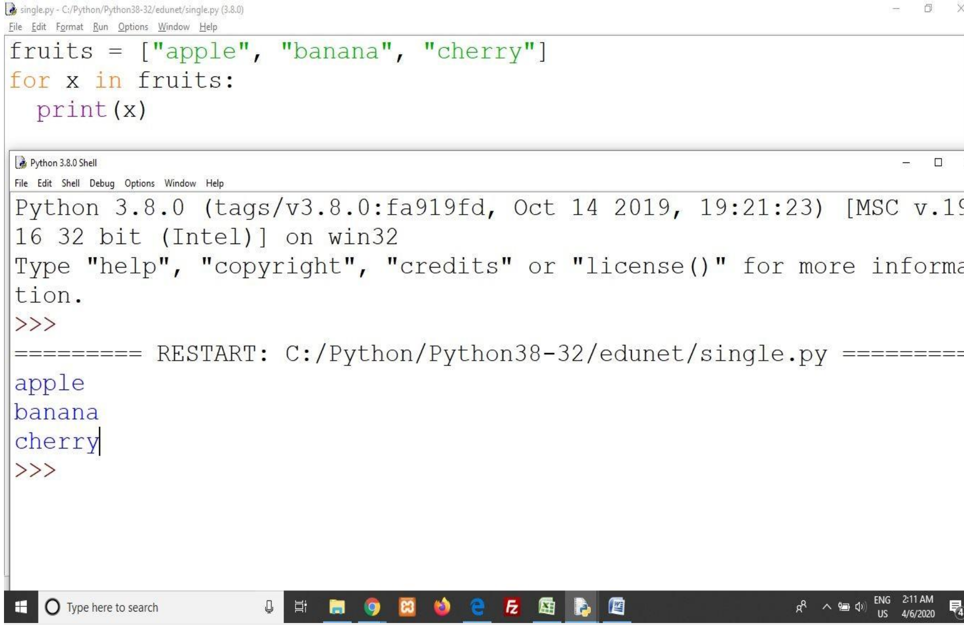
For Loop (Contd..)

- Example Program-

  fruits=[“apple”,”banana”,”cherry”]

  For x in fruits:

  print(x)



single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help

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

Python 3.8.0 Shell
File Edit Shell Debug Options Window Help

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
apple
banana
cherry
>>>
```

Type here to search
ENG
US
2:11 AM
4/6/2020

Looping

Nested Loop

- You can use one or more loop inside any another while, for or do..while loop.
- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop".

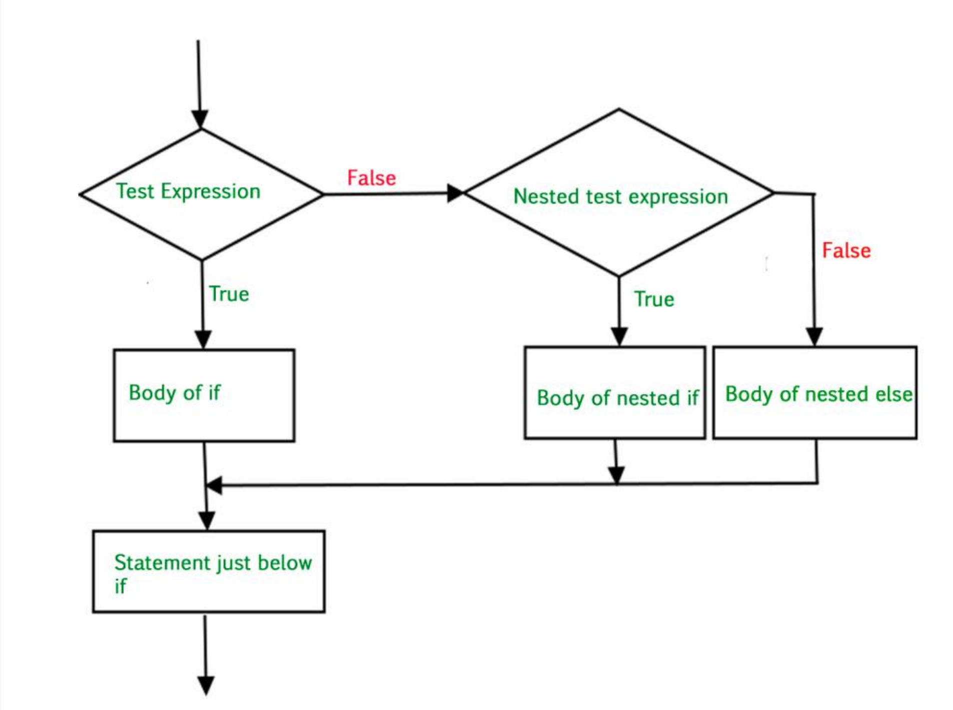


Image Source: https://images.app.goo.gl/4e1ecHdCRFYfqj6z7

Looping

Nested Loop (Contd..)

- Example Program-

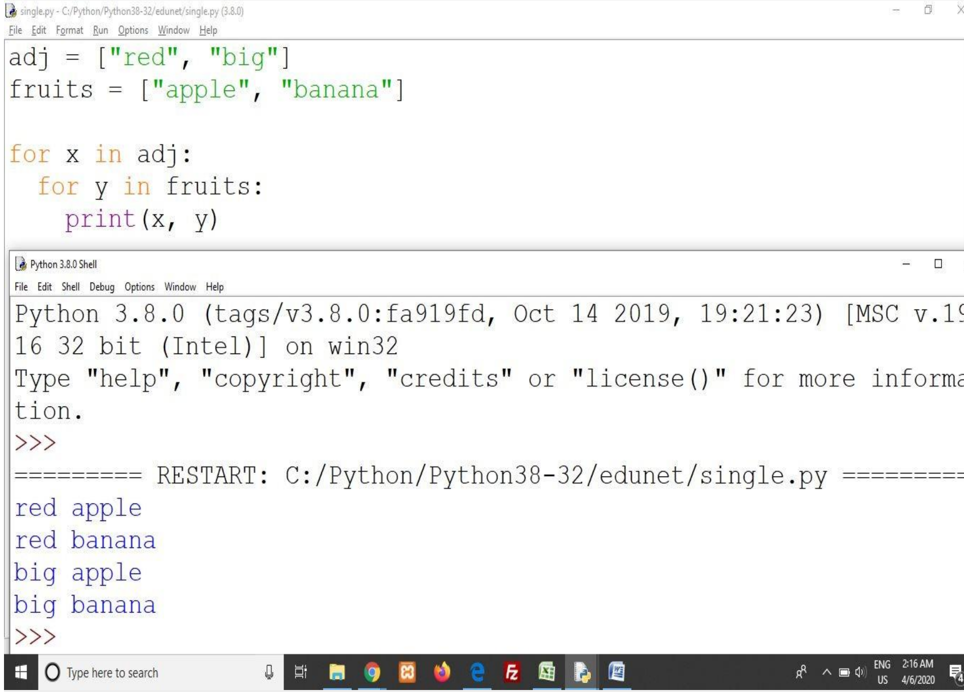
adj=[“red”,”big”]

fruits=[“apple”,”banana”]

For x in adj:

        For y in fruits:

                print(x,y)



single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search
ENG
US
2:16 AM
4/6/2020

```
adj = ["red", "big"]
fruits = ["apple", "banana"]

for x in adj:
  for y in fruits:
    print(x, y)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
red apple
red banana
big apple
big banana
>>>
```

Control Statement

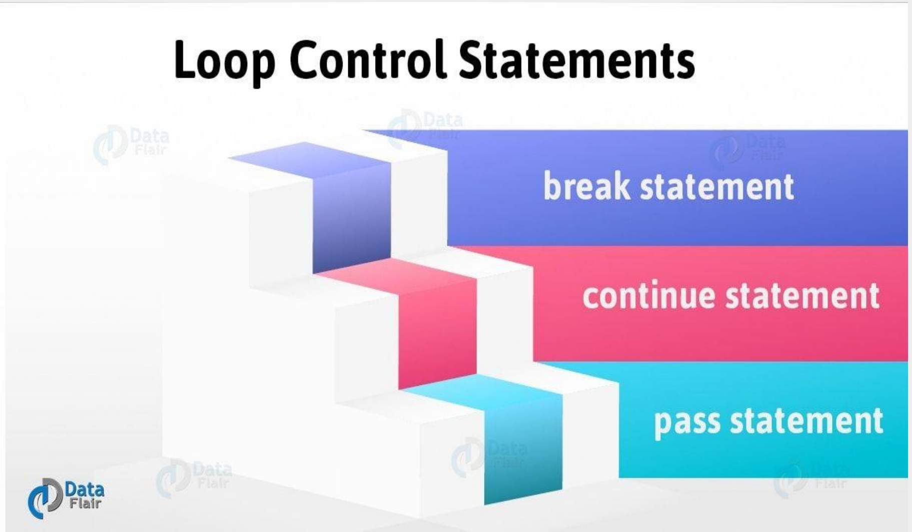


Image Source: https://images.app.goo.gl/naAsWZSy7bGzRCVo8

Control Statement

Loop Control Statement

- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed
- Python supports the following control statements.

  Break Statement
  Continue Statement
  Pass Statement

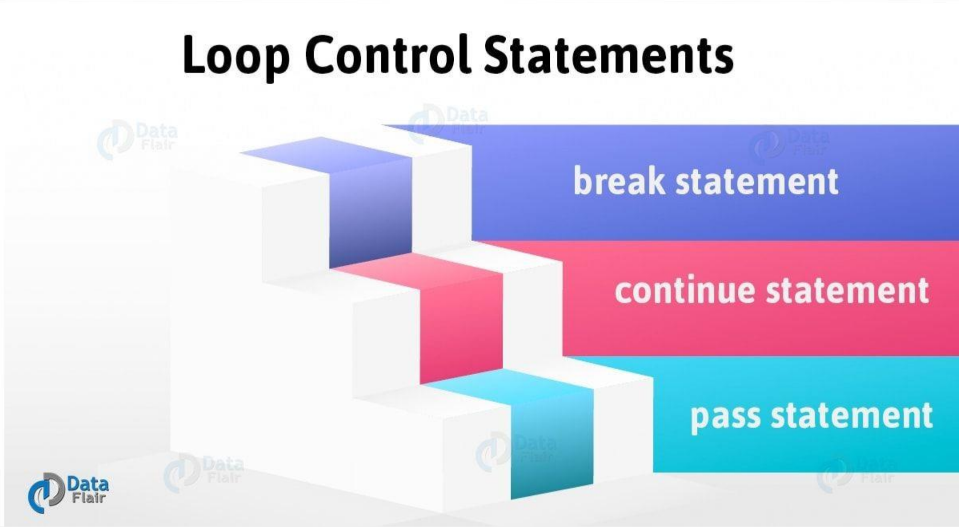


Image Source: https://images.app.goo.gl/naAsWZSy7bGzRCVo8

Control Statement

Break Statement

- Terminates the loop statement and transfers execution to the statement immediately following the loop.
- If break statement inside a nested loop (loop inside another loop), break will terminate the innermost loop.

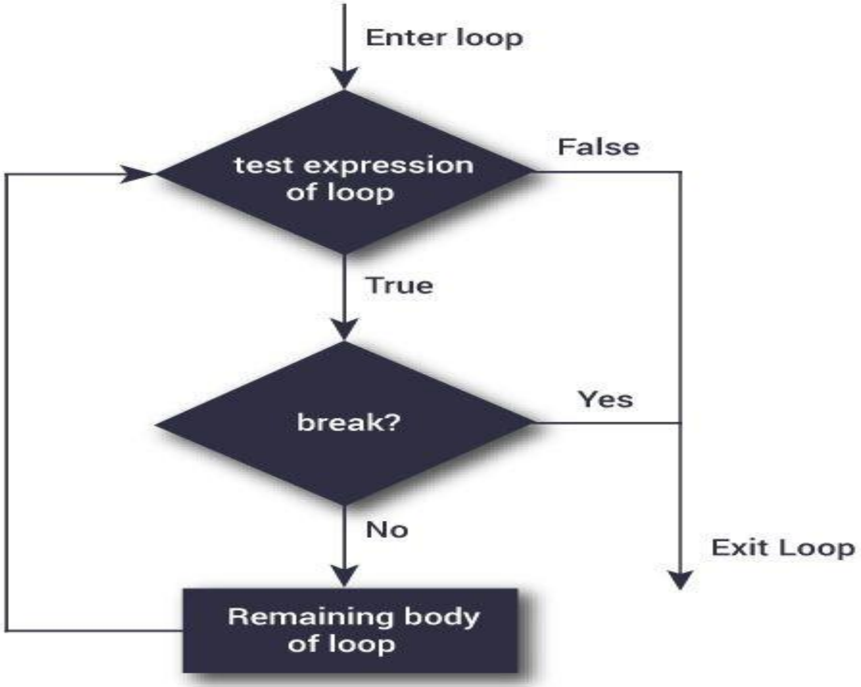


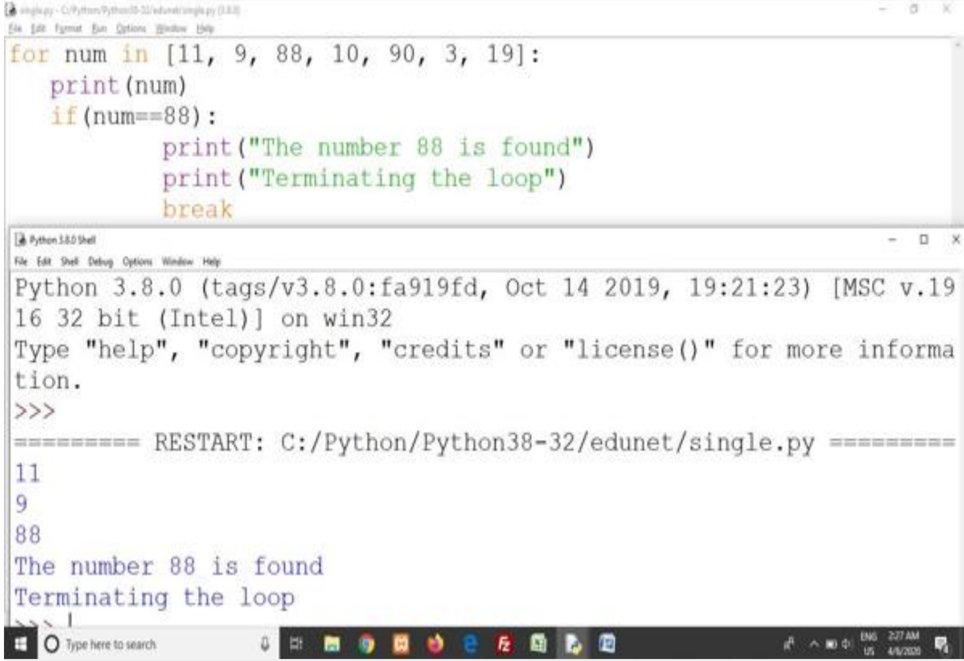
Image Source: https://images.app.goo.gl/JXcQJhxUkA1Z4qiy9

Control Statement

Break Statement (Contd..)

- Example Program-

```
For num in[11, 9, 88, 10, 90, 3, 19]:
    print(num)
    if(num==88):
        print(“The number 88 is found”)
        print(“Terminating the loop”)
        Break
```


single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search

```
for num in [11, 9, 88, 10, 90, 3, 19]:
    print(num)
    if(num==88):
            print("The number 88 is found")
            print("Terminating the loop")
            break
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
11
9
88
The number 88 is found
Terminating the loop
>>>
```

Control Statement

Continue Statement

- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- The continue statement can be used in both while and for loops.

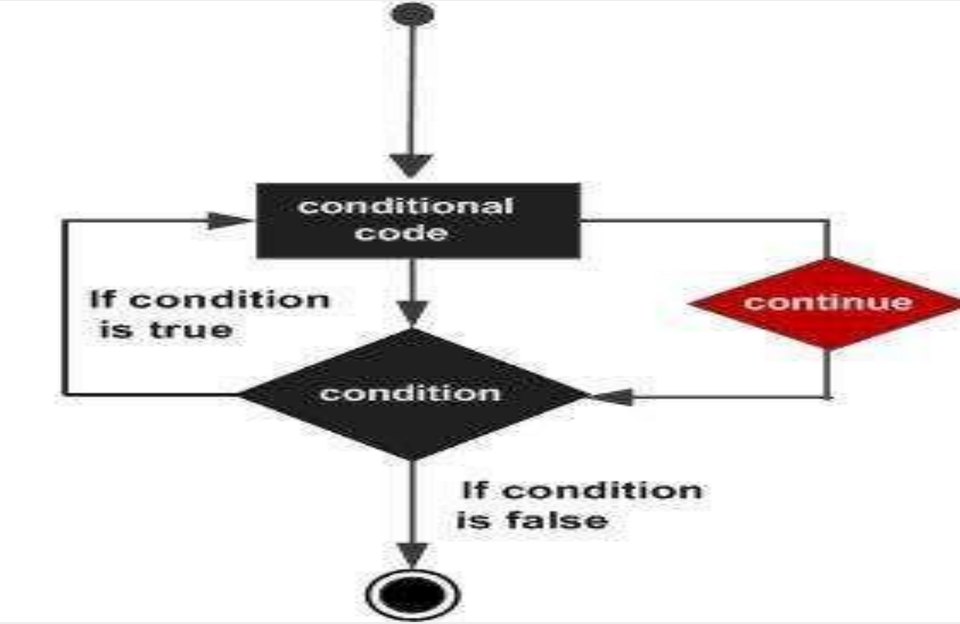


Image Source: https://images.app.goo.gl/LqptGF2grWpkdqWw6

Control Statement

Continue Statement (Contd..)

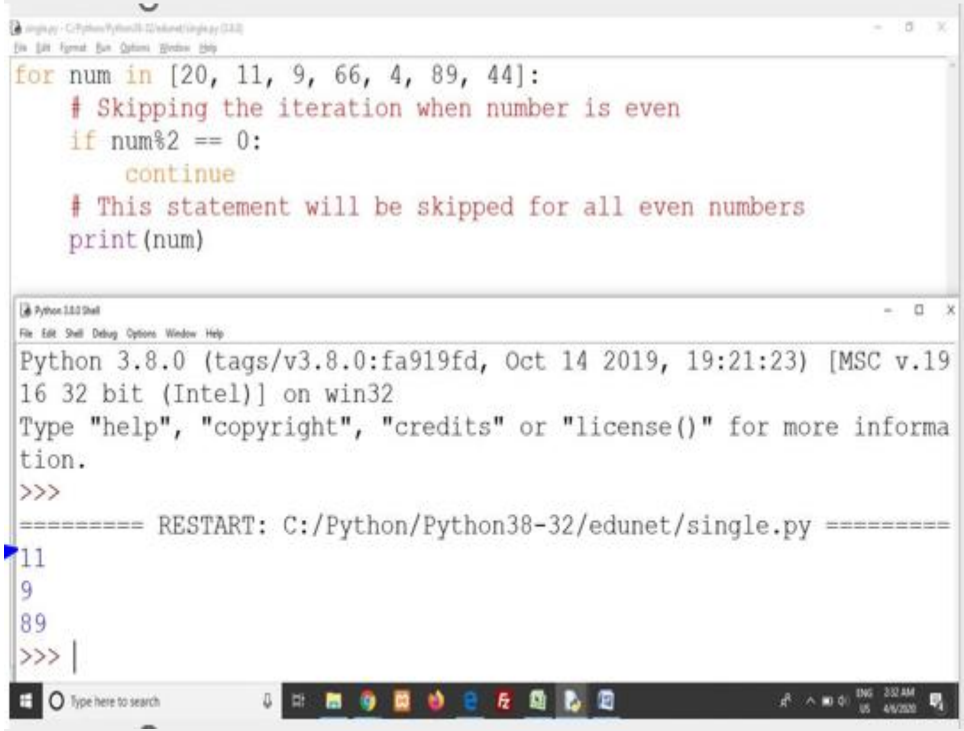
- Example Program-

- For num in[20, 11, 9, 66, 4, 89, 44]:

  If num%2==0:

  continue

  print(num)


File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search

```
for num in [20, 11, 9, 66, 4, 89, 44]:
    # Skipping the iteration when number is even
    if num%2 == 0:
        continue
    # This statement will be skipped for all even numbers
    print(num)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
11
9
89
>>> |
```

Control Statement

Pass Statement

- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- The interpreter does not ignore a pass statement, but nothing happens and the statement results into no operation.

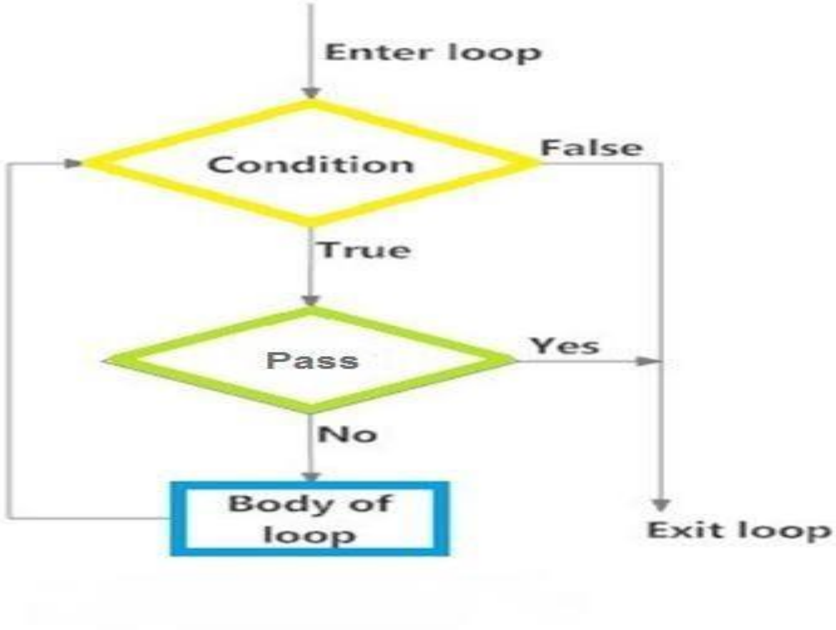


Image Source: https://images.app.goo.gl/Z9BnvaYpSLpGG6G86

Control Statement

Pass Statement (Contd..)

- Example Program-

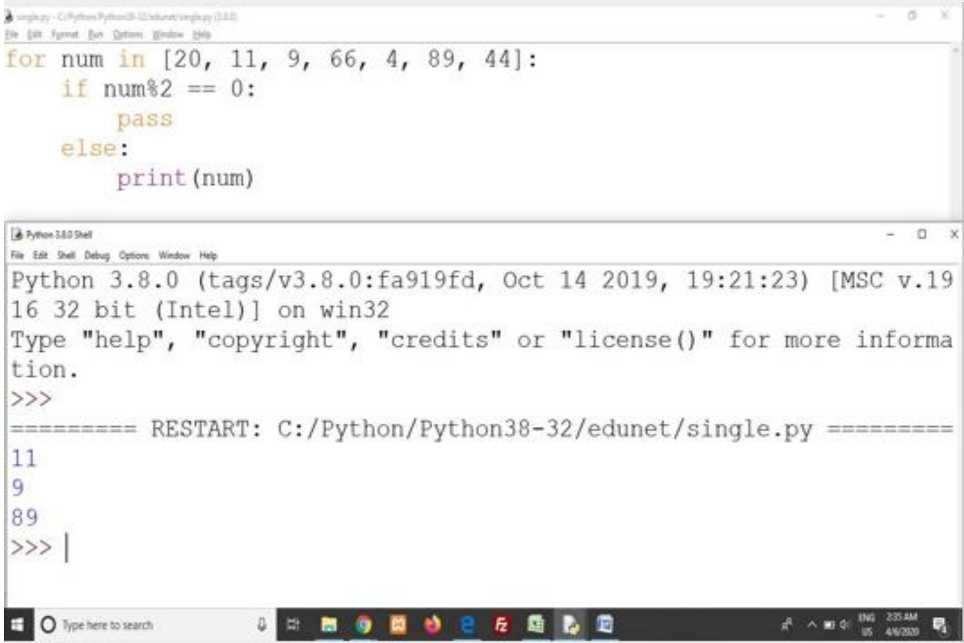
For num in [20, 11, 9, 66, 89, 44]:

    If num%2==0:

        Pass

    else:

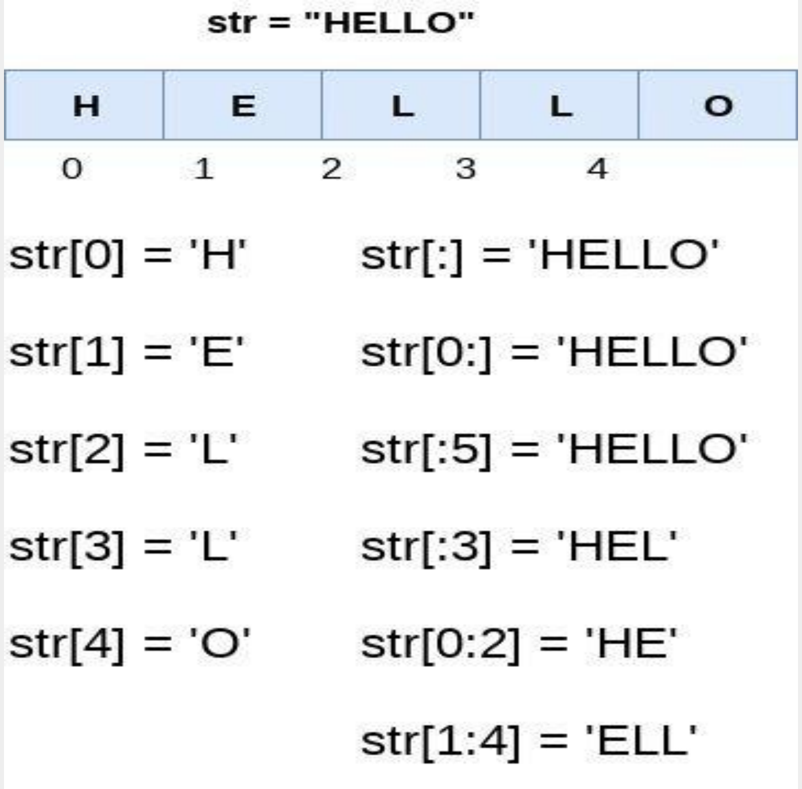
        print(num)


single.py - C:/Python/Python38-32/edunet/single.py (3.8.0)
File Edit Format Run Options Window Help
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Type here to search

```
for num in [20, 11, 9, 66, 4, 89, 44]:
    if num%2 == 0:
        pass
    else:
        print(num)
```

```
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.19
16 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more informa
tion.
>>>
========= RESTART: C:/Python/Python38-32/edunet/single.py =========
11
9
89
>>> |
```

String Manipulation



str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

str[:] = 'HELLO'

str[0:] = 'HELLO'

str[:5] = 'HELLO'

str[:3] = 'HEL'

str[0:2] = 'HE'

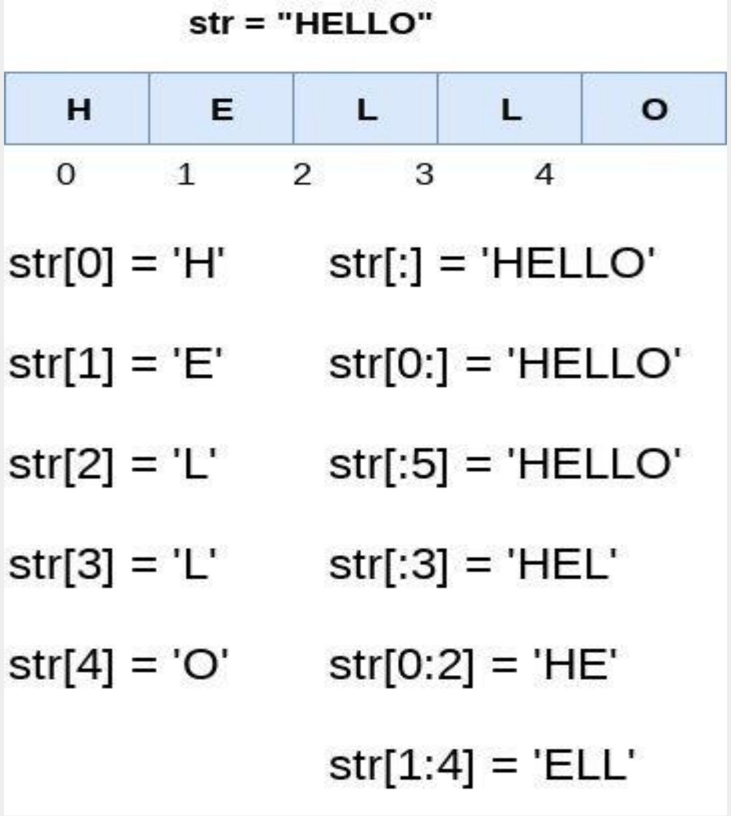
str[1:4] = 'ELL'

Image Source: https://static.javatpoint.com/python/images/strings-indexing-and-splitting2.png

String Manipulation

Introduction

- Python string is an ordered collection of characters which is used to represent and store the text-based information.
- Strings are stored as individual characters in a contiguous memory location.
- It can be accessed from both directions: forward and backward.
- Characters are nothing but symbols.
- Strings are immutable, which means that once a string is created, they cannot be changed.



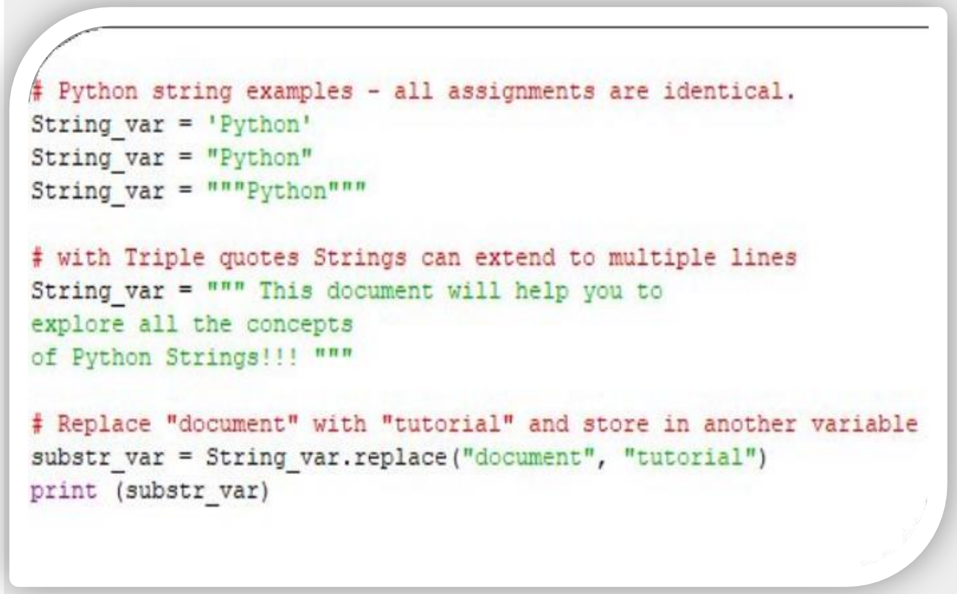
```
str = "HELLO"

str[0] = 'H'        str[:] = 'HELLO'
str[1] = 'E'        str[0:] = 'HELLO'
str[2] = 'L'        str[:5] = 'HELLO'
str[3] = 'L'        str[:3] = 'HEL'
str[4] = 'O'        str[0:2] = 'HE'
                    str[1:4] = 'ELL'
```

Image Source: https://static.javatpoint.com/python/images/strings-indexing-and-splitting2.png

String Manipulation

Create Strings

- Strings can be created by enclosing characters inside a single quote or double-quotes.
- Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.



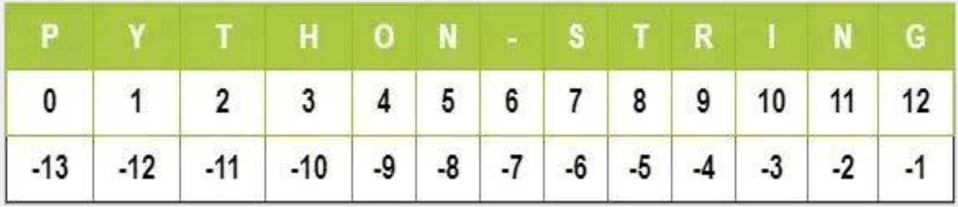
```
# Python string examples - all assignments are identical.
String_var = 'Python'
String_var = "Python"
String_var = """Python"""

# with Triple quotes Strings can extend to multiple lines
String_var = """ This document will help you to
explore all the concepts
of Python Strings!!! """

# Replace "document" with "tutorial" and store in another variable
substr_var = String_var.replace("document", "tutorial")
print (substr_var)
```

String Manipulation

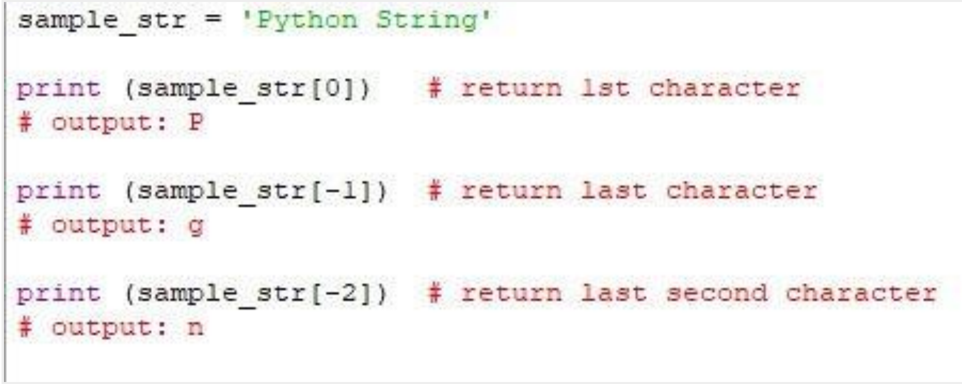
Index and Slice - Indexing

Python allows to index from the 0th position in Strings. But it also supports negative indexes.

- Index of ‘-1’ represents the last character of the String.
- Similarly, using ‘-2’, we can access the penultimate element of the string and so on.



P	Y	T	H	O	N	-	S	T	R	I	N	G
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1



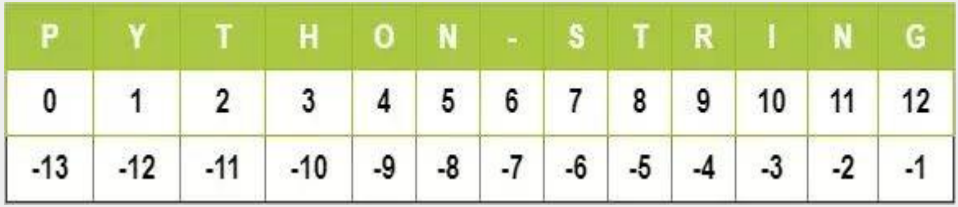
```
sample_str = 'Python String'

print (sample_str[0])   # return 1st character
# output: P

print (sample_str[-1])  # return last character
# output: g

print (sample_str[-2])  # return last second character
# output: n
```

Image Source : [https://cdn.techbeamers.com/wp-content/uploads/2016/03/String-Representation-in- Python.png](https://cdn.techbeamers.com/wp-content/uploads/2016/03/String-Representation-in-Python.png)

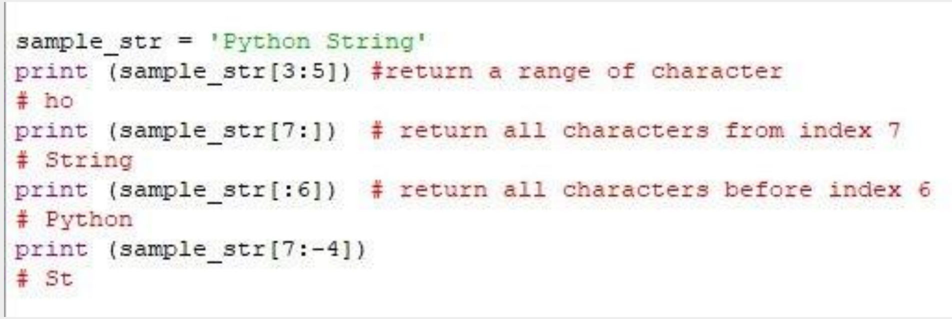
String Manipulation

Index and Slice - Slicing

- To retrieve a range of characters in a String, we use 'slicing operator,' the colon ':' sign.
- With the slicing operator, we define the range as [a:b].
- Let us print all the characters of the String starting from index 'a' up to char at index 'b-1'. So the char at index 'b' is not a part of the output.



P	Y	T	H	O	N	-	S	T	R	I	N	G
0	1	2	3	4	5	6	7	8	9	10	11	12
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1



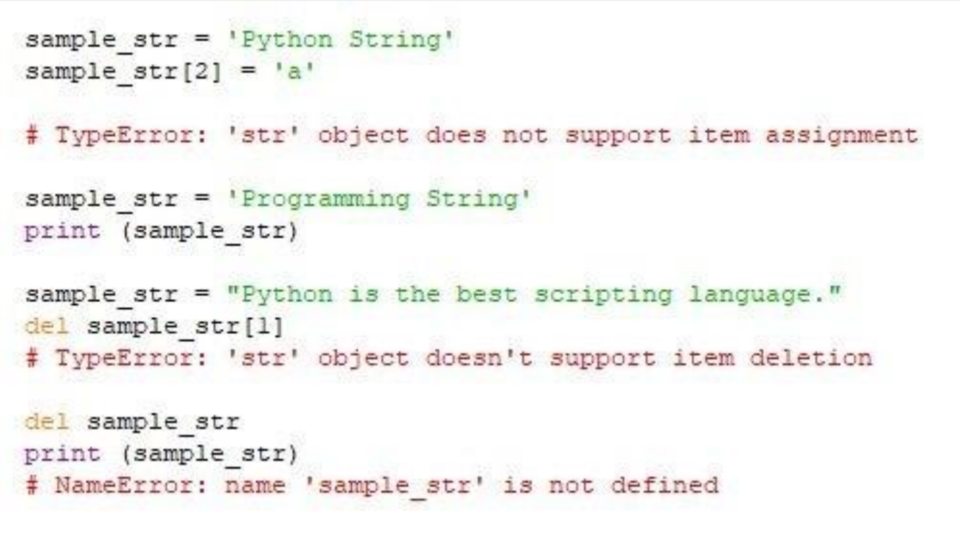
```
sample_str = 'Python String'
print (sample_str[3:5]) #return a range of character
# ho
print (sample_str[7:])  # return all characters from index 7
# String
print (sample_str[:6])  # return all characters before index 6
# Python
print (sample_str[7:-4])
# St
```

Image Source : https://cdn.techbeamers.com/wp-content/uploads/2016/03/String-Representation-in- Python.png

String Manipulation

Modify/Delete Strings

- Python Strings are by design immutable. It suggests that once a String binds to a variable, it can’t be modified.
- If you want to update the String, then re-assign a new String value to the same variable.
- We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

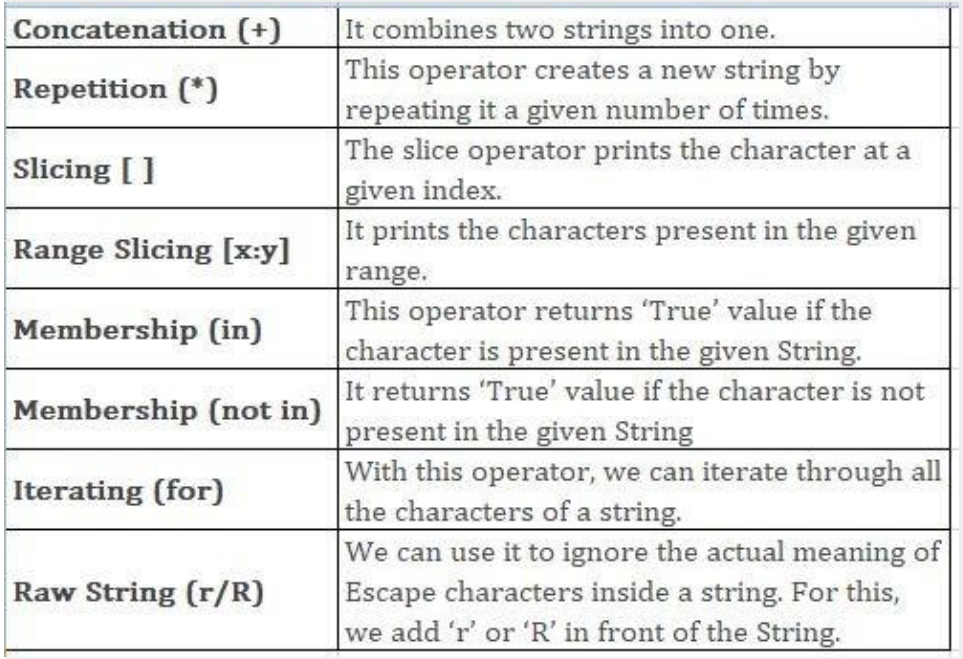


```
sample_str = 'Python String'
sample_str[2] = 'a'

# TypeError: 'str' object does not support item assignment

sample_str = 'Programming String'
print (sample_str)

sample_str = "Python is the best scripting language."
del sample_str[1]
# TypeError: 'str' object doesn't support item deletion

del sample_str
print (sample_str)
# NameError: name 'sample_str' is not defined
```

String Manipulation

String Operators

- There are many operations that can be performed with string



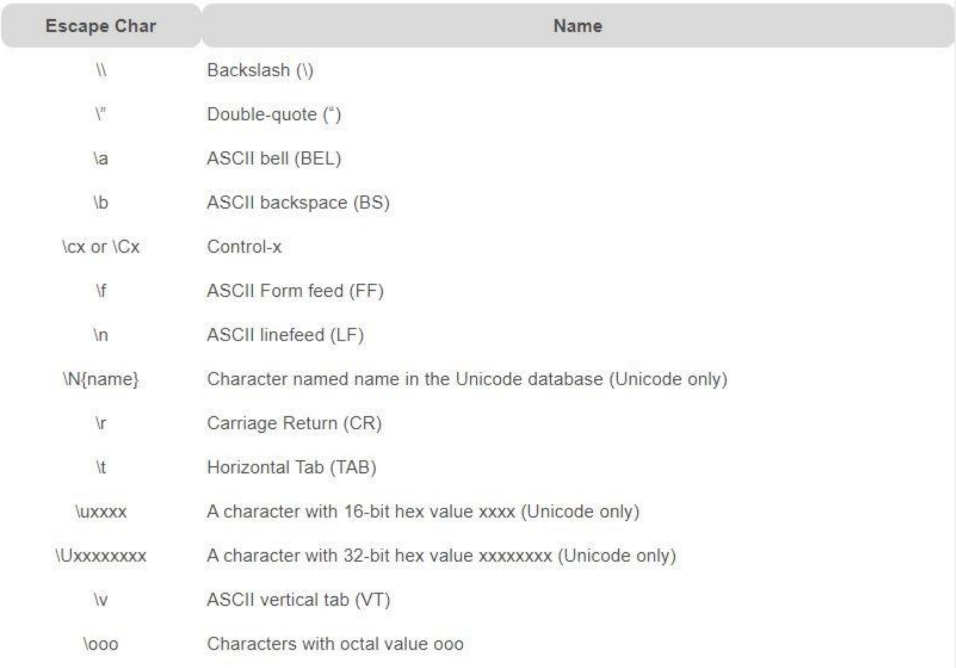
Concatenation (+)	It combines two strings into one.
Repetition (*)	This operator creates a new string by repeating it a given number of times.
Slicing [ ]	The slice operator prints the character at a given index.
Range Slicing [x:y]	It prints the characters present in the given range.
Membership (in)	This operator returns 'True' value if the character is present in the given String.
Membership (not in)	It returns 'True' value if the character is not present in the given String
Iterating (for)	With this operator, we can iterate through all the characters of a string.
Raw String (r/R)	We can use it to ignore the actual meaning of Escape characters inside a string. For this, we add 'r' or 'R' in front of the String.

Image Source : https://www.programiz.com/python-programming

String Manipulation

String Formatting Operators

- An Escape sequence starts with a backslash (\), which signals the compiler to treat it differently.
- Python subsystem automatically interprets an escape sequence irrespective of it is in a single-quoted or double-quoted Strings.



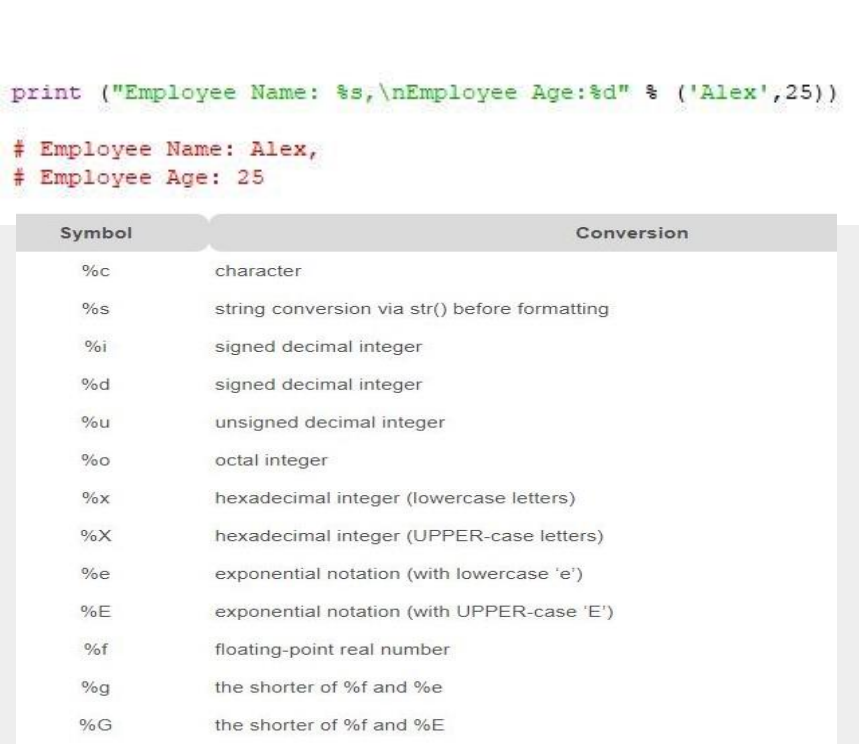
Escape Char	Name
\\	Backslash (\)
\"	Double-quote (“)
\a	ASCII bell (BEL)
\b	ASCII backspace (BS)
\cx or \Cx	Control-x
\f	ASCII Form feed (FF)
\n	ASCII linefeed (LF)
\N{name}	Character named name in the Unicode database (Unicode only)
\r	Carriage Return (CR)
\t	Horizontal Tab (TAB)
\uxxxx	A character with 16-bit hex value xxxx (Unicode only)
\Uxxxxxxxx	A character with 32-bit hex value xxxxxxxx (Unicode only)
\v	ASCII vertical tab (VT)
\ooo	Characters with octal value ooo

Image Source:
https://www.techbeamers.com/python-strings-functions-and-examples/#strin_g-formatting-operators-in-python

String Manipulation

String Formatting Operators

- Python Format Characters
- String ‘%’ operator issued for formatting Strings. We often use this operator with the print() function.



```
print ("Employee Name: %s,\nEmployee Age:%d" % ('Alex',25))

# Employee Name: Alex,
# Employee Age: 25
```

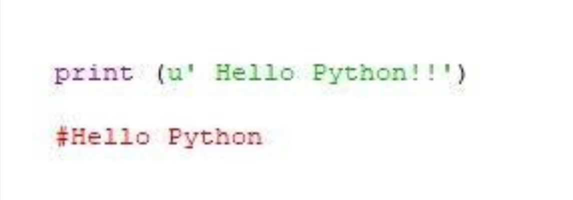
Symbol	Conversion
%c	character
%s	string conversion via str() before formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPER-case letters)
%e	exponential notation (with lowercase ‘e’)
%E	exponential notation (with UPPER-case ‘E’)
%f	floating-point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Image Source:
https://www.techbeamers.com/python-strings-functions-and-examples/#strin_g-formatting-operators-in-python

String Manipulation

Unicode String support

- Regular Strings stores as the 8-bit ASCII value, whereas Unicode String follows the 16-bit ASCII standard.
- This extension allows the strings to include characters from the different languages of the world.
- In Python, the letter ‘u’ works as a prefix to distinguish between Unicode and usual strings.



```
print (u' Hello Python!!')

#Hello Python
```

String Manipulation

Built-in String Functions

- Conversion Functions
- Comparison Functions
- Padding Functions
- Search Functions
- String Substitution Functions
- Misc String Functions

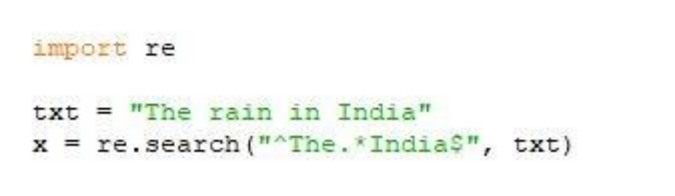
For more information and all functions check this link

https://docs.python.org/2/library/string.html

String Manipulation

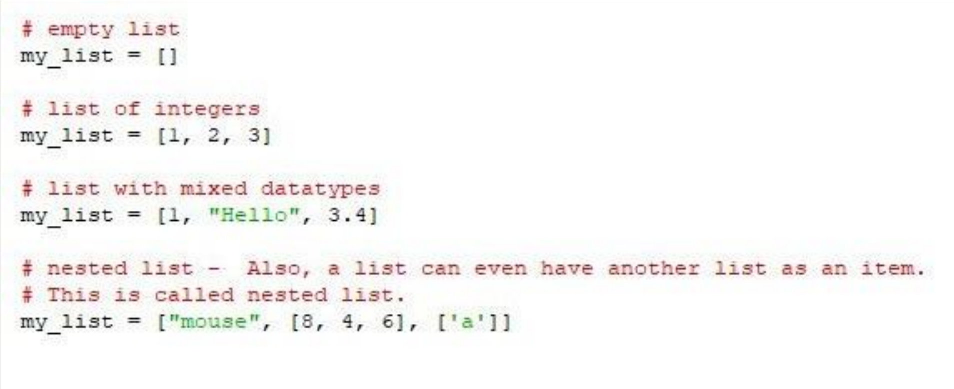
Regular Expressions

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.
- Python has a built-in package called re, which can be used to work with Regular Expressions.



```
import re

txt = "The rain in India"
x = re.search("^The.*India$", txt)
```

List

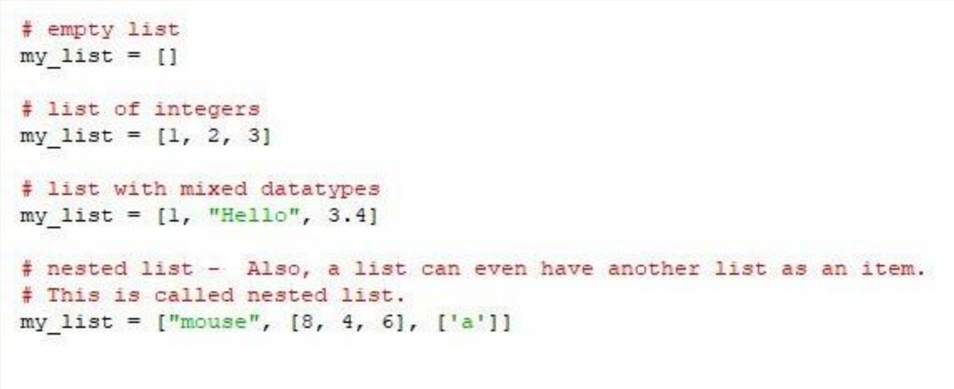


```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]

# nested list -  Also, a list can even have another list as an item.
# This is called nested list.
my_list = ["mouse", [8, 4, 6], ['a']]
```

List

Python List & Create List

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

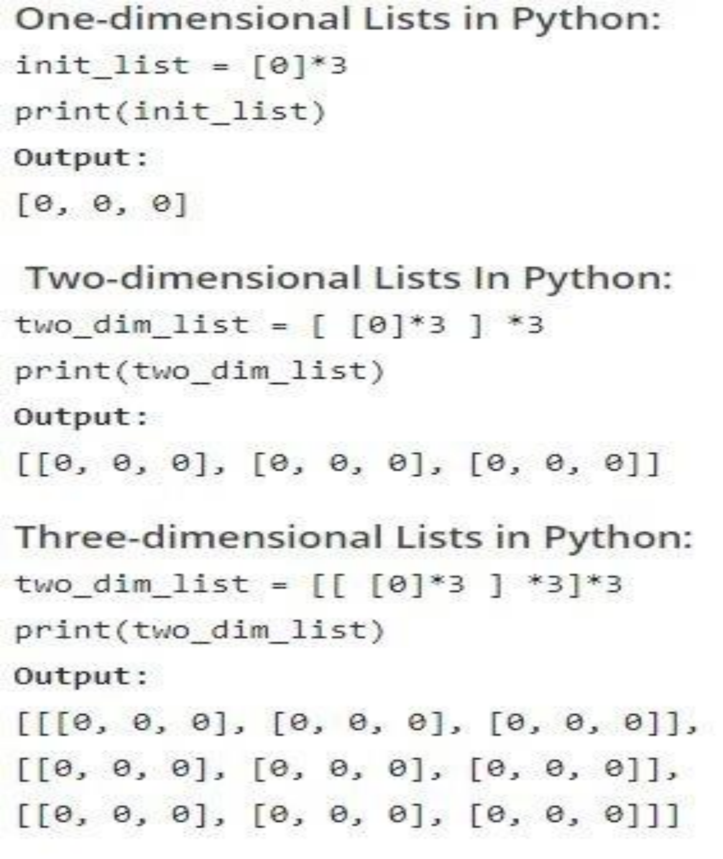


```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]

# nested list -  Also, a list can even have another list as an item.
# This is called nested list.
my_list = ["mouse", [8, 4, 6], ['a']]
```

List

Creating Multi-dimensional Lists

- A list can hold other lists as well which can result in multi-dimensional lists.



One-dimensional Lists in Python:

```
init_list = [0]*3
print(init_list)
Output:
[0, 0, 0]
```

Two-dimensional Lists In Python:

```
two_dim_list = [ [0]*3 ] *3
print(two_dim_list)
Output:
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

Three-dimensional Lists in Python:

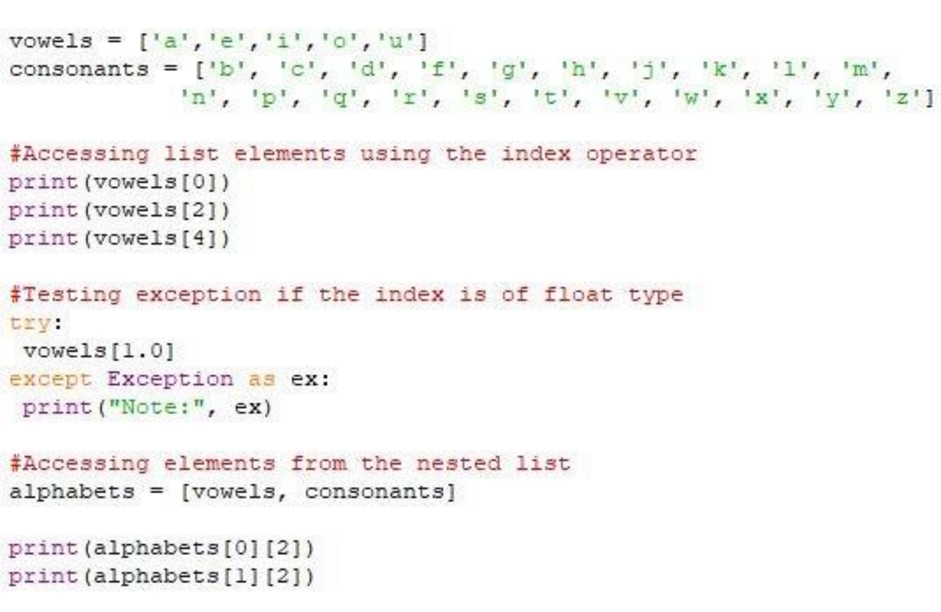
```
two_dim_list = [[ [0]*3 ] *3]*3
print(two_dim_list)
Output:
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]]
```

Image Source : https://intellipaat.com/blog/tutorial/python-tutorial/python-lists/

List

Access elements from a list List indexing

- Index operator : The simplest one is to use the index operator ([ ]) to access an element from the list. Since the list has zero as the first index, so a list of size ten will have indices from 0 to 9.
- Any attempt to access an item beyond this range would result in an IndexError. The index is always an integer.
- Using any other type of value will lead to TypeError.

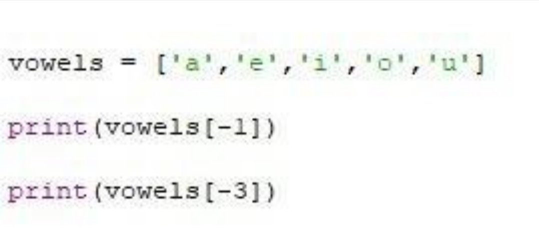


```
vowels = ['a','e','i','o','u']
consonants = ['b', 'c', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm',
              'n', 'p', 'q', 'r', 's', 't', 'v', 'w', 'x', 'y', 'z']

#Accessing list elements using the index operator
print(vowels[0])
print(vowels[2])
print(vowels[4])

#Testing exception if the index is of float type
try:
 vowels[1.0]
except Exception as ex:
 print("Note:", ex)

#Accessing elements from the nested list
alphabets = [vowels, consonants]

print(alphabets[0][2])
print(alphabets[1][2])
```

List

Access elements from a list Reverse/Negative indexing

- Reverse indexing : Python enables reverse (Negative) indexing for the sequence data type. So, for the Python list to index in the opposite order, you need to set the index using the minus (-) sign. Indexing the list with “-1” will return the last element of the list, -2 the second last and so on.

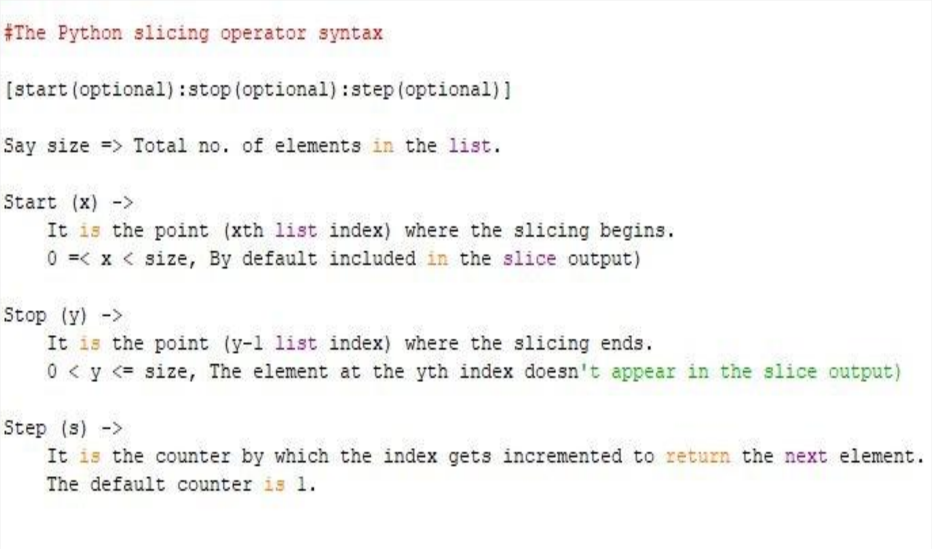


```
vowels = ['a','e','i','o','u']

print(vowels[-1])

print(vowels[-3])
```

List

List slicing

- Python comes with a magical slice operator which returns the part of a sequence.
- It operates on objects of different data types such as strings, tuples, and works the same on a Python list.

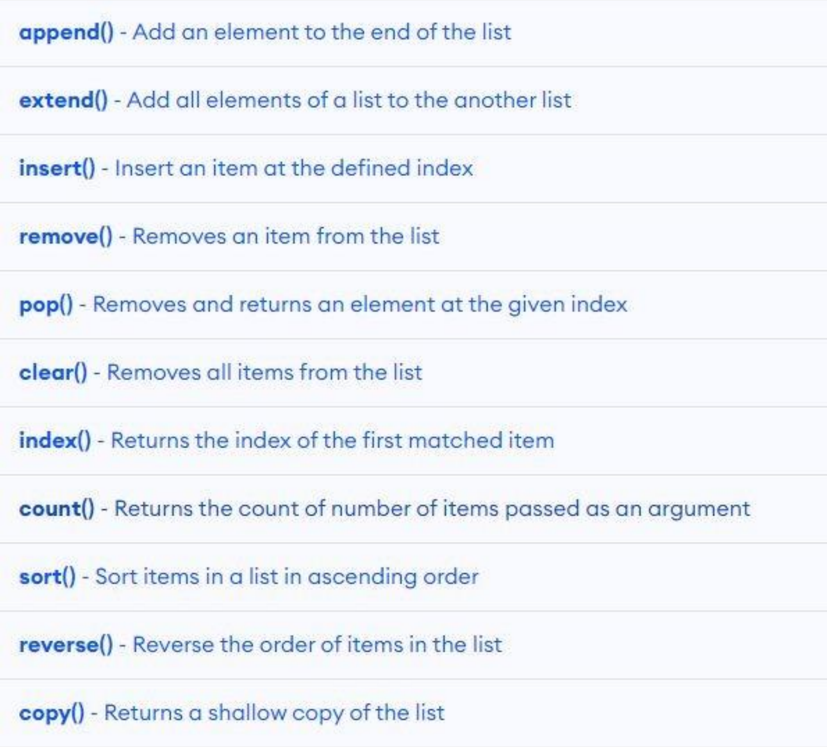


```
#The Python slicing operator syntax

[start(optional):stop(optional):step(optional)]

Say size => Total no. of elements in the list.

Start (x) ->
    It is the point (xth list index) where the slicing begins.
    0 =< x < size, By default included in the slice output)

Stop (y) ->
    It is the point (y-1 list index) where the slicing ends.
    0 < y <= size, The element at the yth index doesn't appear in the slice output)

Step (s) ->
    It is the counter by which the index gets incremented to return the next element.
    The default counter is 1.
```

List

Change or Add elements

- List are mutable, meaning, their elements can be changed unlike string or tuple.
- We can use assignment operator (=) to change an item or a range of items.
- We can also use + operator to combine two lists. This is also called concatenation.
- The * operator repeats a list for the given number of times.



append() - Add an element to the end of the list

extend() - Add all elements of a list to the another list

insert() - Insert an item at the defined index

remove() - Removes an item from the list

pop() - Removes and returns an element at the given index

clear() - Removes all items from the list

index() - Returns the index of the first matched item

count() - Returns the count of number of items passed as an argument

sort() - Sort items in a list in ascending order

reverse() - Reverse the order of items in the list

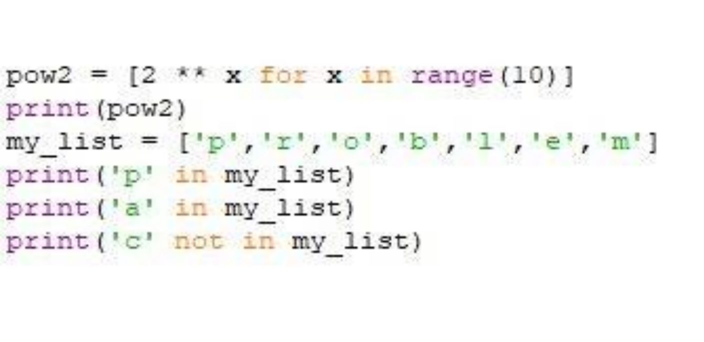
copy() - Returns a shallow copy of the list

Image Source: https://www.programiz.com/python-programming

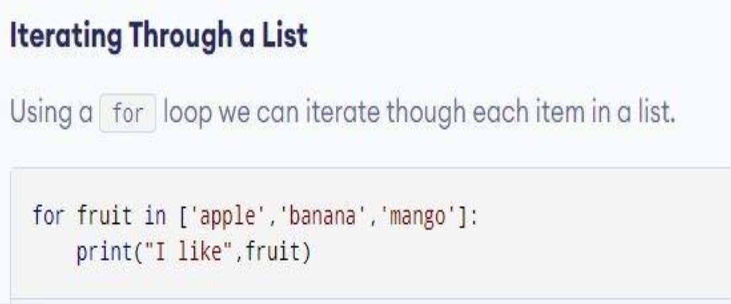
List

Elegant way to create new List

- List comprehension is an elegant and concise way to create a new list from an existing list in Python.
- List comprehension consists of an expression followed by for statement inside square brackets.
- A list comprehension can optionally contain more for or if statements.
- An optional if statement can filter out items for the new list.
- We can test if an item exists in a list or not, using the keyword in.



```
pow2 = [2 ** x for x in range(10)]
print(pow2)
my_list = ['p','r','o','b','l','e','m']
print('p' in my_list)
print('a' in my_list)
print('c' not in my_list)
```



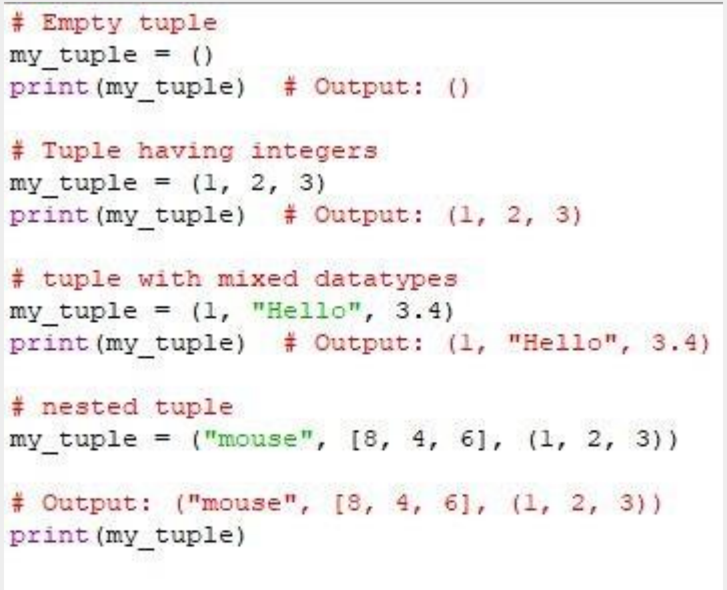
Iterating Through a List

Using a `for` loop we can iterate though each item in a list.

```
for fruit in ['apple','banana','mango']:
    print("I like",fruit)
```

Image Source: https://www.programiz.com/python-programming

Tuple

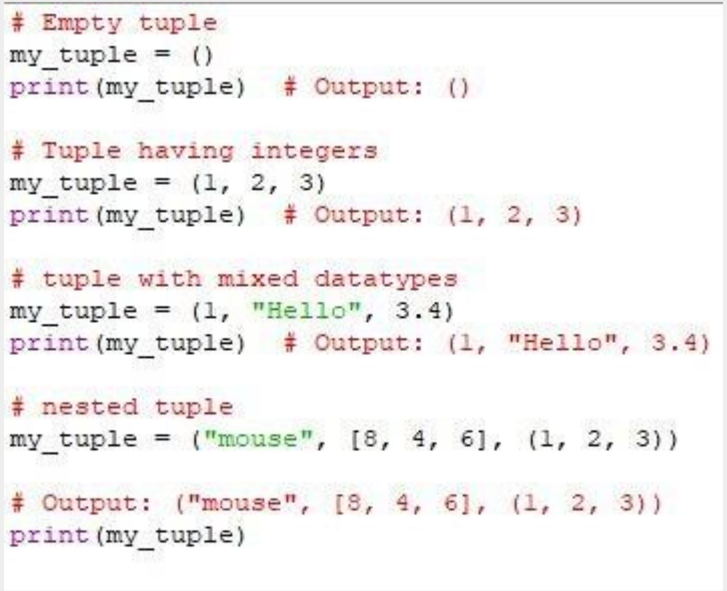


```
# Empty tuple
my_tuple = ()
print(my_tuple)  # Output: ()

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)  # Output: (1, 2, 3)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)  # Output: (1, "Hello", 3.4)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# Output: ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Tuple

Tuple & Create Tuple

- A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas, in a list, elements can be changed.
- A tuple is created by placing all the items (elements) inside parentheses (), separated by commas.
- A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

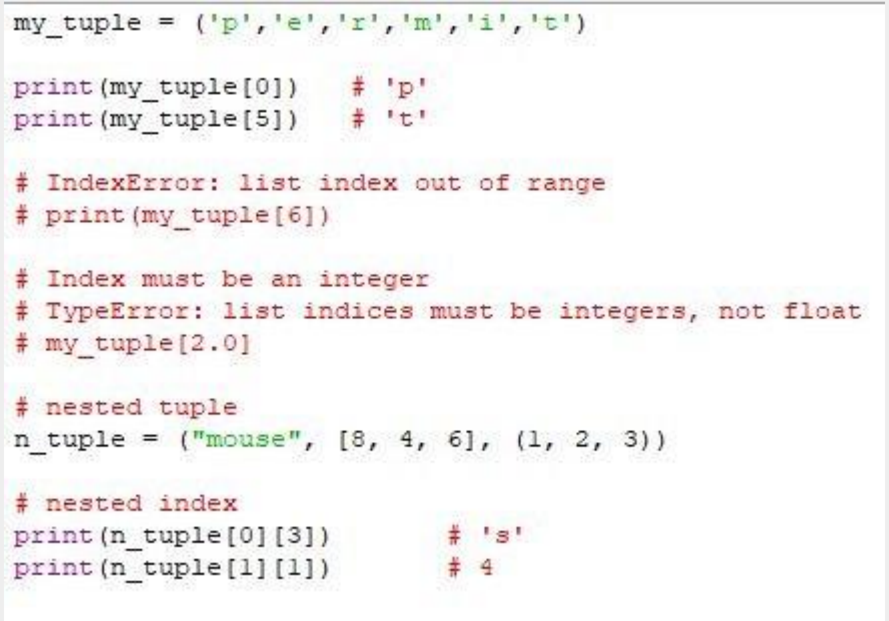


```
# Empty tuple
my_tuple = ()
print(my_tuple)  # Output: ()

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)  # Output: (1, 2, 3)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)  # Output: (1, "Hello", 3.4)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# Output: ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

Tuple

Access Tuple Elements

- Indexing - We can use the index operator [] to access an item in a tuple where the index starts from 0.
- So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an element outside of tuple (for example, 6, 7,...) will raise an IndexError.
- The index must be an integer; so we cannot use float or other types. This will result in TypeError.

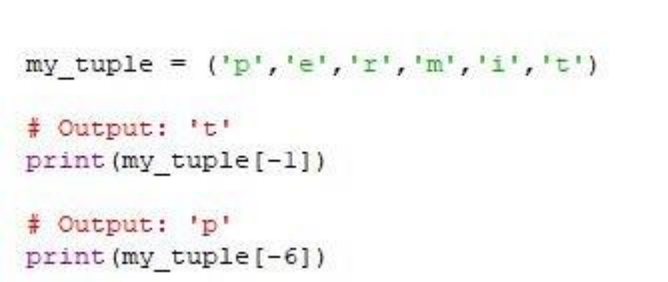


```
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'
print(my_tuple[5])   # 't'

# IndexError: list index out of range
# print(my_tuple[6])

# Index must be an integer
# TypeError: list indices must be integers, not float
# my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])       # 's'
print(n_tuple[1][1])       # 4
```

Tuple

Access Tuple Elements

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

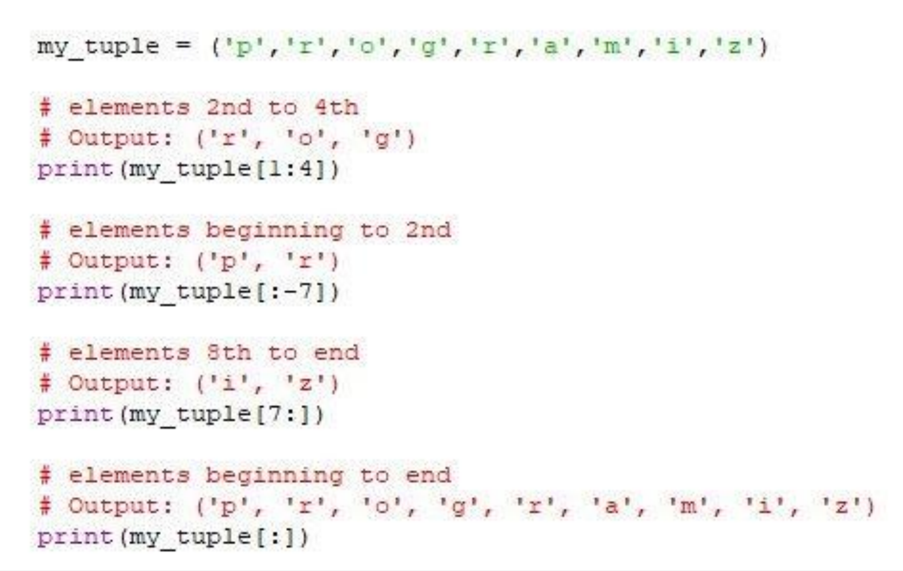


```
my_tuple = ('p','e','r','m','i','t')

# Output: 't'
print(my_tuple[-1])

# Output: 'p'
print(my_tuple[-6])
```

Tuple

Access Tuple Elements

- We can access a range of items in a tuple by using the slicing operator - colon ":".
- Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.

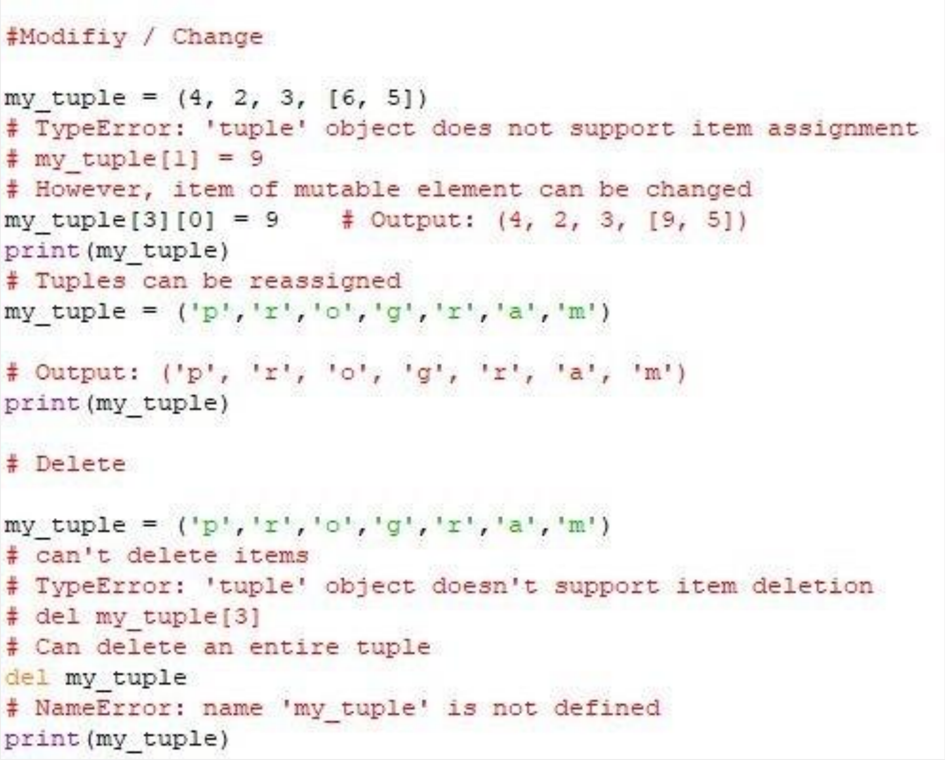


```
my_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:-7])

# elements 8th to end
# Output: ('i', 'z')
print(my_tuple[7:])

# elements beginning to end
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
print(my_tuple[:])
```

Tuple

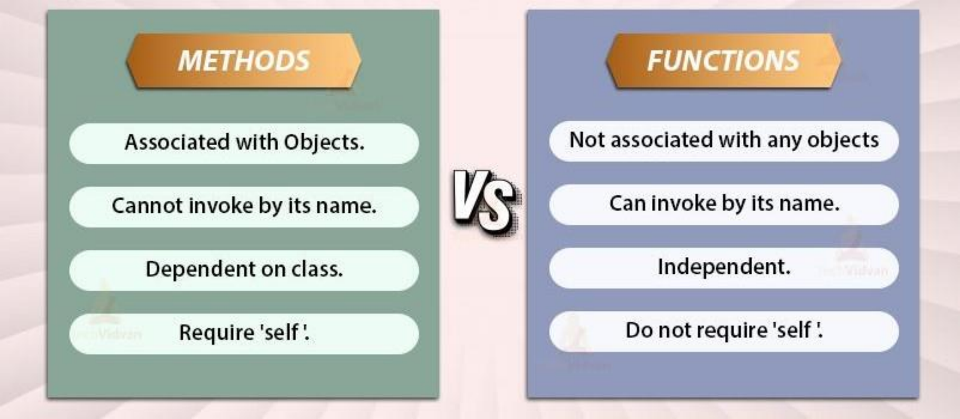
Performing Operations Modifying, Deleting

- Tuple cannot be changed once it has
- been assigned.
- the element is itself a mutable datatype like list, its nested items can be changed.
- + operator to combine two tuples
- repeat the elements in a tuple for a given number of times using
- the * operator.
- Cannot delete or remove items from a tuple.
- Deleting a tuple entirely is possible



```
#Modifiy / Change

my_tuple = (4, 2, 3, [6, 5])
# TypeError: 'tuple' object does not support item assignment
# my_tuple[1] = 9
# However, item of mutable element can be changed
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
print(my_tuple)
# Tuples can be reassigned
my_tuple = ('p','r','o','g','r','a','m')

# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm')
print(my_tuple)

# Delete

my_tuple = ('p','r','o','g','r','a','m')
# can't delete items
# TypeError: 'tuple' object doesn't support item deletion
# del my_tuple[3]
# Can delete an entire tuple
del my_tuple
# NameError: name 'my_tuple' is not defined
print(my_tuple)
```

Function & Methods



METHODS	VS	FUNCTIONS
Associated with Objects.		Not associated with any objects
Cannot invoke by its name.		Can invoke by its name.
Dependent on class.		Independent.
Require 'self '.		Do not require 'self '.

Image source: https://techvidvan.com/tutorials/python-methods-vs-functions/

Function & Methods

Why functions required??

- Functions in Python are a set of related statements grouped together to carry out a specific task.
- Including functions in our program helps in making it much more organized and manageable.
- Especially, if we are working on a large program, having smaller and modular chunks of code blocks will increase the readability of the code along with providing it reusability.

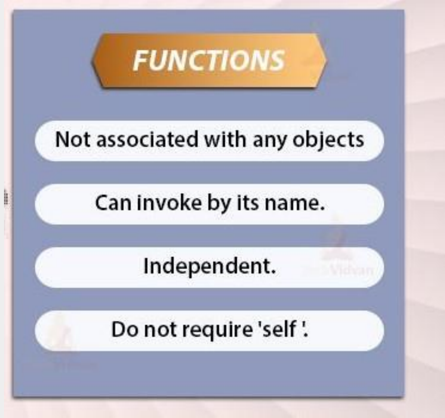
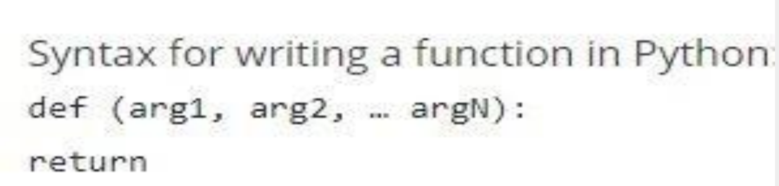


Image source: https://techvidvan.com/tutorials/python-methods-vs-functions/

Function & Methods

Create & Def Statement

- The def keyword is used to start the function definition.
- The def keyword is followed by a function-name which is followed by parentheses containing the arguments passed by the user and a colon at the end.
- After adding the colon, the body of the function starts with an indented block in a new line.
- The return statement sends a result object back to the caller.



Syntax for writing a function in Python

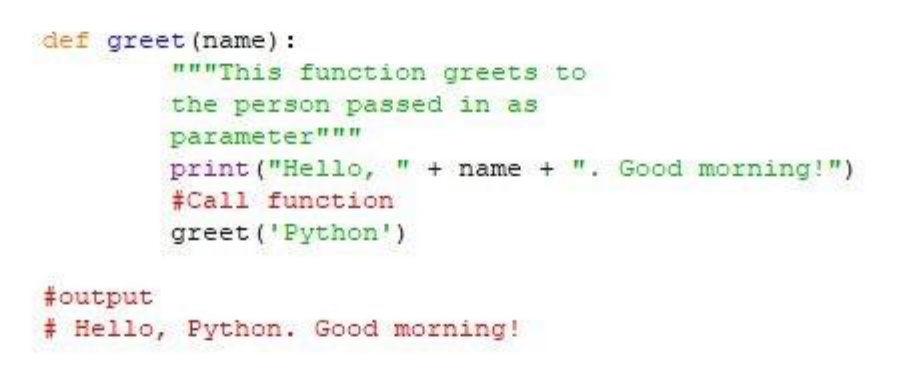
```
def (arg1, arg2, … argN):
return
```

Image Source : https://intellipaat.com/blog/tutorial/python-tutorial/python-functions/

Function & Methods

Calling a Function

- To call a function we simply type the function name with appropriate parameters.

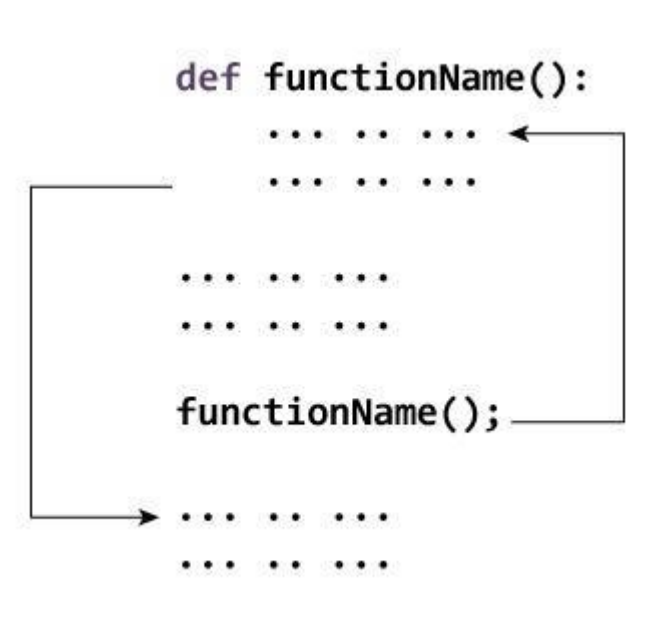


```
def greet(name):
        """This function greets to
        the person passed in as
        parameter"""
        print("Hello, " + name + ". Good morning!")
        #Call function
        greet('Python')

#output
# Hello, Python. Good morning!
```

Function & Methods

Function works & Types

- Basically, we can divide functions into the following two types:
- Built-in functions - Functions that are built into Python.
- User-defined functions - Functions defined by the users themselves



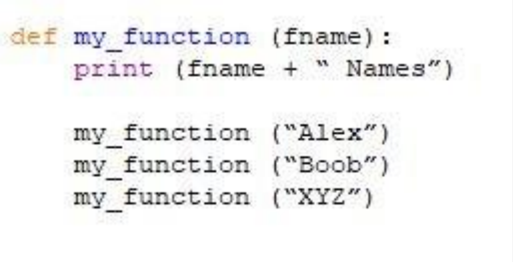
```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

Image Source: https://www.programiz.com/python-programming

Function & Methods

Arguments

- Define a function that takes variable number of arguments.

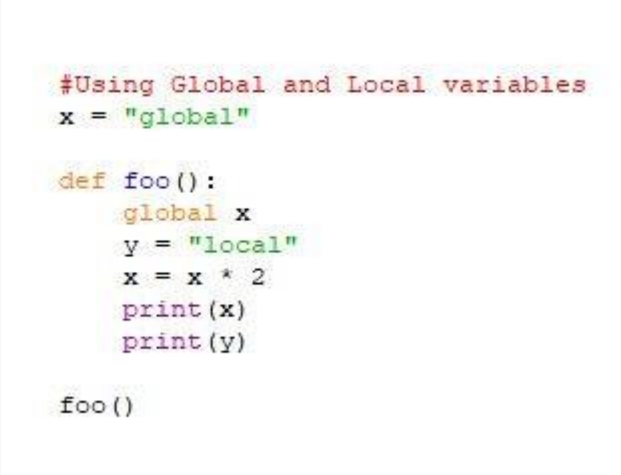


```
def my_function (fname):
    print (fname + “ Names”)

    my_function (“Alex”)
    my_function (“Boob”)
    my_function (“XYZ”)
```

Function & Methods

Global, Local and Nonlocal

- Global variable can be accessed inside or outside of the function.
- A variable declared inside the function's body or in the local scope is known as local variable.

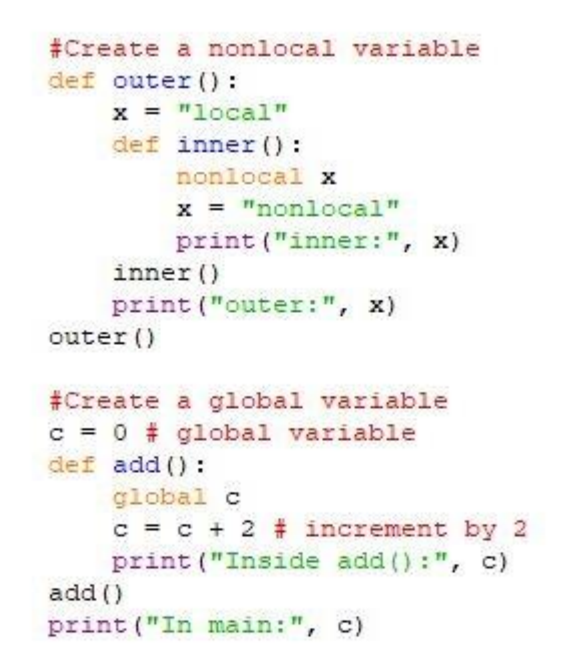


```
#Using Global and Local variables
x = "global"

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)
    print(y)

foo()
```

Function & Methods

Global, Local and Nonlocal

- Nonlocal variable are used in nested function whose local scope is not defined. This means, the variable can be neither in the local nor the global scope.
- We use nonlocal keyword to create nonlocal variable.
- In Python, global keyword allows you to modify the variable outside of the current scope.
- It is used to create a global variable and make changes to the variable in a local context.

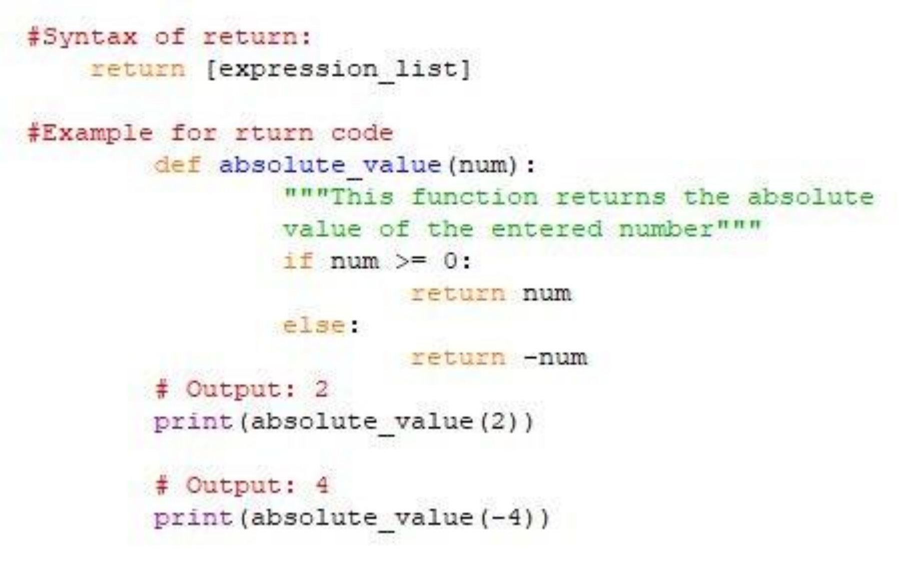


```
#Create a nonlocal variable
def outer():
    x = "local"
    def inner():
        nonlocal x
        x = "nonlocal"
        print("inner:", x)
    inner()
    print("outer:", x)
outer()

#Create a global variable
c = 0 # global variable
def add():
    global c
    c = c + 2 # increment by 2
    print("Inside add():", c)
add()
print("In main:", c)
```

Function & Methods

Return statement

- return statement is used to exit a function and go back to the place from where it was called.

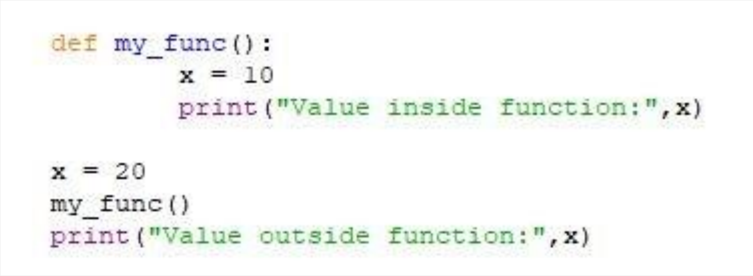


```
#Syntax of return:
    return [expression_list]

#Example for rturn code
        def absolute_value(num):
                """This function returns the absolute
                value of the entered number"""
                if num >= 0:
                        return num
                else:
                        return -num
        # Output: 2
        print(absolute_value(2))

        # Output: 4
        print(absolute_value(-4))
```

Function & Methods

Scope and Lifetime of variables

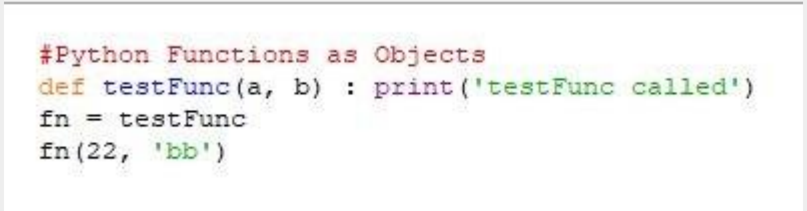
- Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
- Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.
- They are destroyed once we return from the function.



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

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

Function & Methods

Functions as Objects



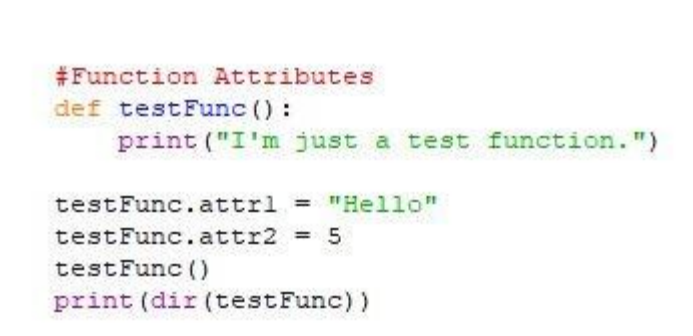
```
#Python Functions as Objects
def testFunc(a, b) : print('testFunc called')
fn = testFunc
fn(22, 'bb')
```

- Python treats everything as an object and functions are no different

Function & Methods

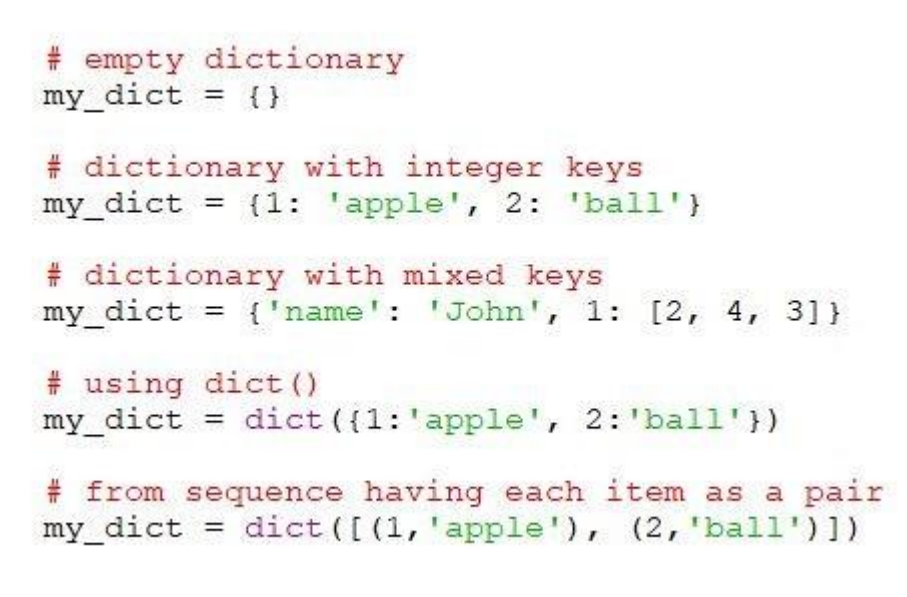
Function Attributes

- Python functions also have attributes.
- You can list them via the dir() built-in function.
- The attributes can be system-defined.
- Some of them can be user-defined as well.
- The dir() function also lists the user-defined attributes.



```
#Function Attributes
def testFunc():
    print("I'm just a test function.")

testFunc.attr1 = "Hello"
testFunc.attr2 = 5
testFunc()
print(dir(testFunc))
```

Dictionary

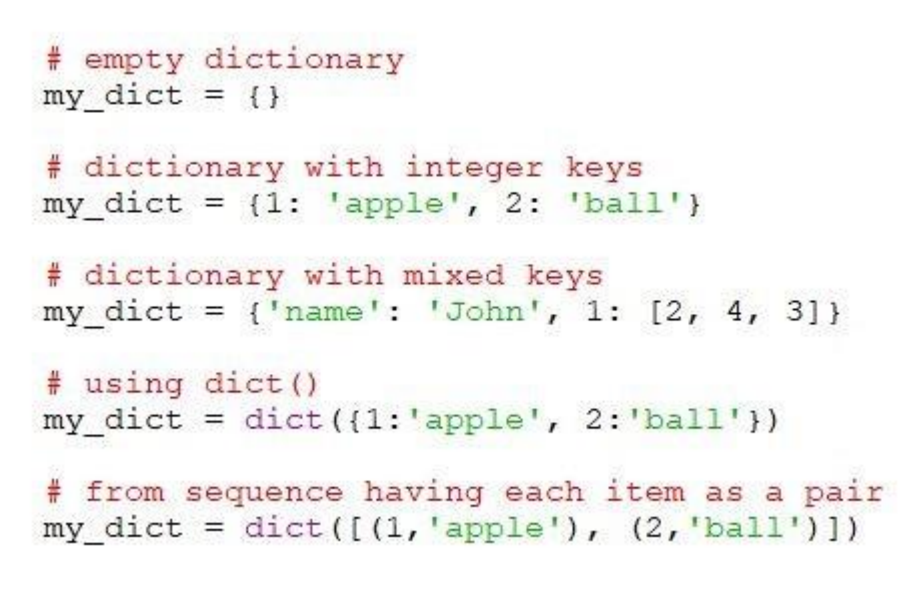


```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

Dictionary

Create a Dictionary

- Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.
- Dictionaries are optimized to retrieve values when the key is known.

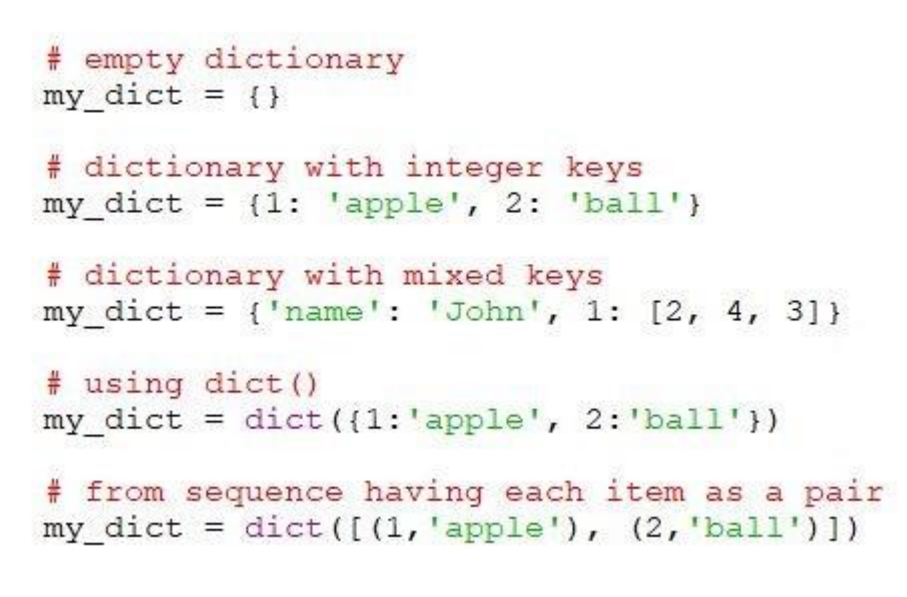


```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

Dictionary

Create a Dictionary

- Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.
- An item has a key and the corresponding value expressed as a pair, key: value.
- While values can be of any data type and can repeat, keys must be of immutable type
- (string, number or tuple with immutable elements) and must be unique.
- we can also create a dictionary using -

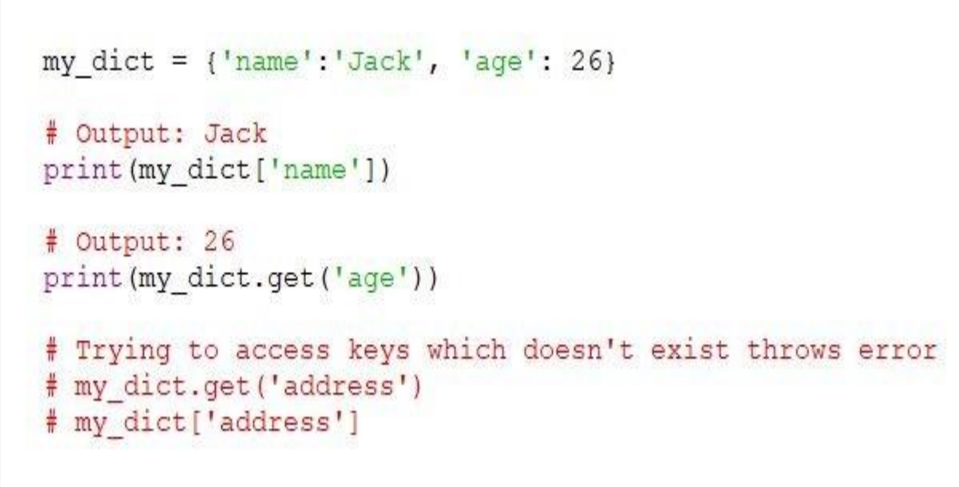


```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

Dictionary

Access elements from a dictionary

- While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.
- The difference while using get() is that it returns None instead of KeyError, if the key is not found.

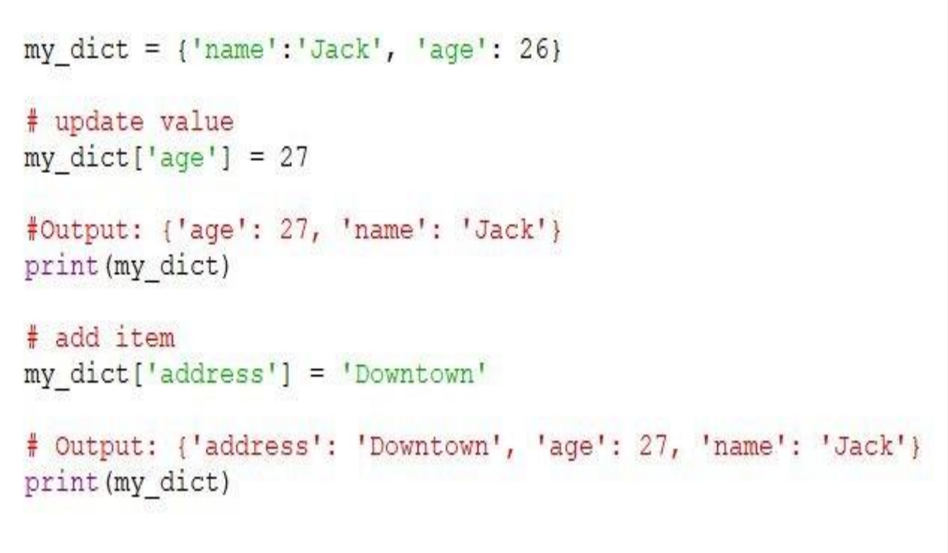


```
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# my_dict.get('address')
# my_dict['address']
```

Dictionary

Change or Add elements

- Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.
- If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

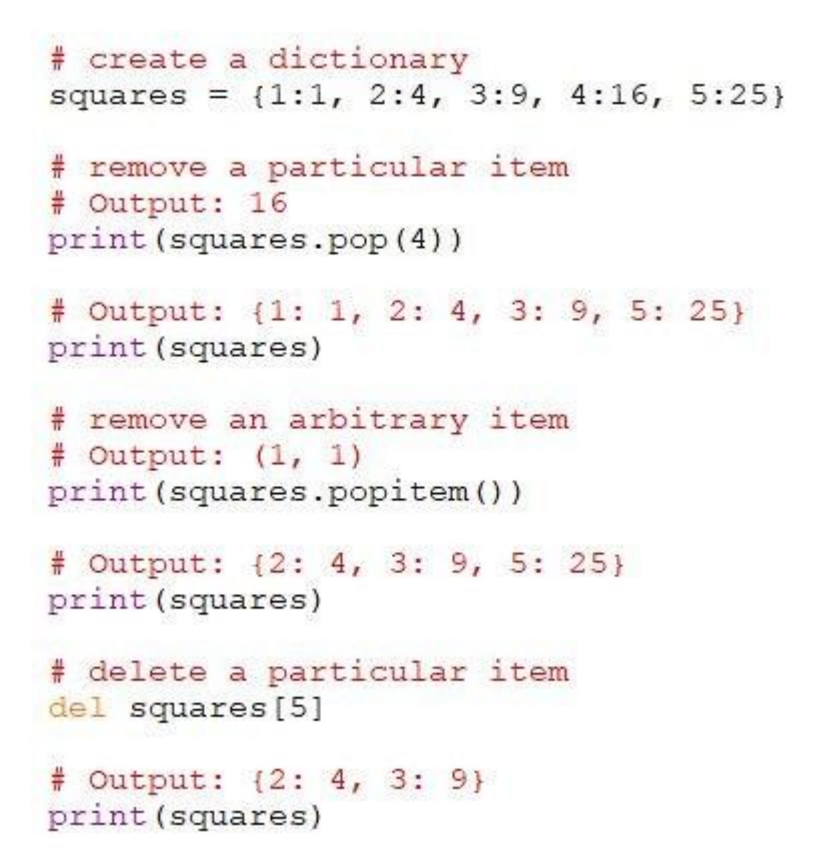


```
my_dict = {'name':'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

Dictionary

Delete or Remove elements

- We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.
- The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method.
- We can also use the del keyword to remove individual items or the entire dictionary itself.

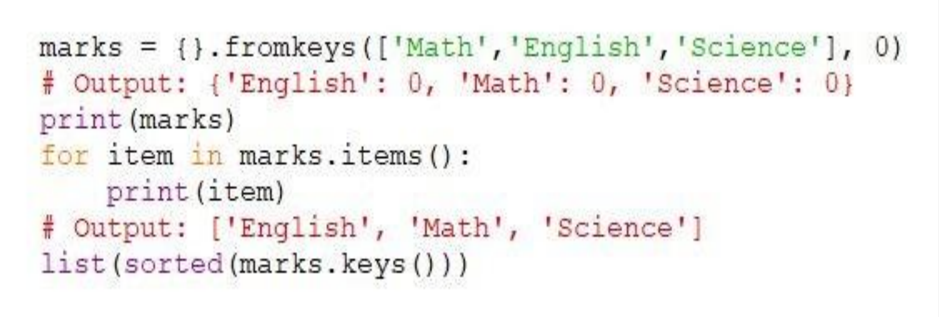


```
# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item
# Output: (1, 1)
print(squares.popitem())

# Output: {2: 4, 3: 9, 5: 25}
print(squares)

# delete a particular item
del squares[5]

# Output: {2: 4, 3: 9}
print(squares)
```

Dictionary

Example for use dictionary methods

Common Python Dictionary Methods



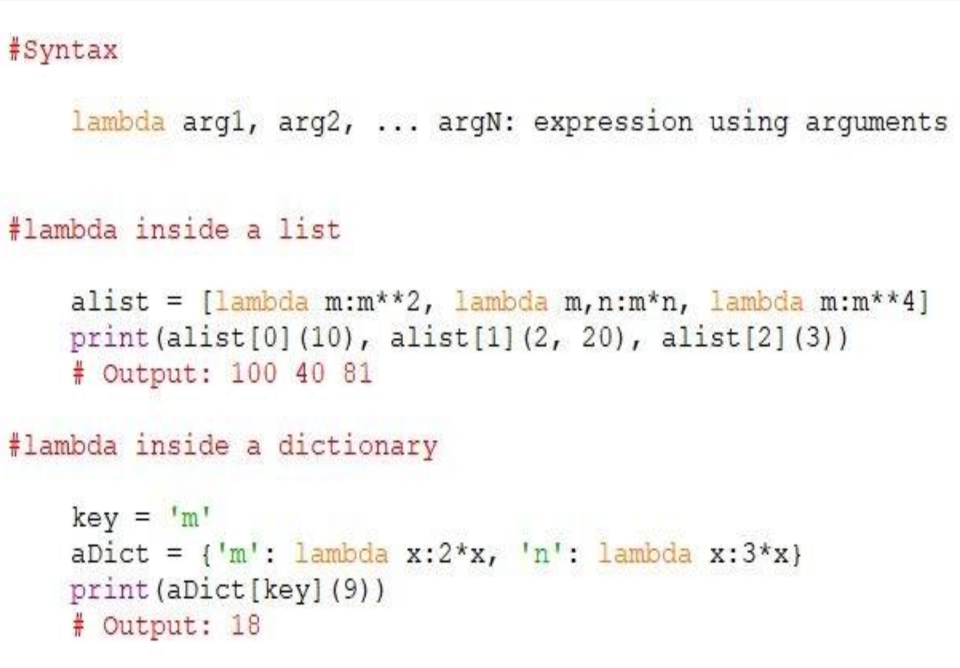
```
marks = {}.fromkeys(['Math','English','Science'], 0)
# Output: {'English': 0, 'Math': 0, 'Science': 0}
print(marks)
for item in marks.items():
    print(item)
# Output: ['English', 'Math', 'Science']
list(sorted(marks.keys()))
```

- Refer this link for more information
- https://docs.python.org/3/tutorial/datastr_ uctures.html

Functions

Built-in Functions Dictionary

- Built-in functions like all(), any(), len(), cmp(), sorted() etc. are commonly used with dictionary to perform different tasks.



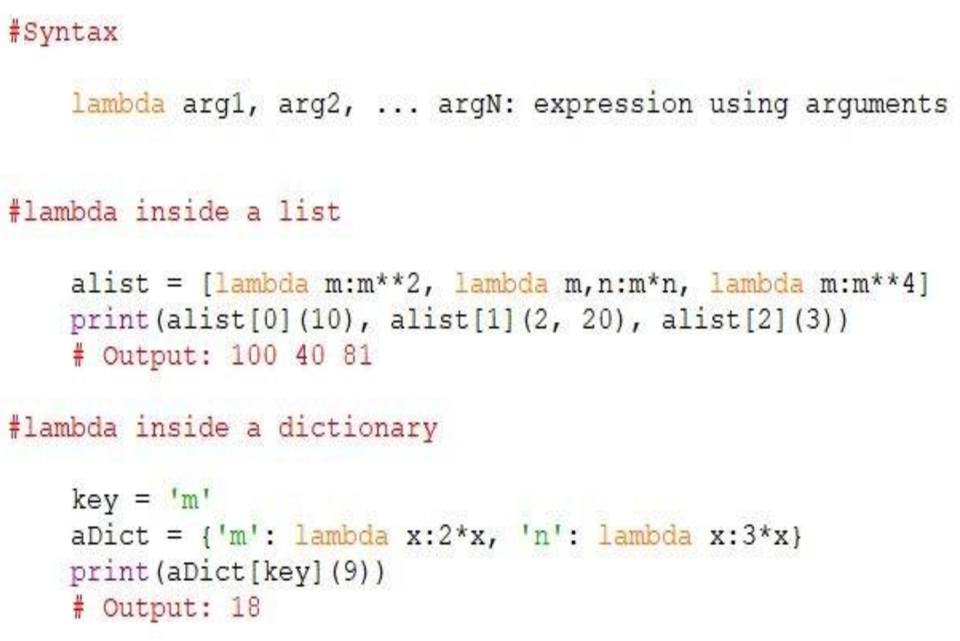
```
#Syntax

    lambda arg1, arg2, ... argN: expression using arguments

#lambda inside a list

    alist = [lambda m:m**2, lambda m,n:m*n, lambda m:m**4]
    print(alist[0](10), alist[1](2, 20), alist[2](3))
    # Output: 100 40 81

#lambda inside a dictionary

    key = 'm'
    aDict = {'m': lambda x:2*x, 'n': lambda x:3*x}
    print(aDict[key](9))
    # Output: 18
```

Functions

Lambda Functions

- Use Def keyword: It creates a function object and assigns it to a name.
- Use lambda: It creates an inline function and returns it as a result.

- A lambda function is a lightweight anonymous function. It can accept any number of arguments but can only have a single expression.

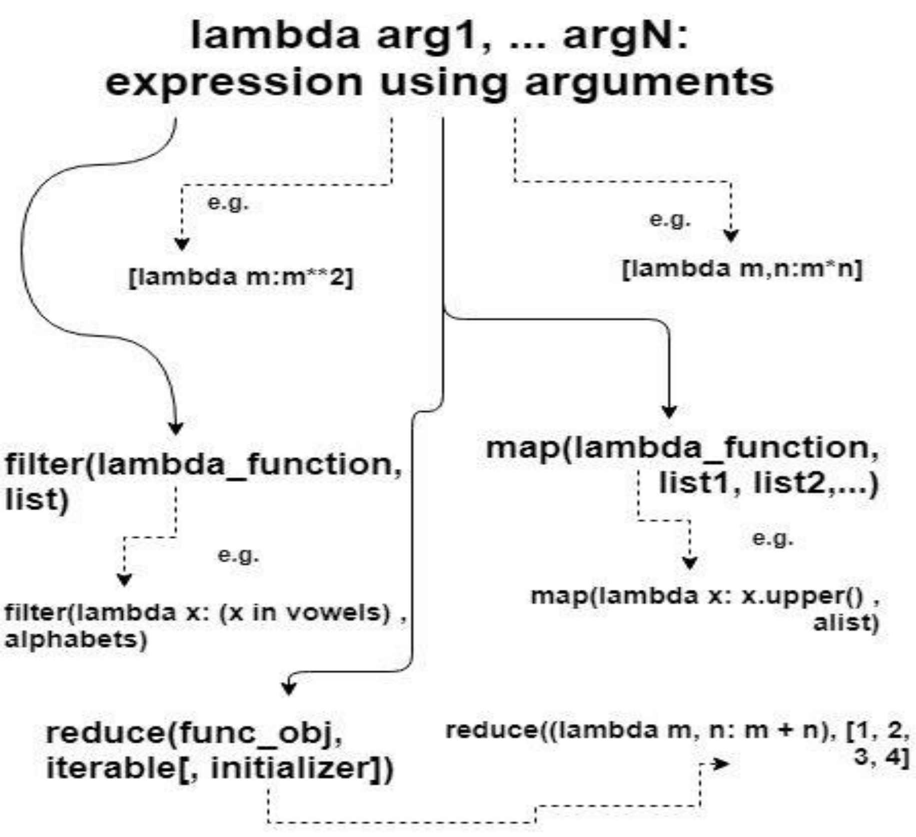


```
#Syntax

    lambda arg1, arg2, ... argN: expression using arguments

#lambda inside a list

    alist = [lambda m:m**2, lambda m,n:m*n, lambda m:m**4]
    print(alist[0](10), alist[1](2, 20), alist[2](3))
    # Output: 100 40 81

#lambda inside a dictionary

    key = 'm'
    aDict = {'m': lambda x:2*x, 'n': lambda x:3*x}
    print(aDict[key](9))
    # Output: 18
```

Functions

Use of Lambda Functions

- A Lambda function behaves like a regular function, takes an argument, and returns a value but is not bound to any name or identifier. There is no need to use the return statement in a lambda function in Python; it will always return the value obtained by evaluating the lambda expression in Python



lambda arg1, ... argN:
expression using arguments

e.g.

[lambda m:m**2]

e.g.

[lambda m,n:m*n]

filter(lambda_function, list)

e.g.

filter(lambda x: (x in vowels) , alphabets)

map(lambda_function, list1, list2,...)

e.g.

map(lambda x: x.upper() , alist)

reduce(func_obj, iterable[, initializer])

reduce((lambda m, n: m + n), [1, 2, 3, 4])

Image Source : https://www.techbeamers.com/python-lambda/

Functions

Properties of Lambda Functions

- Anonymous functions created using the lambda keyword can have any number of arguments, but they are syntactically restricted to just one expression, that is, they can have only one expression.
- Lambda function in Python can be used wherever a function object is required.
- Lambda functions do not require any return statement; they always return a value obtained by evaluating the lambda expression in Python.
- Python Lambda functions are widely used with some Python built-in function.

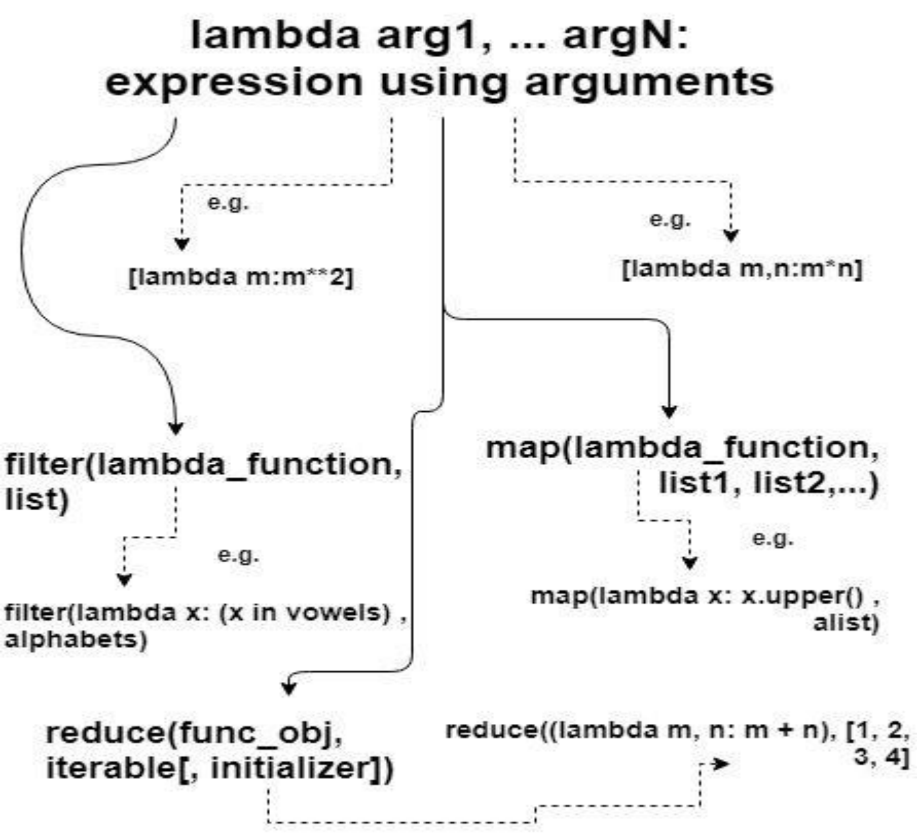
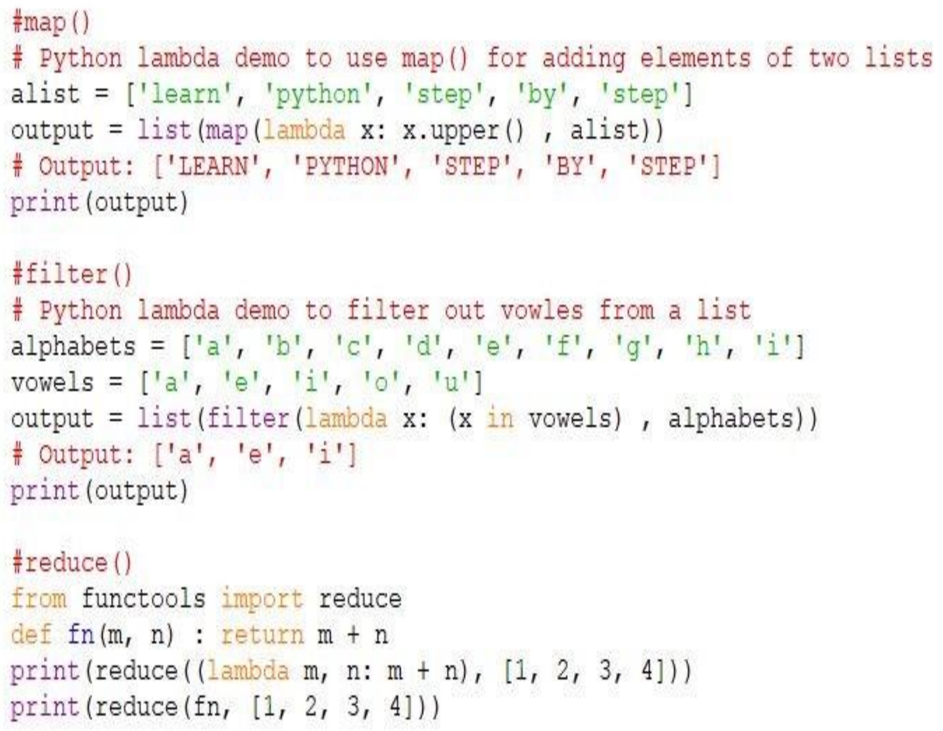

lambda arg1, ... argN:
expression using arguments
e.g.
[lambda m:m**2]
e.g.
[lambda m,n:m*n]
filter(lambda_function, list)
e.g.
filter(lambda x: (x in vowels) , alphabets)
map(lambda_function, list1, list2,...)
e.g.
map(lambda x: x.upper() , alist)
reduce(func_obj, iterable[, initializer])
reduce((lambda m, n: m + n), [1, 2, 3, 4])

Image Source : https://www.techbeamers.com/python-lambda/

Functions

Built in Functions

- Map functions over iterables – map()
- Select items in iterables – filter()
- Aggregate items in iterables – reduce()



```
#map()
# Python lambda demo to use map() for adding elements of two lists
alist = ['learn', 'python', 'step', 'by', 'step']
output = list(map(lambda x: x.upper() , alist))
# Output: ['LEARN', 'PYTHON', 'STEP', 'BY', 'STEP']
print(output)

#filter()
# Python lambda demo to filter out vowles from a list
alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
vowels = ['a', 'e', 'i', 'o', 'u']
output = list(filter(lambda x: (x in vowels) , alphabets))
# Output: ['a', 'e', 'i']
print(output)

#reduce()
from functools import reduce
def fn(m, n) : return m + n
print(reduce((lambda m, n: m + n), [1, 2, 3, 4]))
print(reduce(fn, [1, 2, 3, 4]))
```

Modules

- Import: Lets a client obtain a module as a whole
- From: Permits a client to fetch particular names from a module
- Reload: Gives a way to reload a code of module without stopping Python

Modules

What are modules?

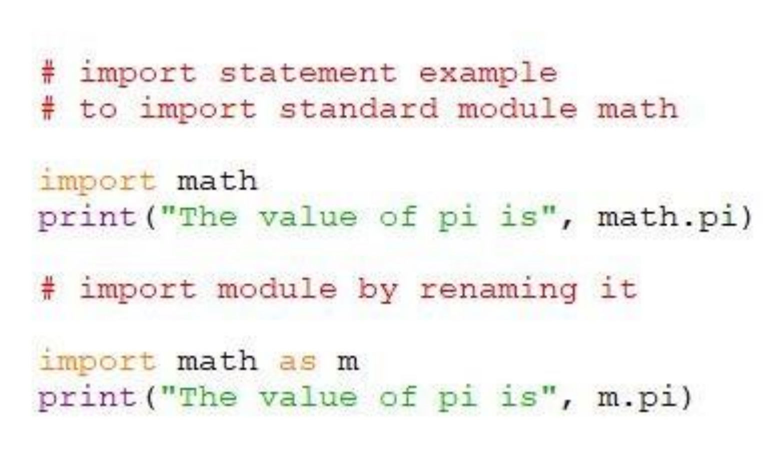
- In Python, modules are used to divide the code into smaller parts. In this, we can group similar data which makes the program easier to understand.
- The module is a simple Python file which can contain (Python functions, python variables, python classes) .
- Modules are processed with two new statements and one important built-in function

- Import: Lets a client obtain a module as a whole
- From: Permits a client to fetch particular names from a module
- Reload: Gives a way to reload a code of module without stopping Python

Modules

import statement import with renaming

- import statement - We can import a module using import statement and access the definitions inside it using the dot operator as described rightside.
- import with renaming- We can import a module by renaming it as follows.

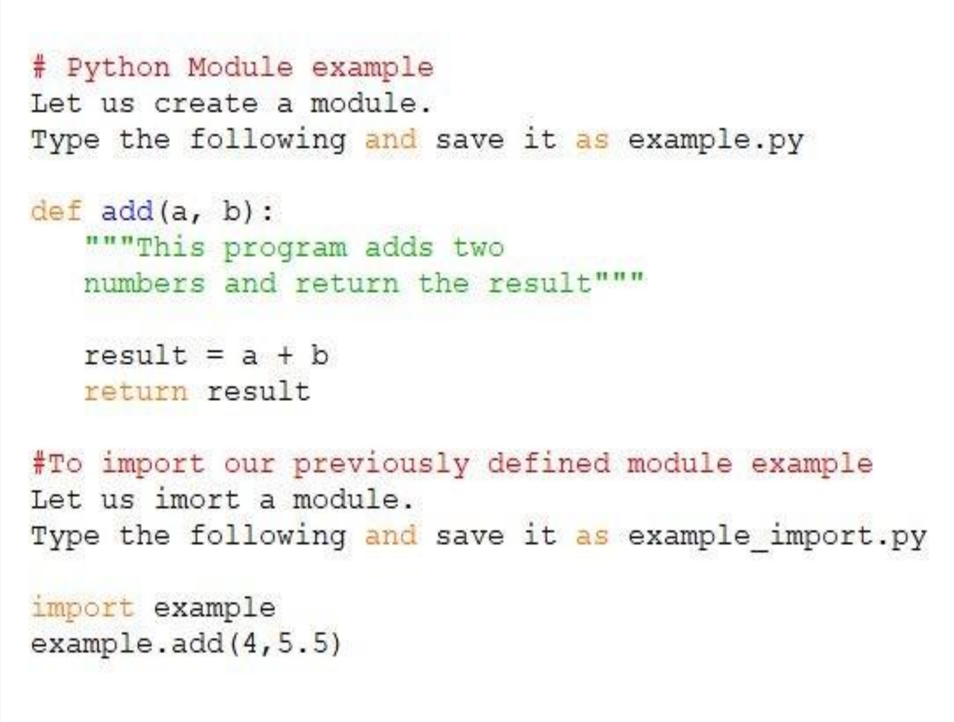


```
# import statement example
# to import standard module math

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

# import module by renaming it

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

Modules

Import modules

- We can import the definitions inside a module to another module or the interactive interpreter in Python.
- We use the import keyword to do this



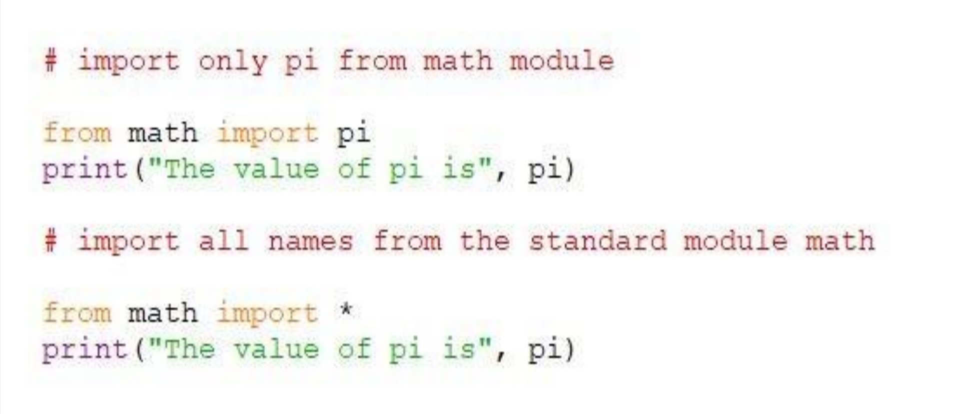
```
# Python Module example
Let us create a module.
Type the following and save it as example.py

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

   result = a + b
   return result

#To import our previously defined module example
Let us imort a module.
Type the following and save it as example_import.py

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

Modules

from...import statement

Import all names

- Python from...import statement
- We can import specific names from a module without importing the module as a whole.
- Import all names
- We can import all names(definitions) from a module using the following construct.

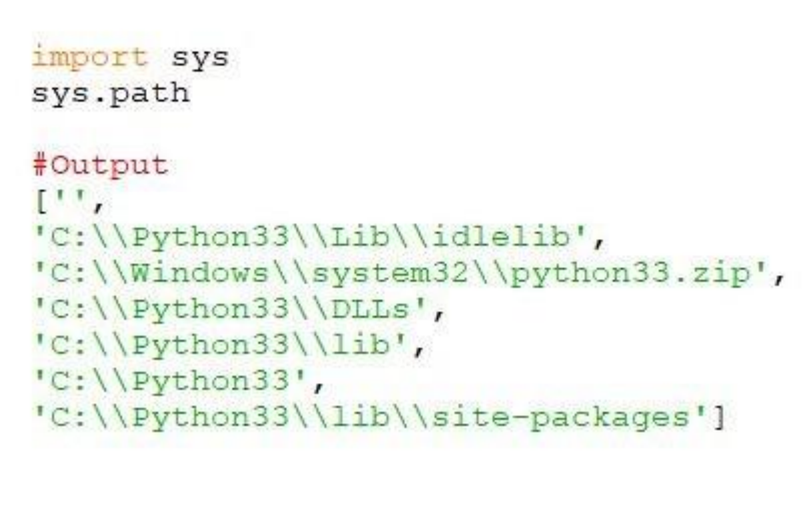


```
# import only pi from math module

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

# import all names from the standard module math

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

Modules

Module Search Path

- While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order.
- The current directory.
- PYTHONPATH (an environment variable with a list of directory).
- The installation-dependent default directory.

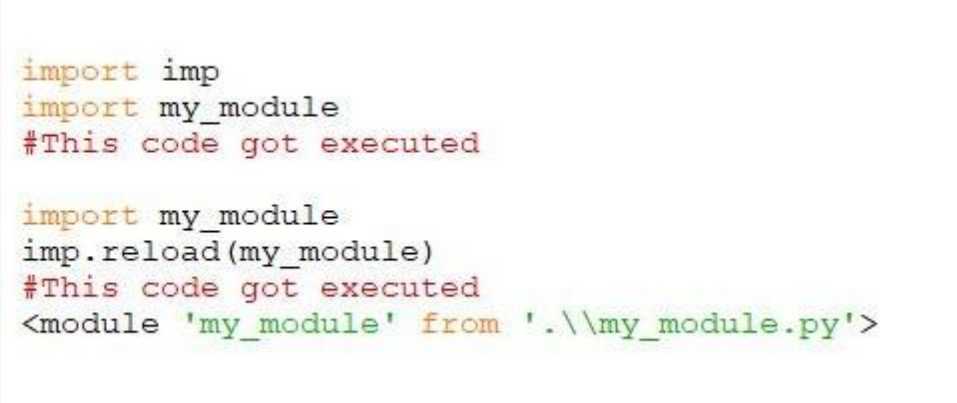


```
import sys
sys.path

#Output
['',
'C:\\Python33\\Lib\\idlelib',
'C:\\Windows\\system32\\python33.zip',
'C:\\Python33\\DLLs',
'C:\\Python33\\lib',
'C:\\Python33',
'C:\\Python33\\lib\\site-packages']
```

Modules

Reloading a module

- The Python interpreter imports a module only once during a session. This makes things more efficient.
- Python provides a neat way of doing this. We can use the reload() function inside the imp module to reload a module.

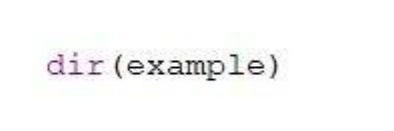


```
import imp
import my_module
#This code got executed

import my_module
imp.reload(my_module)
#This code got executed
<module 'my_module' from '.\\my_module.py'>
```

Modules

dir() built-in function

- We can use the dir() function to find out names that are defined inside a module.



```
dir(example)
```

Modules

Package

- Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.
- A directory must contain a file
- Named init .py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

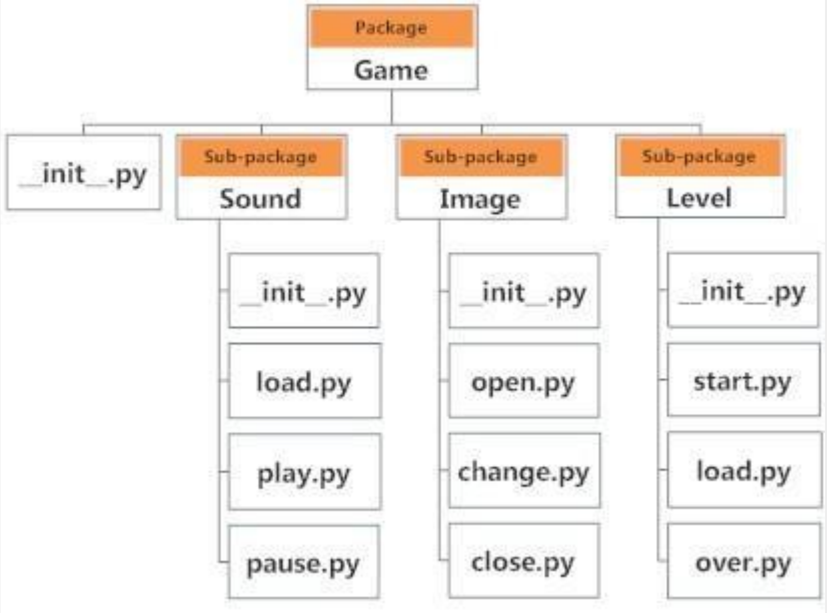
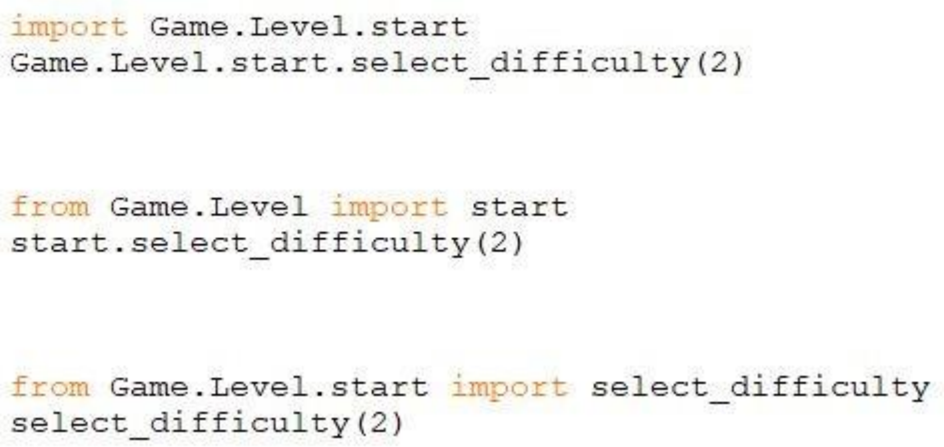


Image Source:https://www.programiz.com/python-programming/package

Modules

Importing module from a package

- We can import modules from packages using the dot (.) operator
- For examples : 3 different types of imports

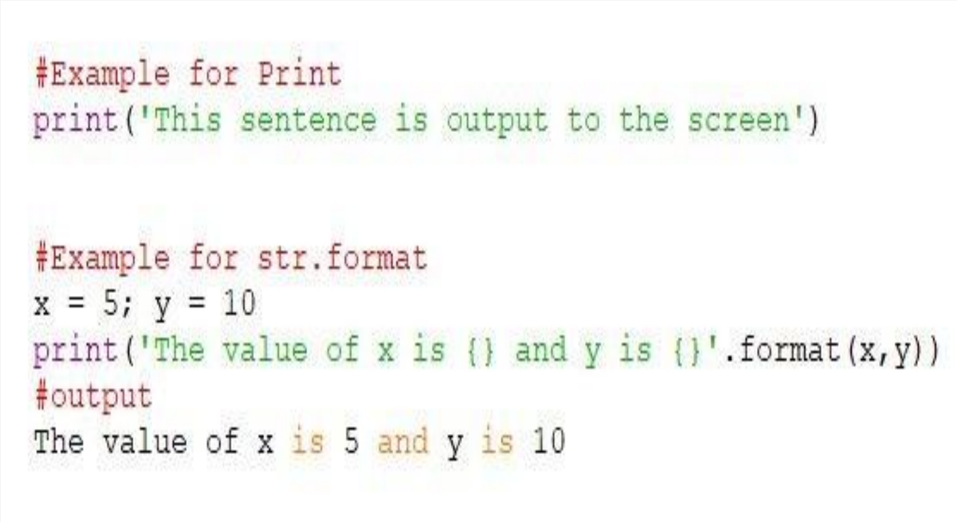


```
import Game.Level.start
Game.Level.start.select_difficulty(2)
```

```
from Game.Level import start
start.select_difficulty(2)
```

```
from Game.Level.start import select_difficulty
select_difficulty(2)
```

Input and Output

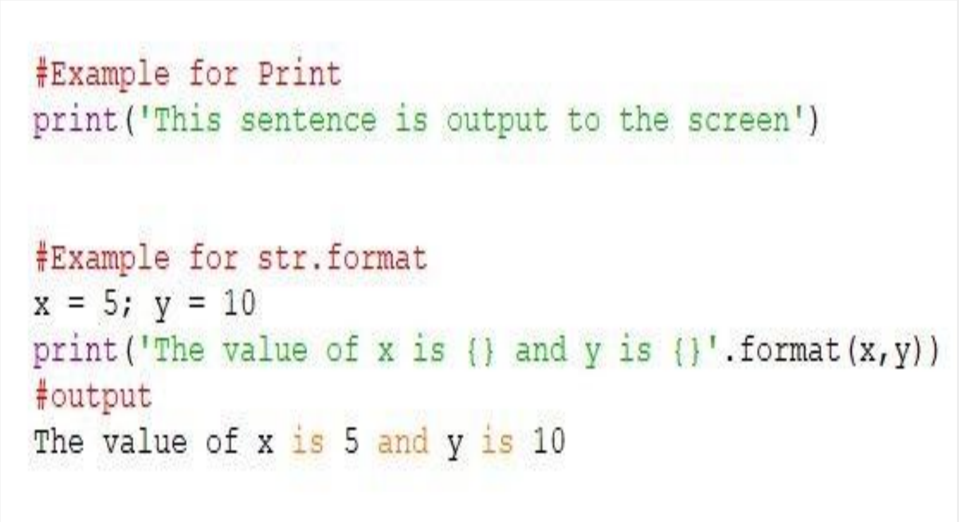


```
#Example for Print
print('This sentence is output to the screen')


#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

Output

- print() function to output data to the standard output device (screen).
- str.format() method - format our output to make it look attractive
- curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index)

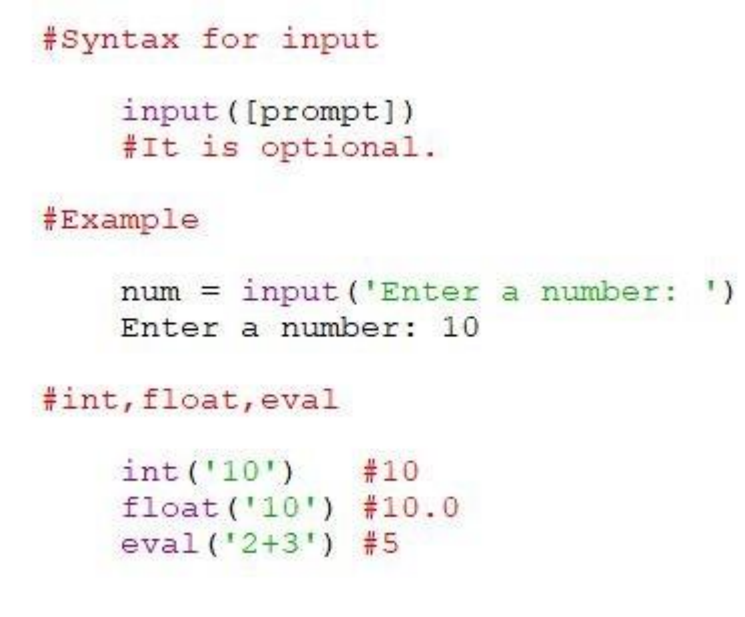


```
#Example for Print
print('This sentence is output to the screen')

#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

Input

- To allow flexibility, we might want to take the input from the user.
- In Python, we have the input() function to allow this.
- To convert this into a number we can use int() or float() functions.
- Above same operation can be performed using the eval() function. But eval takes it further.
- It can evaluate even expressions, provided the input is a string

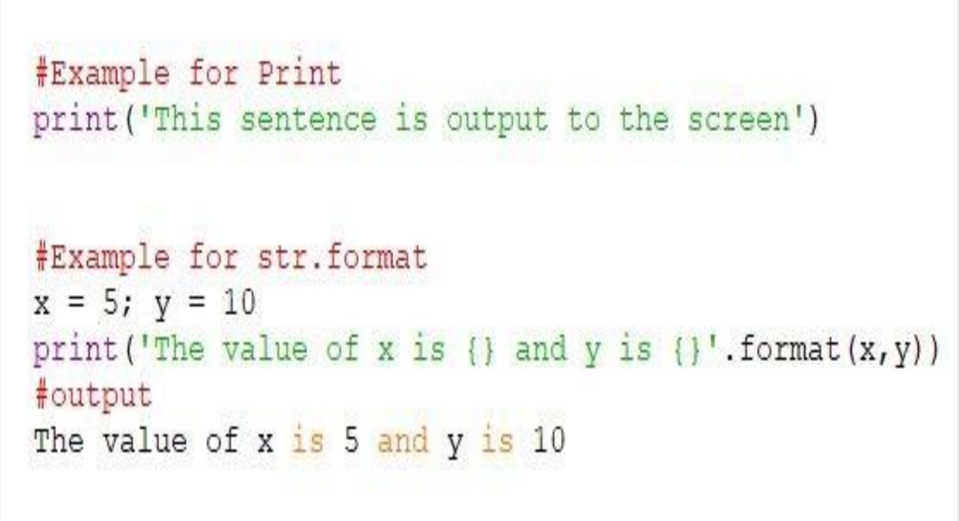


```
#Syntax for input

    input([prompt])
    #It is optional.

#Example

    num = input('Enter a number: ')
    Enter a number: 10

#int,float,eval

    int('10')    #10
    float('10') #10.0
    eval('2+3') #5
```

Input and Output

Manual String Formatting

- See Notes section with examples

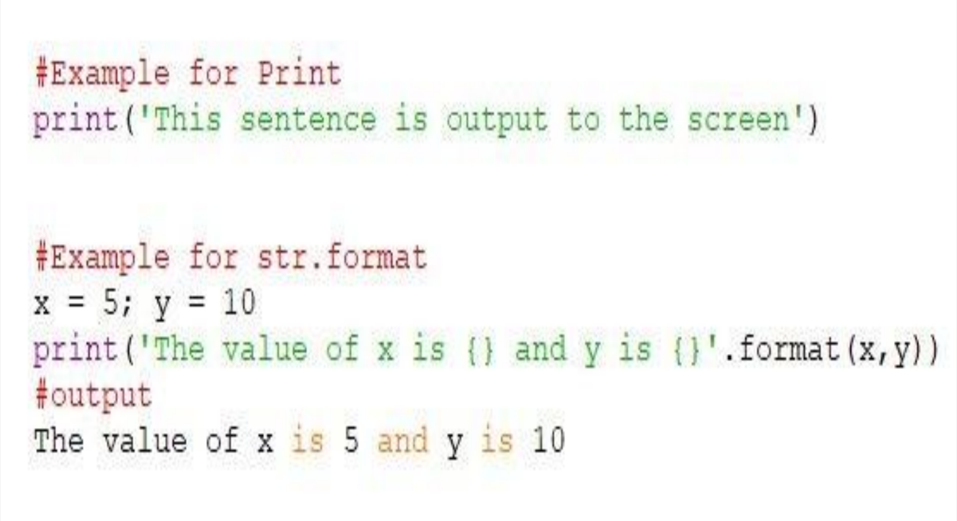


```
#Example for Print
print('This sentence is output to the screen')


#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

File handling means

- File is a named location on the system storage which records data for later access. It enables persistent storage in a non-volatile memory i.e. Hard disk.

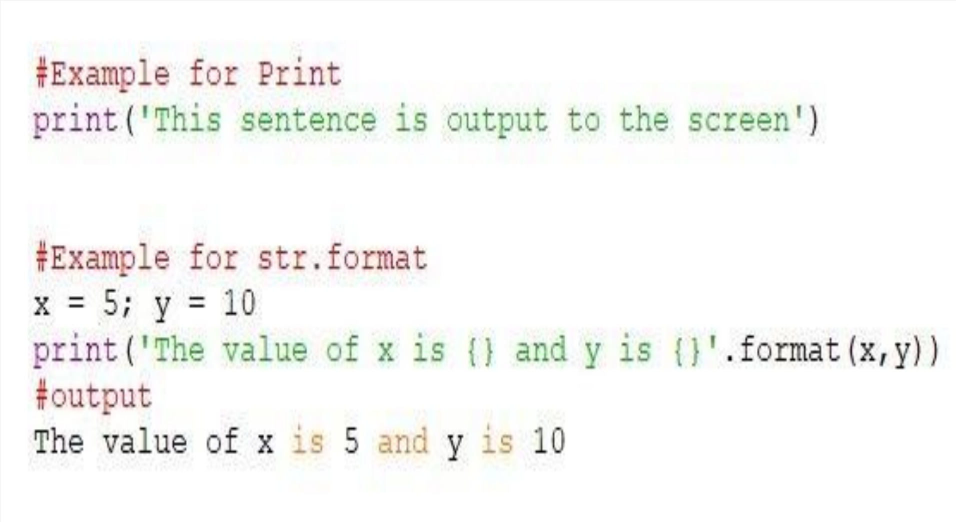


```
#Example for Print
print('This sentence is output to the screen')


#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

Python file handling

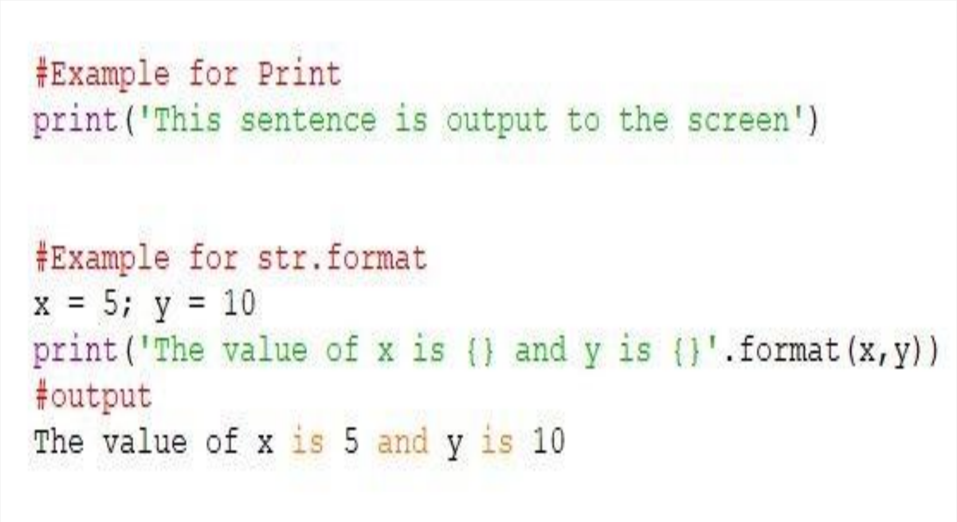
- Python I/O deal with two types of files.
- Text & Binary Files
- Even though the two file types may look the same on the surface, they encode data differently.
- See notes section for more information



```
#Example for Print
print('This sentence is output to the screen')


#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

- In Python, file processing takes place in the following order.
- Open a file that returns a filehandle.
- Use the handle to perform read or write action.
- Close the filehandle.

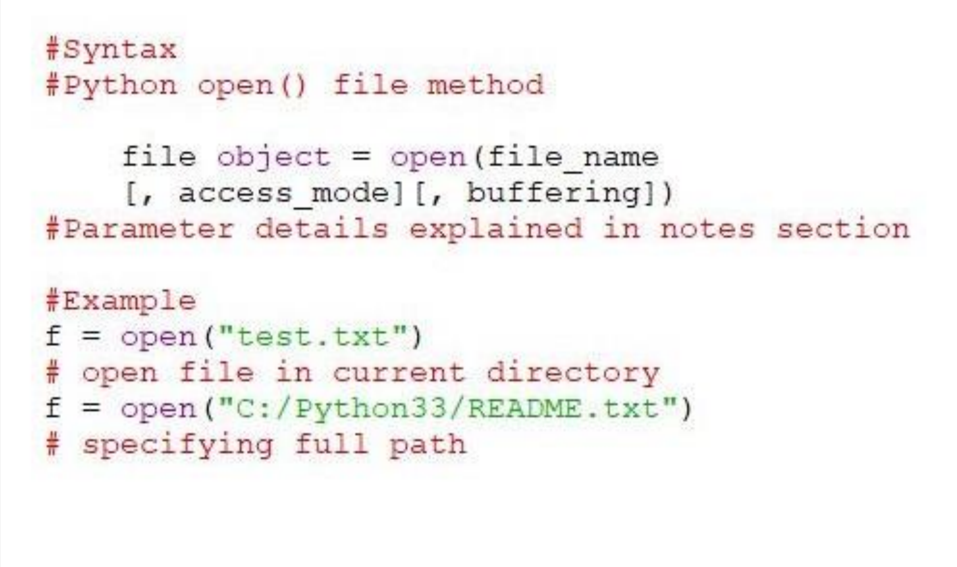


```
#Example for Print
print('This sentence is output to the screen')


#Example for str.format
x = 5; y = 10
print('The value of x is {} and y is {}'.format(x,y))
#output
The value of x is 5 and y is 10
```

Input and Output

Open a file in Python

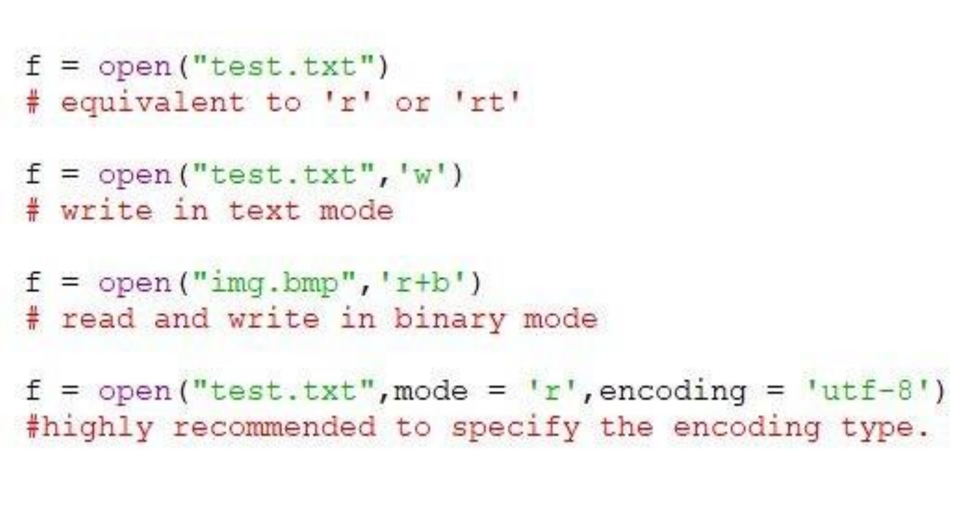
- To read or write to a file, you need to open it first. To open a file in Python, use its built open() function. This function returns a file object, i.e., a handle. You can use it to read or modify the file.



```
#Syntax
#Python open() file method

    file object = open(file_name
    [, access_mode][, buffering])
#Parameter details explained in notes section

#Example
f = open("test.txt")
# open file in current directory
f = open("C:/Python33/README.txt")
# specifying full path
```

Input and Output

- 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 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.
- Hence, when working with files in text mode, it is highly recommended to specify the encoding type.

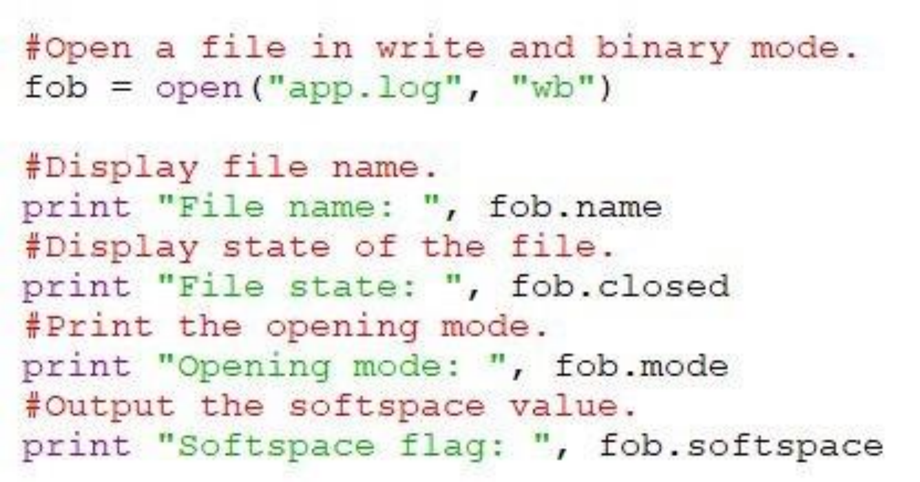


```
f = open("test.txt")
# equivalent to 'r' or 'rt'

f = open("test.txt",'w')
# write in text mode

f = open("img.bmp",'r+b')
# read and write in binary mode

f = open("test.txt",mode = 'r',encoding = 'utf-8')
#highly recommended to specify the encoding type.
```

Input and Output

Python file object attributes

- When you call the Python open() function, it returns an object, which is the filehandle. Also, you should know that Python files have several linked attributes. And we can make use of the filehandle to list the attributes of a file.

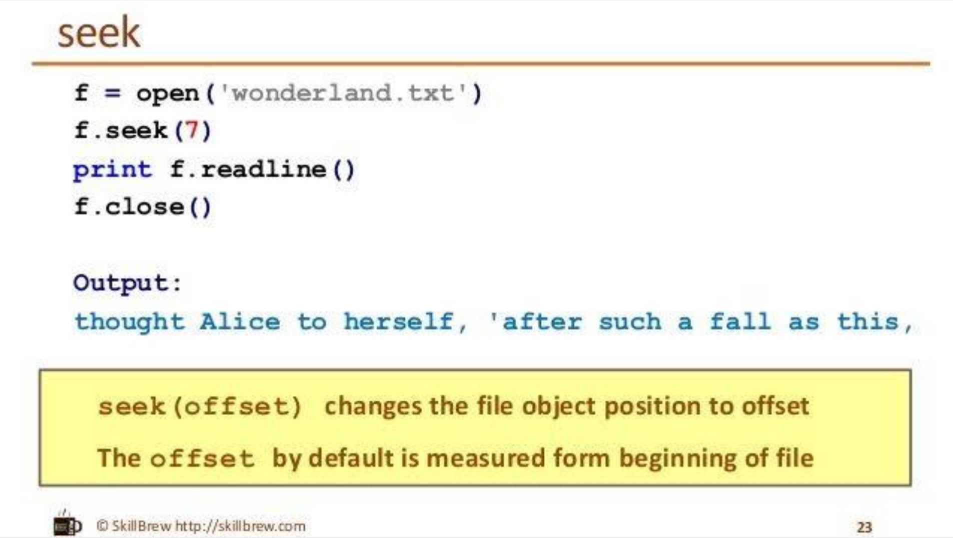


```
#Open a file in write and binary mode.
fob = open("app.log", "wb")

#Display file name.
print "File name: ", fob.name
#Display state of the file.
print "File state: ", fob.closed
#Print the opening mode.
print "Opening mode: ", fob.mode
#Output the softspace value.
print "Softspace flag: ", fob.softspace
```

Input and Output

Python File object methods

- file.close()
- file.flush()
- file.isatty()
- file.tell()
- file.write(string)
- file.next()
- file.read(size)
- See notes section for description



seek

```
f = open('wonderland.txt')
f.seek(7)
print f.readline()
f.close()

Output:
thought Alice to herself, 'after such a fall as this,
```

seek(offset) changes the file object position to offset

The offset by default is measured form beginning of file

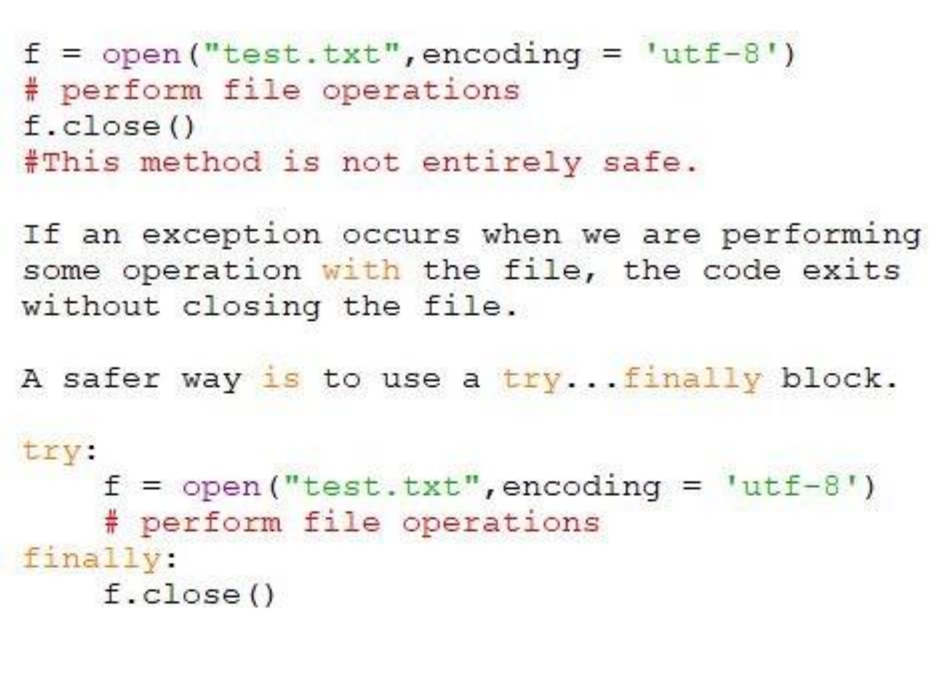
© SkillBrew http://skillbrew.com 23

Image source - https://www.slideshare.net/p3infotech_solutions/python-programming-essentials-m22-file-opera

Input and Output

close a file

- When we are done with operations to the file, we need to properly close the file.
- Closing a file will free up the resources that were tied with the file and is done using Python close() method.
- Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

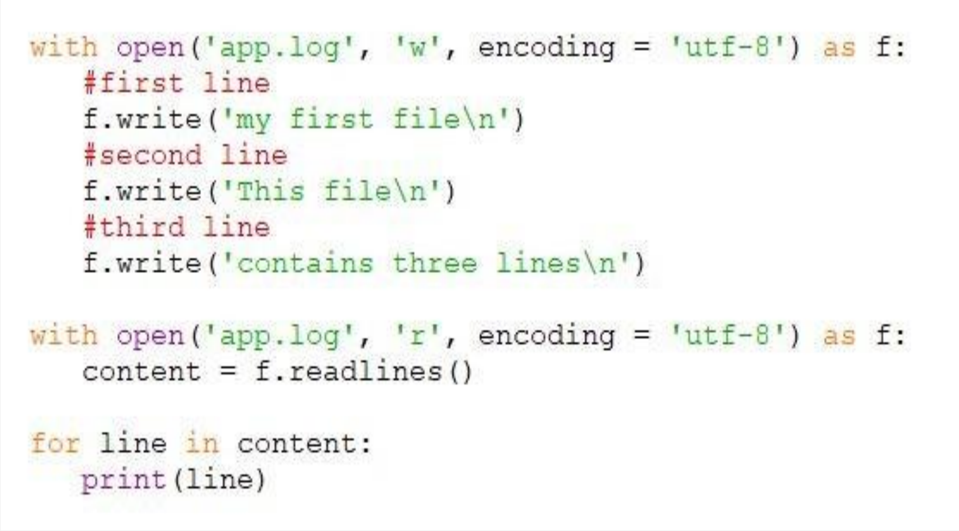


```
f = open("test.txt",encoding = 'utf-8')
# perform file operations
f.close()
#This method is not entirely safe.

If an exception occurs when we are performing
some operation with the file, the code exits
without closing the file.

A safer way is to use a try...finally block.

try:
    f = open("test.txt",encoding = 'utf-8')
    # perform file operations
finally:
    f.close()
```

Input and Output

write() file method

- Python provides the write() method to write a string or sequence of bytes to a file. This function returns a number, which is the size of data written in a single Write call.

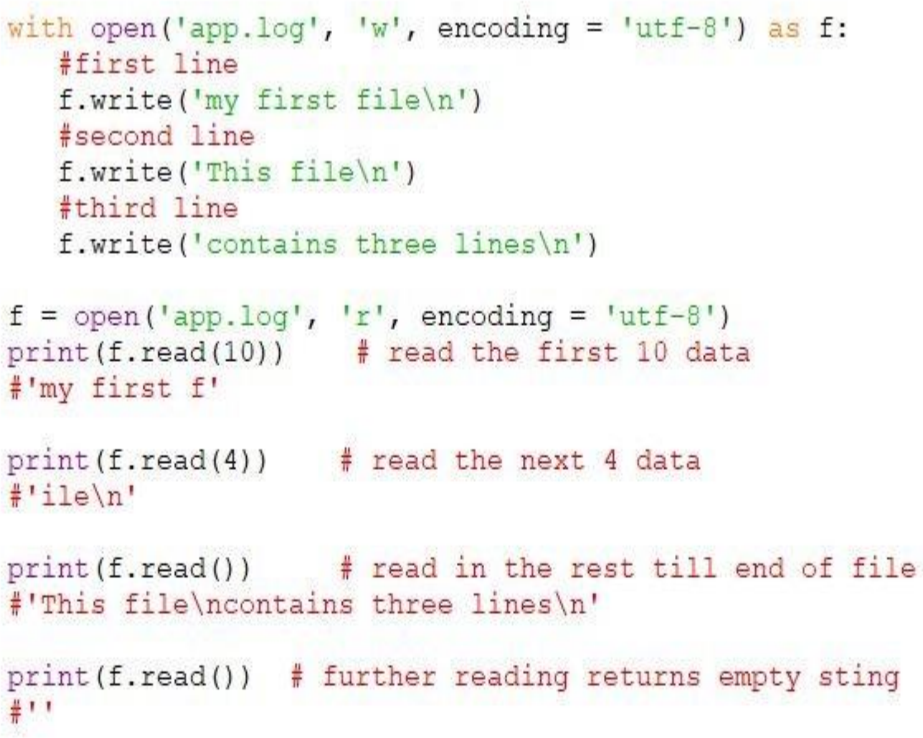


```
with open('app.log', 'w', encoding = 'utf-8') as f:
   #first line
   f.write('my first file\n')
   #second line
   f.write('This file\n')
   #third line
   f.write('contains three lines\n')

with open('app.log', 'r', encoding = 'utf-8') as f:
   content = f.readlines()

for line in content:
   print(line)
```

Input and Output

Perform read operation

- To read data from a file, first of all, you need to open it in reading mode. Then, you can call any of the methods that Python provides for reading from a file.
- Usually, we use
- Python <read(size)> function to read the content of a file up to the size. If you don't pass the size, then it'll read the whole file.



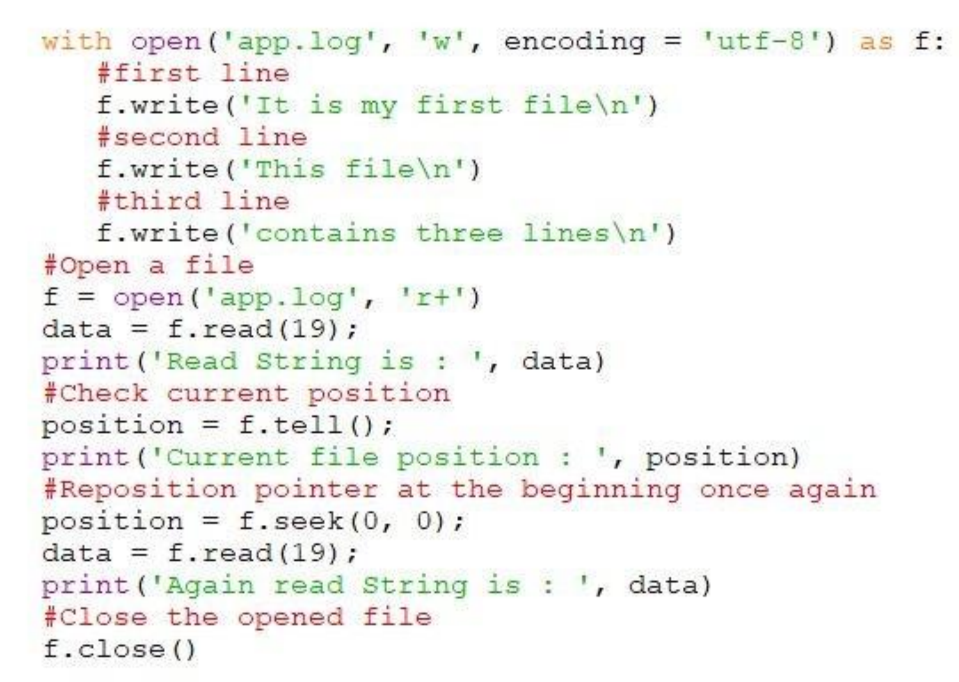
```
with open('app.log', 'w', encoding = 'utf-8') as f:
    #first line
    f.write('my first file\n')
    #second line
    f.write('This file\n')
    #third line
    f.write('contains three lines\n')

f = open('app.log', 'r', encoding = 'utf-8')
print(f.read(10))    # read the first 10 data
#'my first f'

print(f.read(4))    # read the next 4 data
#'ile\n'

print(f.read())     # read in the rest till end of file
#'This file\ncontains three lines\n'

print(f.read())  # further reading returns empty sting
#''
```

Input and Output

Set File offset

- Tell() Method
- Syntax: file.tell()

  Seek() Method
- Syntax: file.seek(offset[, from])

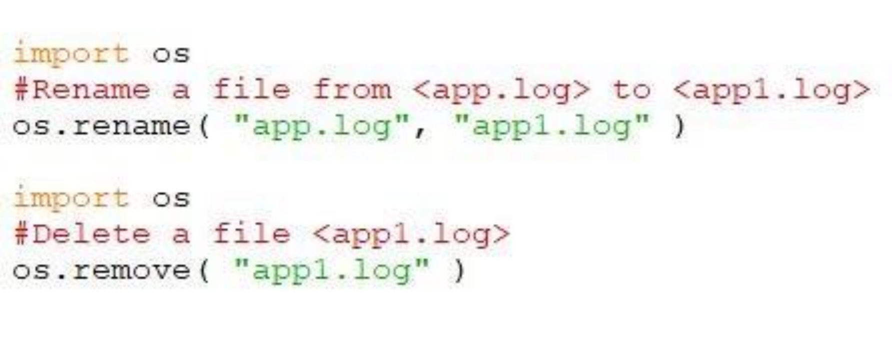


```
with open('app.log', 'w', encoding = 'utf-8') as f:
    #first line
    f.write('It is my first file\n')
    #second line
    f.write('This file\n')
    #third line
    f.write('contains three lines\n')
#Open a file
f = open('app.log', 'r+')
data = f.read(19);
print('Read String is : ', data)
#Check current position
position = f.tell();
print('Current file position : ', position)
#Reposition pointer at the beginning once again
position = f.seek(0, 0);
data = f.read(19);
print('Again read String is : ', data)
#Close the opened file
f.close()
```

Input and Output

Renaming and deleting files

- Rename - os.rename(cur_file, new_file) The <rename()> method takes two arguments, the current filename and the new filename.
- Remove - os.remove(file_name)
- The <remove()> method deletes a file which it receives in the argument.

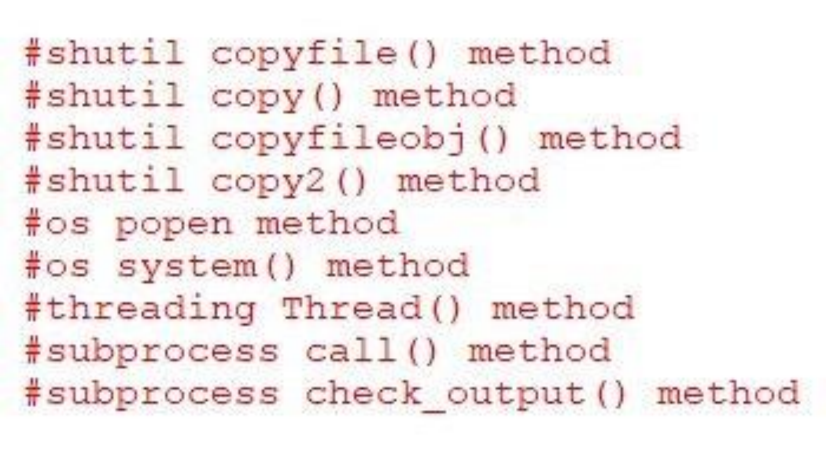


```
import os
#Rename a file from <app.log> to <app1.log>
os.rename( "app.log", "app1.log" )

import os
#Delete a file <app1.log>
os.remove( "app1.log" )
```

Input and Output

Python Copy File – 9 Ways

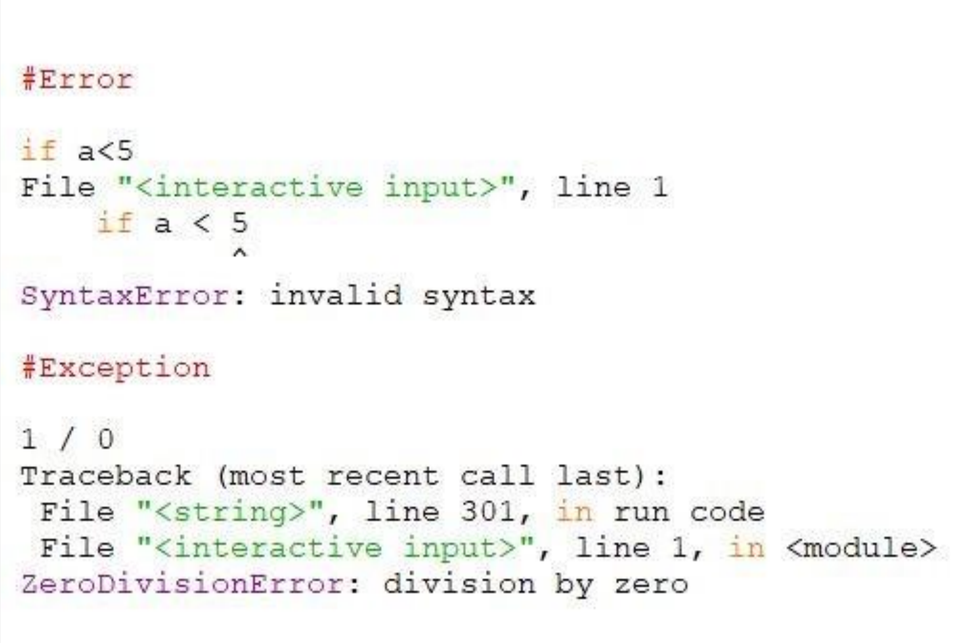
- Different methods to do Python copy file operation.



```
#shutil copyfile() method
#shutil copy() method
#shutil copyfileobj() method
#shutil copy2() method
#os popen method
#os system() method
#threading Thread() method
#subprocess call() method
#subprocess check_output() method
```

Image Source: https://www.techbeamers.com/python-copy-file/

Exception Handling

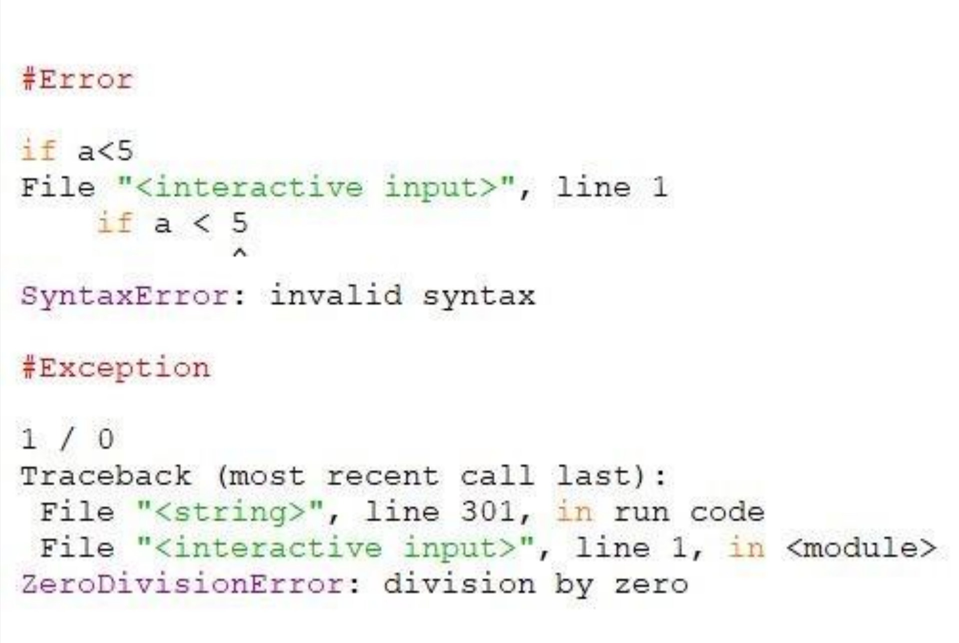


```
#Error

if a<5
File "<interactive input>", line 1
    if a < 5
           ^
SyntaxError: invalid syntax

#Exception

1 / 0
Traceback (most recent call last):
 File "<string>", line 301, in run code
 File "<interactive input>", line 1, in <module>
ZeroDivisionError: division by zero
```

Exception Handling

Error vs. Exception in Python

- Error - Error is something that goes wrong in the program, e.g., like a syntactical error. It occurs at compile time.
- Exception - An exception is an event which occurs during the execution of a program and disrupts the normal flow of the program’s instructions.

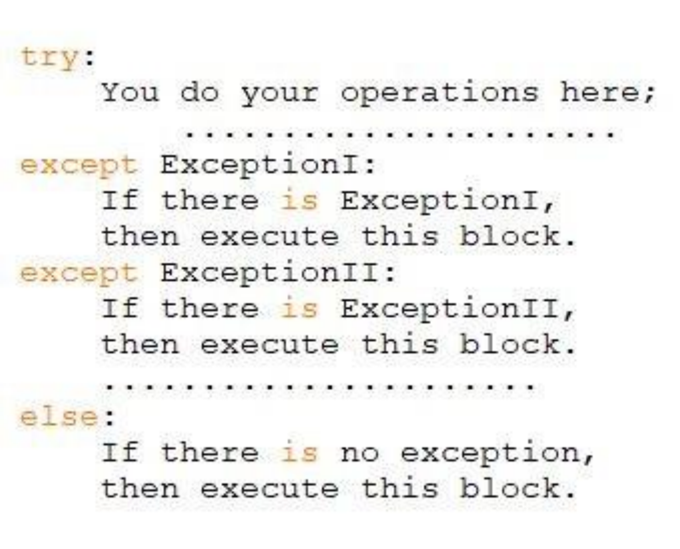


```
#Error

if a<5
File "<interactive input>", line 1
    if a < 5
           ^
SyntaxError: invalid syntax

#Exception

1 / 0
Traceback (most recent call last):
 File "<string>", line 301, in run code
 File "<interactive input>", line 1, in <module>
ZeroDivisionError: division by zero
```

Exception Handling

Handle Exceptions

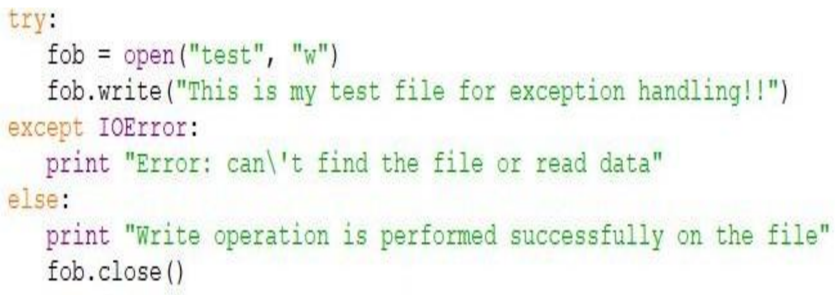
- Try-Except Statement?
- We use the try-except statement to enable exception handling in Python programs.
- Inside the try block, you write the code which can raise an exception.
- And the code that handles or catches the exception, we place in the except clause.



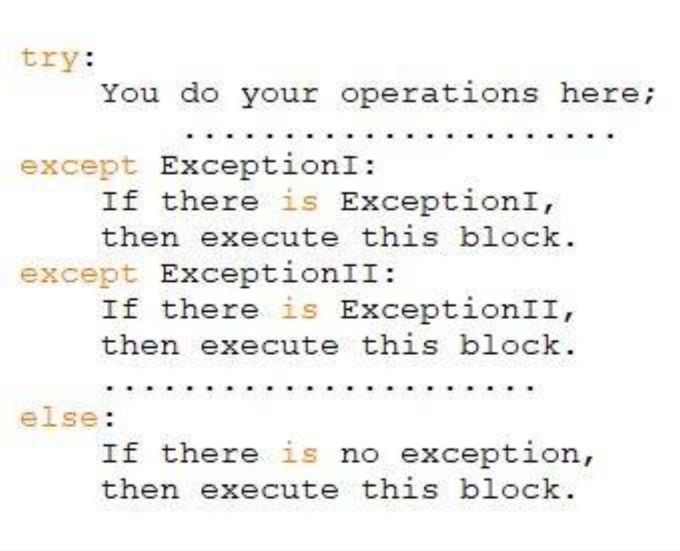
```
try:
    You do your operations here;
       ......................
except ExceptionI:
    If there is ExceptionI,
    then execute this block.
except ExceptionII:
    If there is ExceptionII,
    then execute this block.
    ......................
else:
    If there is no exception,
    then execute this block.
```

Exception Handling

Exception Handling Examples



```
try:
   fob = open("test", "w")
   fob.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can\'t find the file or read data"
else:
   print "Write operation is performed successfully on the file"
   fob.close()
```

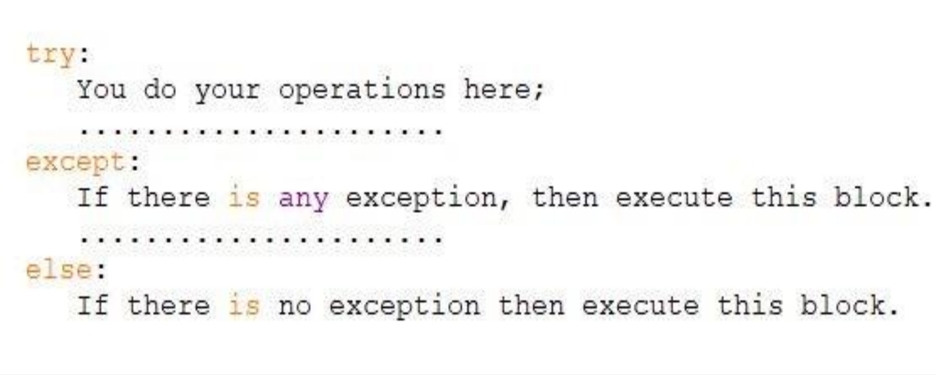


```
try:
    You do your operations here;
       ......................
except ExceptionI:
    If there is ExceptionI,
    then execute this block.
except ExceptionII:
    If there is ExceptionII,
    then execute this block.
    ......................
else:
    If there is no exception,
    then execute this block.
```

Exception Handling

Handling All Types of Exceptions with Except

- If we use a bare “except” clause, then it would catch all types of exceptions.

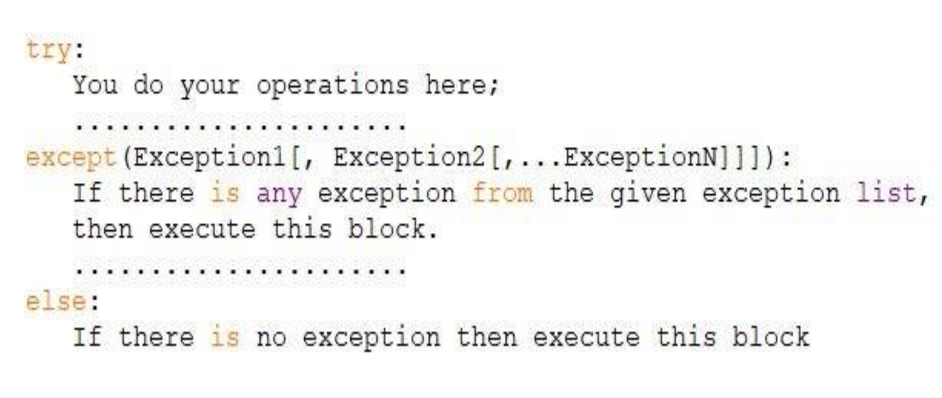


```
try:
   You do your operations here;
   ......................
except:
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Exception Handling

Handling Multiple Exceptions with Except

- We can define multiple exceptions with the same except clause. It means that if the Python interpreter finds a matching exception, then it'll execute the code written under the except clause.

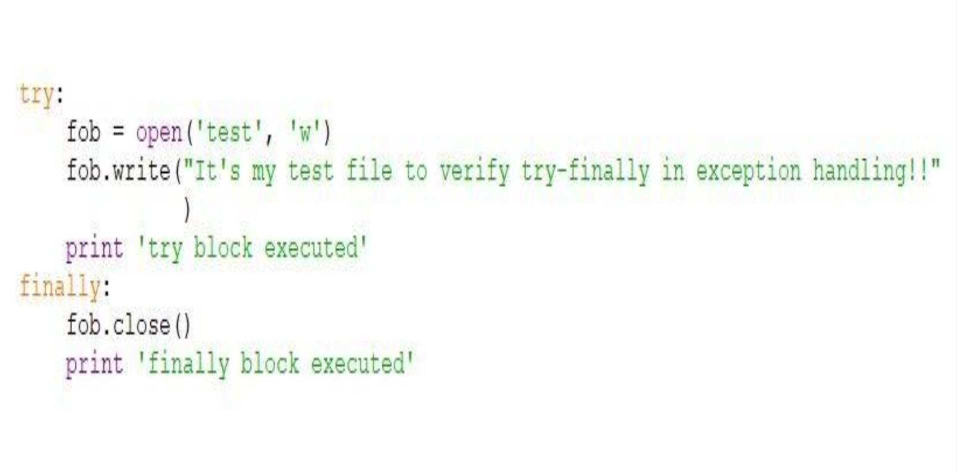


```
try:
    You do your operations here;
    ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
    If there is any exception from the given exception list,
    then execute this block.
    ......................
else:
    If there is no exception then execute this block
```

Exception Handling

Handle Exceptions with Try-Finally

- With try block, we also have the option to define the “finally” block. This clause allows defining statements that we want to execute, no matters whether the try block has raised an exception or not

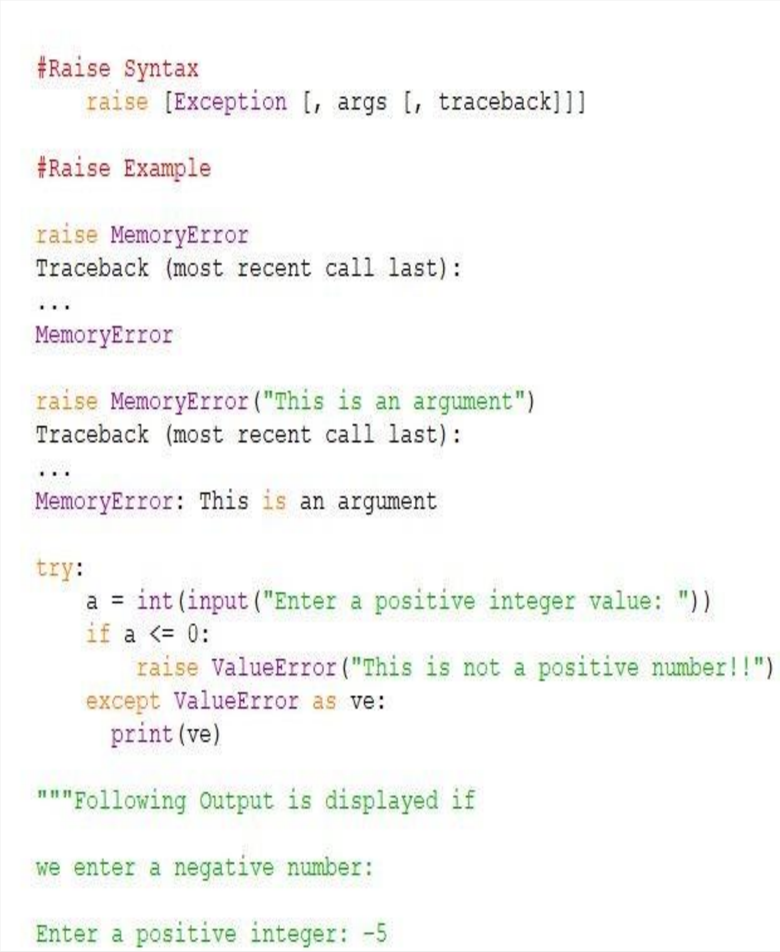


```
try:
    fob = open('test', 'w')
    fob.write("It's my test file to verify try-finally in exception handling!!"
              )
    print 'try block executed'
finally:
    fob.close()
    print 'finally block executed'
```

Exception Handling

Raise Exception with Arguments

- What is Raise?
- We can forcefully raise an exception using the raise keyword.
- We can also optionally pass values to the exception and specify why it has occurred.

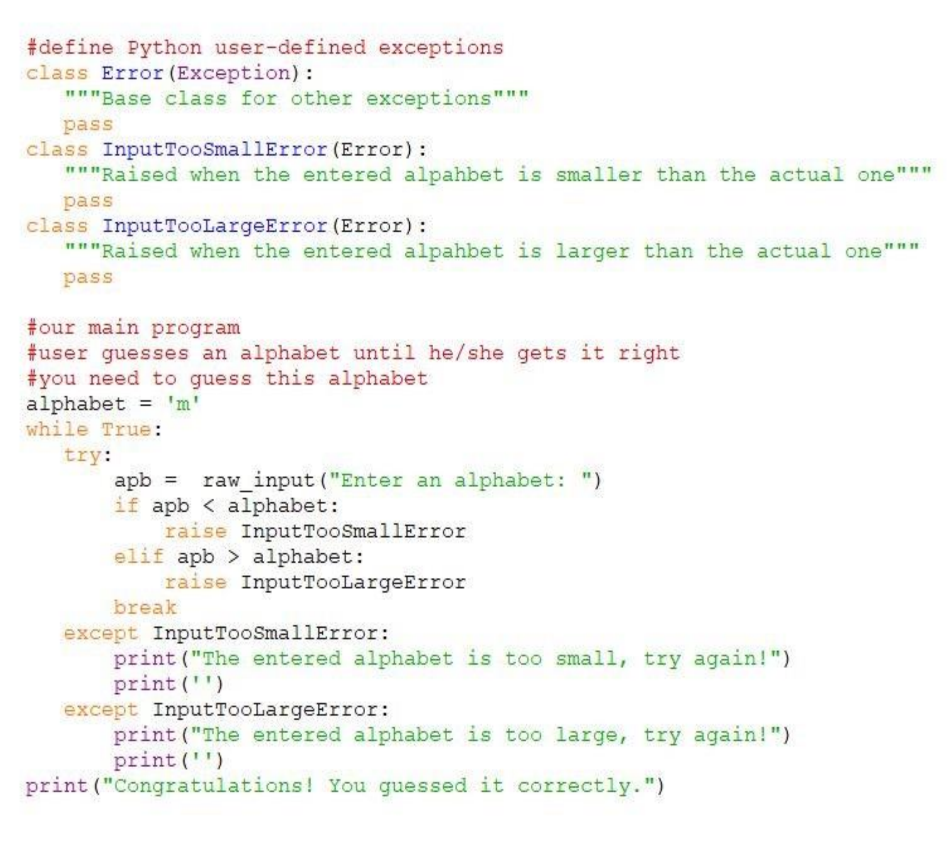


```
#Raise Syntax
    raise [Exception [, args [, traceback]]]

#Raise Example

raise MemoryError
Traceback (most recent call last):
...
MemoryError

raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument

try:
    a = int(input("Enter a positive integer value: "))
    if a <= 0:
        raise ValueError("This is not a positive number!!")
    except ValueError as ve:
      print(ve)

"""Following Output is displayed if

we enter a negative number:

Enter a positive integer: -5
```

Exception Handling

Create Custom Exceptions

- A custom exception is one which the programmer creates himself.
- He does it by adding a new class. The trick here is to derive the custom exception class from the base exception class.
- Most of the built-in exceptions do also have a corresponding class

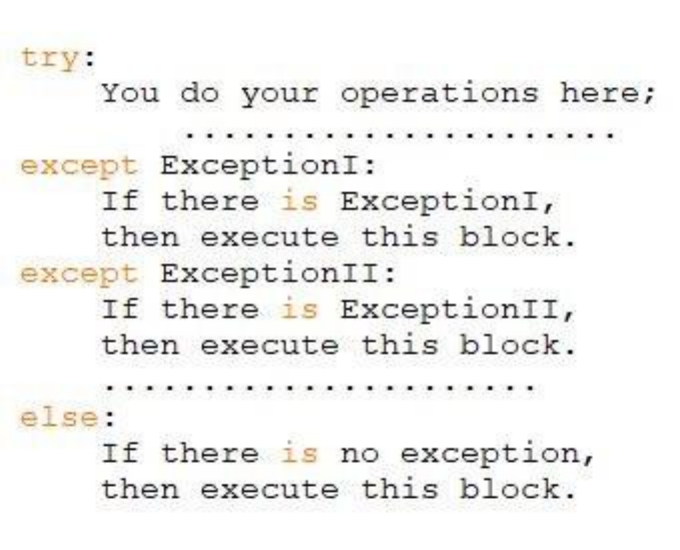


```
#define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass
class InputTooSmallError(Error):
    """Raised when the entered alpahbet is smaller than the actual one"""
    pass
class InputTooLargeError(Error):
    """Raised when the entered alpahbet is larger than the actual one"""
    pass

#our main program
#user guesses an alphabet until he/she gets it right
#you need to guess this alphabet
alphabet = 'm'
while True:
    try:
        apb =  raw_input("Enter an alphabet: ")
        if apb < alphabet:
            raise InputTooSmallError
        elif apb > alphabet:
            raise InputTooLargeError
        break
    except InputTooSmallError:
        print("The entered alphabet is too small, try again!")
        print('')
    except InputTooLargeError:
        print("The entered alphabet is too large, try again!")
        print('')
print("Congratulations! You guessed it correctly.")
```

Exception Handling

Python Built-in Exceptions

- Arithmetic Error
- Assertion Error
- Attribute Error
- EOF Error
- Environment Error
- Floating Point Error
- IO Error
- Memory Error
- Zero Division Error



```
try:
    You do your operations here;
       ......................
except ExceptionI:
    If there is ExceptionI,
    then execute this block.
except ExceptionII:
    If there is ExceptionII,
    then execute this block.
    ......................
else:
    If there is no exception,
    then execute this block.
```

Object Oriented Programming

- Object-oriented programming (OOP) is a method of structuring a program by bundling related properties and behaviors into individual objects.
- Conceptually, objects are like the components of a system. Think of a program as a factory assembly line of sorts. At each step of the assembly line a system component processes some material, ultimately transforming raw material into a finished product.
- An object contains data, like the raw or preprocessed materials at each step on an assembly line, and behavior, like the action each assembly line component performs.

Image source - https://images.app.goo.gl/aWykp4Cd11vrda3k8

OOPS in Python

- Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- For instance, an object could represent a person with properties like a name, age, and address and behaviors such as walking, talking, breathing, and running. Or it could represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.

Image source - https://images.app.goo.gl/aWykp4Cd11vrda3k8

OOPS in Python

- Put another way, object-oriented programming is an approach for modeling concrete, real-world things, like cars, as well as relations between things, like companies and employees, students and teachers, and so on. OOP models real-world entities as software objects that have some data associated with them and can perform certain functions.
- Another common programming paradigm is procedural programming, which structures a program like a recipe in that it provides a set of steps, in the form of functions and code blocks, that flow sequentially in order to complete a task.

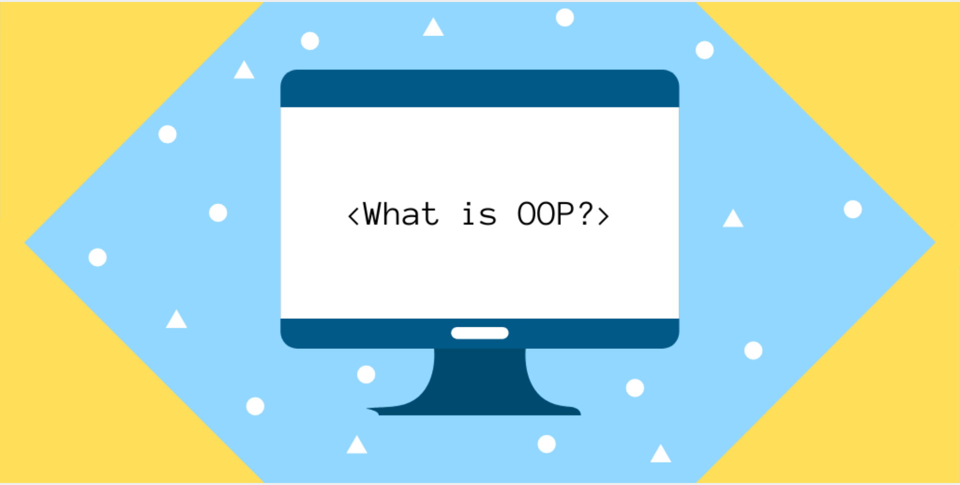


Image source - https://images.app.goo.gl/6X4t6Kvg3XSFVgtU6

OOPS in Python

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

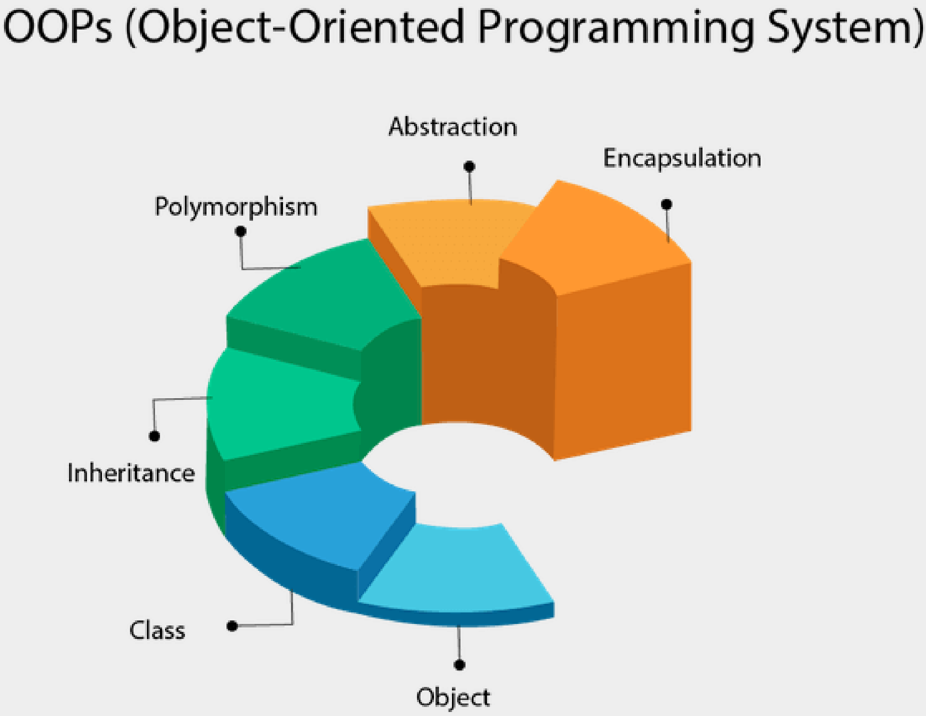


Image source- https://images.app.goo.gl/gNm4QJbMV9WwXYeL6

Class in Python

- Primitive data structures—like numbers, strings, and lists—are designed to represent simple pieces of information, such as the cost of an apple, the name of a poem, or your favorite colors, respectively. What if you want to represent something more complex?
- For example, let’s say you want to track employees in an organization. You need to store some basic information about each employee, such as their name, age, position, and the year they started working.

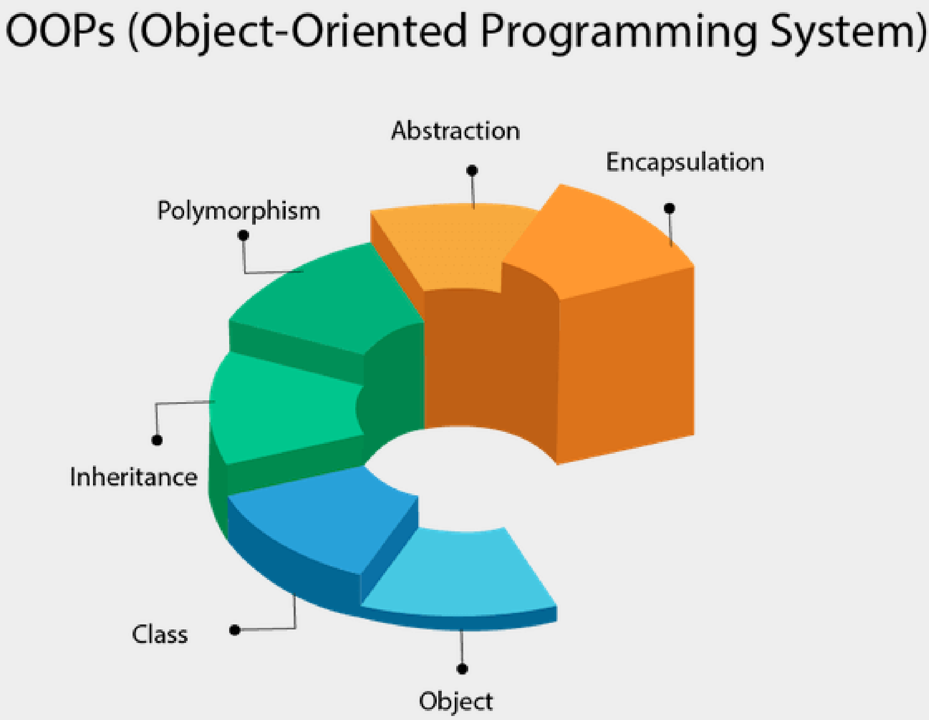
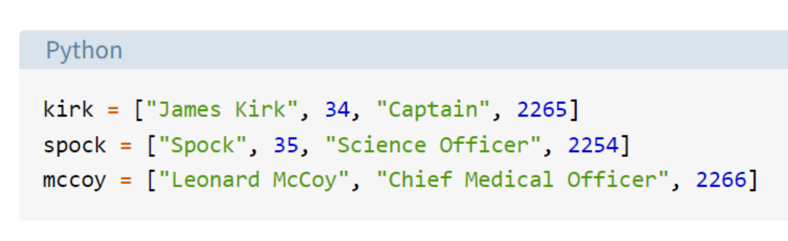


Image source- https://images.app.goo.gl/gNm4QJbMV9WwXYeL6

Class in Python

- There are a number of issues with this approach.
- First, it can make larger code files more difficult to manage. If you reference kirk[0] several lines away from where the kirk list is declared, will you remember that the element with index 0 is the employee’s name?
- Second, it can introduce errors if not every employee has the same number of elements in the list. In the mccoy list above, the age is missing, so mccoy[1] will return "Chief Medical Officer" instead of Dr. McCoy’s age.
- A great way to make this type of code more manageable and more maintainable is to use classes.



```
kirk = ["James Kirk", 34, "Captain", 2265]
spock = ["Spock", 35, "Science Officer", 2254]
mccoy = ["Leonard McCoy", "Chief Medical Officer", 2266]
```

Classes vs Instances

- Classes are used to create user-defined data structures. Classes define functions called methods, which identify the behaviors and actions that an object created from the class can perform with its data.

- Now create a Dog class that stores some information about the characteristics and behaviors that an individual dog can have.

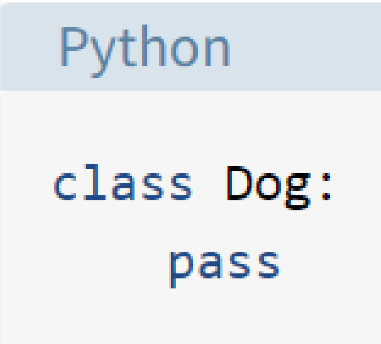
- A class is a blueprint for how something should be defined. It doesn’t actually contain any data. The Dog class specifies that a name and an age are necessary for defining a dog, but it doesn’t contain the name or age of any specific dog.

- While the class is the blueprint, an instance is an object that is built from a class and contains real data. An instance of the Dog class is not a blueprint anymore. It’s an actual dog with a name, like Miles, who’s four years old.

How to define a class

- All class definitions start with the class keyword, which is followed by the name of the class and a colon. Any code that is indented below the class definition is considered part of the class's body.
- Here's an example of a Dog class:

  ```
  class Dog:
          pass
  ```
- The body of the Dog class consists of a single statement: the pass keyword. pass is often used as a placeholder indicating where code will eventually go. It allows you to run this code without Python throwing an error.



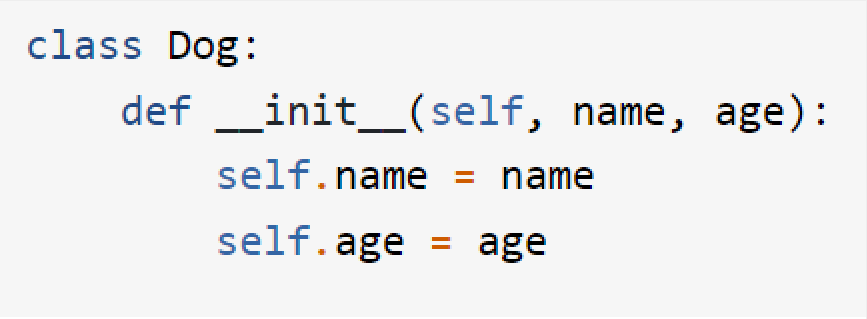
Python

```
class Dog:
    pass
```

How to define a class

- The Dog class isn’t very interesting right now, so let’s spruce it up a bit by defining some properties that all Dog objects should have. There are a number of properties that we can choose from, including name, age, coat color, and breed. To keep things simple, we’ll just use name and age.

- The properties that all Dog objects must have are defined in a method called .__init__(). Every time a new Dog object is created, .__init__() sets the initial state of the object by assigning the values of the object’s properties. That is, .__init__() initializes each new instance of the class.



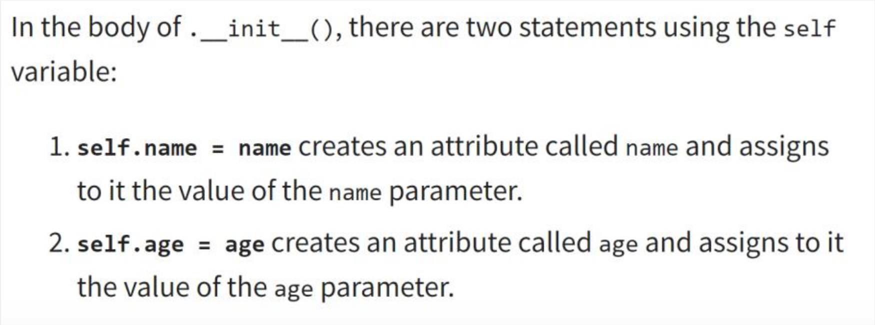
```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

How to define a class

- In the body of .__init__(), there are two statements using the self variable:

- Attributes created in .__init__() are called instance attributes. An instance attribute’s value is specific to a particular instance of the class. All Dog objects have a name and an age, but the values for the name and age attributes will vary depending on the Dog instance.

- On the other hand, class attributes are attributes that have the same value for all class instances. You can define a class attribute by assigning a value to a variable name outside of .__init__().



In the body of `.__init__()`, there are two statements using the `self` variable:

1. **`self.name = name`** creates an attribute called `name` and assigns to it the value of the `name` parameter.
2. **`self.age = age`** creates an attribute called `age` and assigns to it the value of the `age` parameter.

Encapsulation in Python

- Encapsulation is one of the fundamental concepts in object-oriented programming (OOP).
- It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data.
- To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

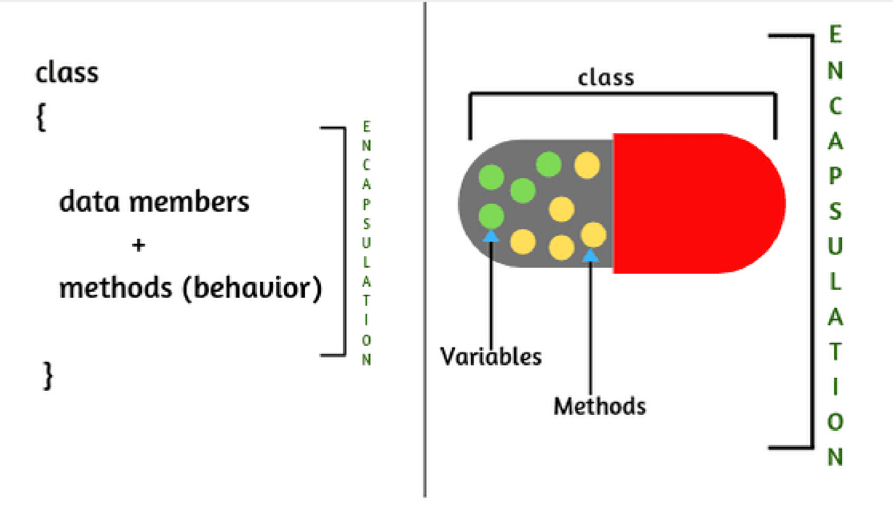


Fig: Encapsulation

Image source -https://images.app.goo.gl/gjCCBpgATvRdmFNA7

Inheritance in Python

- Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more.

- Inheritance is a powerful feature in object oriented programming.

- It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

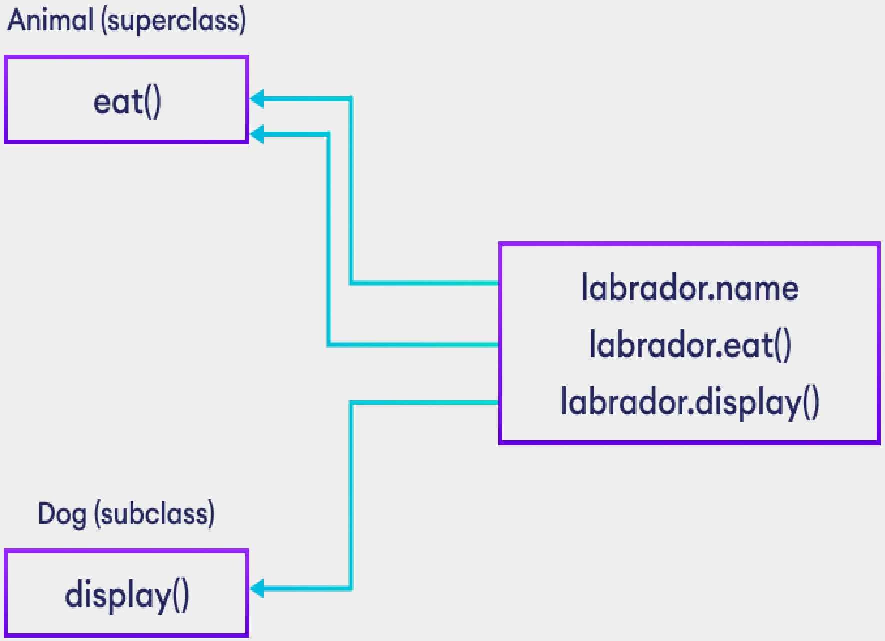
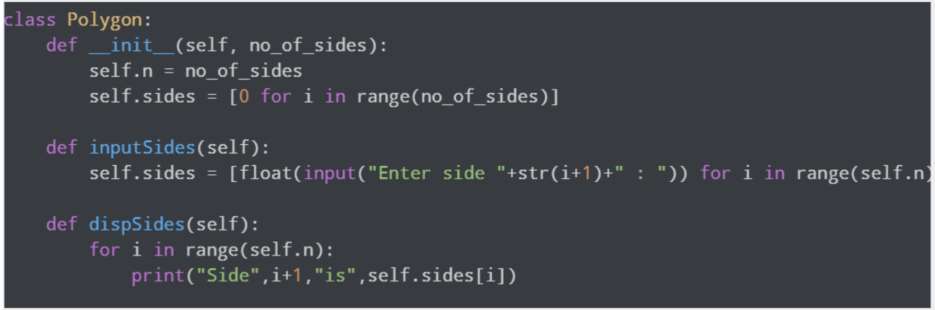


Image source -https://images.app.goo.gl/UGsMbVSTVbetG8B1A

Example of Inheritance

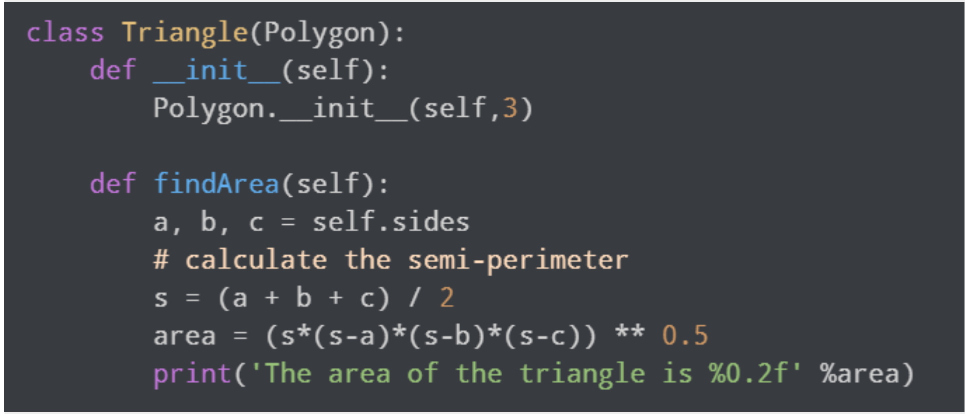
- To demonstrate the use of inheritance, let us take an example.
- A polygon is a closed figure with 3 or more sides. Say, we have a class called Polygon defined as follows.
- This class has data attributes to store the number of sides n and magnitude of each side as a list called sides.
- The inputSides() method takes in the magnitude of each side and dispSides() displays these side lengths.
- A triangle is a polygon with 3 sides. So, we can create a class called Triangle which inherits from Polygon. This makes all the attributes of Polygon class available to the Triangle class.



```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]

    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)

    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
```

Example of Inheritance

- We don't need to define them again (code reusability).
- Triangle can be defined as follows.



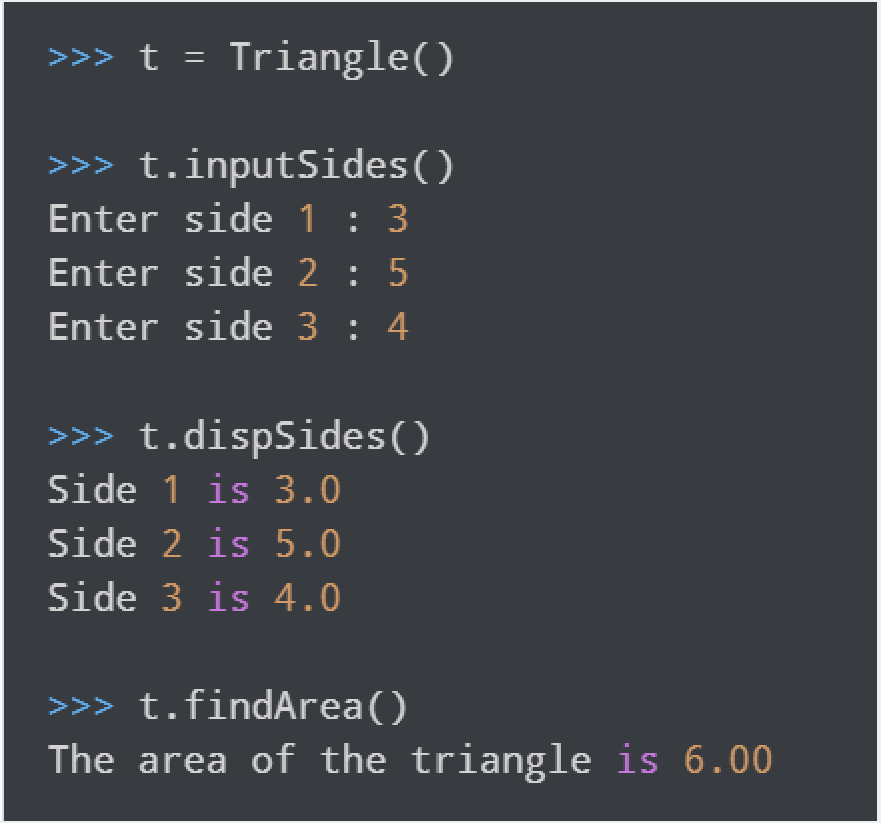
```
class Triangle(Polygon):
    def __init__(self):
        Polygon.__init__(self,3)

    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f' %area)
```

Example of Inheritance

- However, class Triangle has a new method findArea() to find and print the area of the triangle. Here is a sample run.

- We can see that even though we did not define methods like inputSides() or dispSides() for class Triangle separately, we were able to use them.

- If an attribute is not found in the class itself, the search continues to the base class. This repeats recursively, if the base class is itself derived from other classes.



```
>>> t = Triangle()

>>> t.inputSides()
Enter side 1 : 3
Enter side 2 : 5
Enter side 3 : 4

>>> t.dispSides()
Side 1 is 3.0
Side 2 is 5.0
Side 3 is 4.0

>>> t.findArea()
The area of the triangle is 6.00
```

Polymorphism

- In Python, Polymorphism lets us define methods in the child class that have the same name as the methods in the parent class.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee.

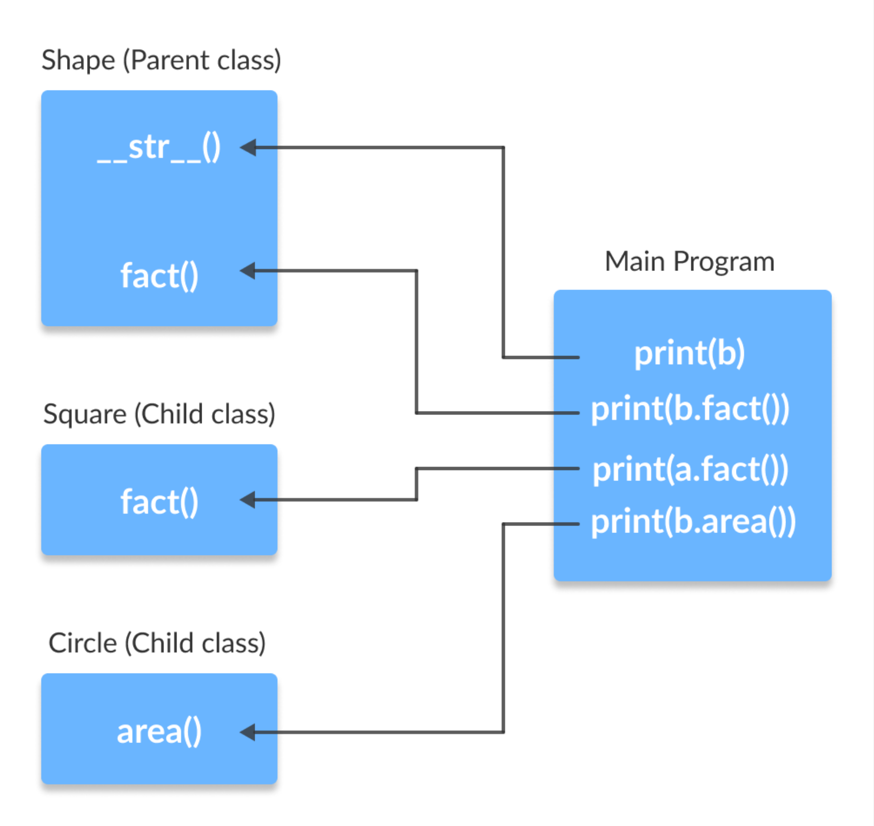
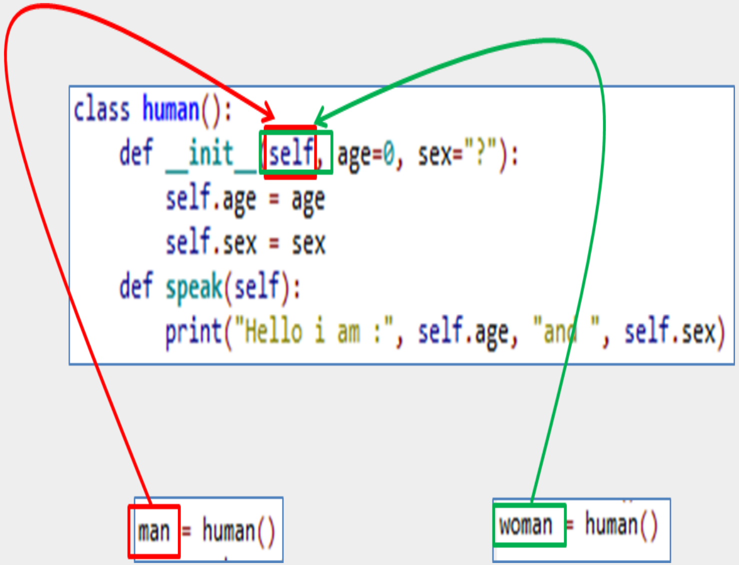


Image source -https://images.app.goo.gl/4n9APBRvUHQqBqor8

Self Parameter

- Methods or functions should have self as first parameter.
- When objects are instantiated, the object itself is passed into the self parameter.
- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.



```
class human():
    def __init__(self, age=0, sex="?"):
        self.age = age
        self.sex = sex
    def speak(self):
        print("Hello i am :", self.age, "and ", self.sex)
```

```
man = human()
```

```
woman = human()
```

Image source -https://images.app.goo.gl/53w7TN9iWnHjbuZE6

Returning Values

- A return statement is used to end the execution of the function call .
- It “returns” the result (value of the expression following the return keyword) to the caller.

```
class student:
        def details(self,n,a):
                self.name=n
                self.age=a
        def display(self):
                return(self.name,self.age)
s=student()
s.details (“xyz”,30)
m,n=s.display()
print(“details are”,m,n)
```

Output

```
details are xyz 30
```

Instances as return values

```
class student:
        def details(self,n,a):
                self.name=n
                self.age=a
        def display(self):
                return(self)
s=student()
s.details (“xyz”,30)
s1=s.display()
print(“details are “, s1.name,s1.age)
```

Output

```
details are xyz 30
```

Constructors

- It is a special method that is automatically invoked right after a new object is created.
- It is used to initialize the attribute values of new object created.
- __init__() is a reserved method in python classes. It is called as a constructor in object oriented terminology.
- This method is called when an object is created from a class and it allows the class to initialize the attributes of the class

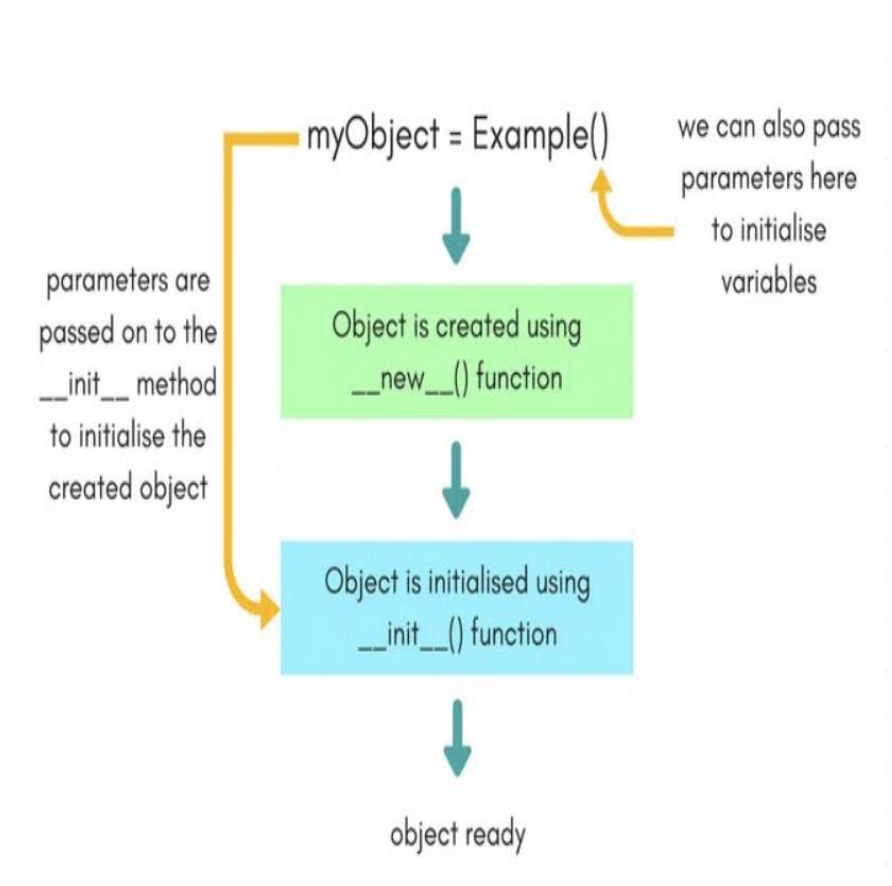

myObject = Example()
we can also pass parameters here to initialise variables
parameters are passed on to the __init__ method to initialise the created object
Object is created using __new__() function
Object is initialised using __init__() function
object ready

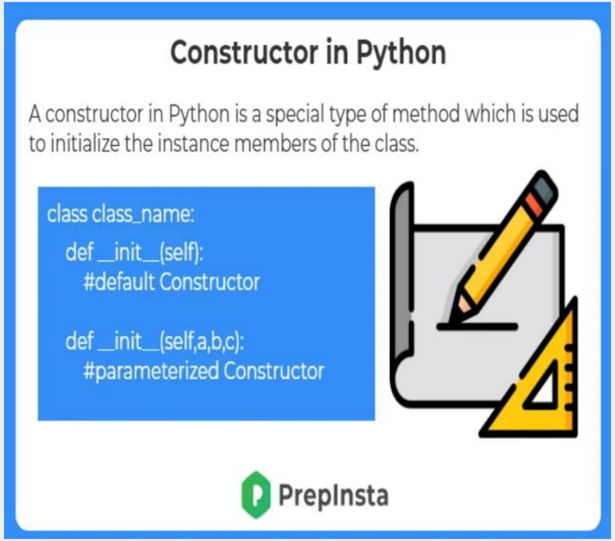
Image Source: https://www.studytonight.com/python/constructors-in-python

Syntax for constructor declaration

```
def __init__(self):
            # body  of constructor
```

Constructor Types

- Default Constructor
- Doesn’t have any arguments
- It has only one argument which is a reference to the instance being constructed
- Parameterized Constructor
- constructor with parameters
- first argument is reference to instance



Constructor in Python

A constructor in Python is a special type of method which is used to initialize the instance members of the class.

```
class class_name:
  def __init__(self):
    #default Constructor

  def __init__(self,a,b,c):
    #parameterized Constructor
```

PrepInsta

Image source- https://images.app.goo.gl/FWHxRVxD7qWEHy9R8

Example – Default Constructor

```
class student:
    def __init__(self):
                self.name=“xyz”
                self.age=30
                def display(self):
                                print(“details are”,self.name,self.age)
     s=student(“xyz”,30)
     s.display()
```

Output

```
details are xyz 30
```

Example – Parameterized Constructor

```
class student:
        def __init__(self,n,a):
                self.name=n
                self.age=a
        def display(self):
                print(“details are”,self.name,self.age)
        s=student(“xyz”,30)
        s.display()
```

Output

```
details are xyz 30
```

Class variables and Instance Variables

- Class Variables — Declared inside the class definition (but outside any of the instance methods). They are not tied to any particular object of the class, hence shared across all the objects of the class. Modifying a class variable affects all objects instance at the same time.
- Instance Variable — Declared inside the constructor method of class (the __init__ method). They are tied to the particular object instance of the class, hence the contents of an instance variable are completely independent from one object instance to the other.

```
class Car:
    wheels = 4 # Class variable
    def __init__(self, name):
        self.name = name  #Instance variable
```

Destructors in Python

- When an object is destroyed, destructors are invoked. Destructors aren't as important in Python as they are in C++ because Python has a garbage collector that handles memory management for you.

- In Python, the del () function is known as a destructor method. It is called after all references to the object have been destroyed i.e when an object is garbage collected.
- Syntax:

```
def __del__(self):
            # body of destructor
```

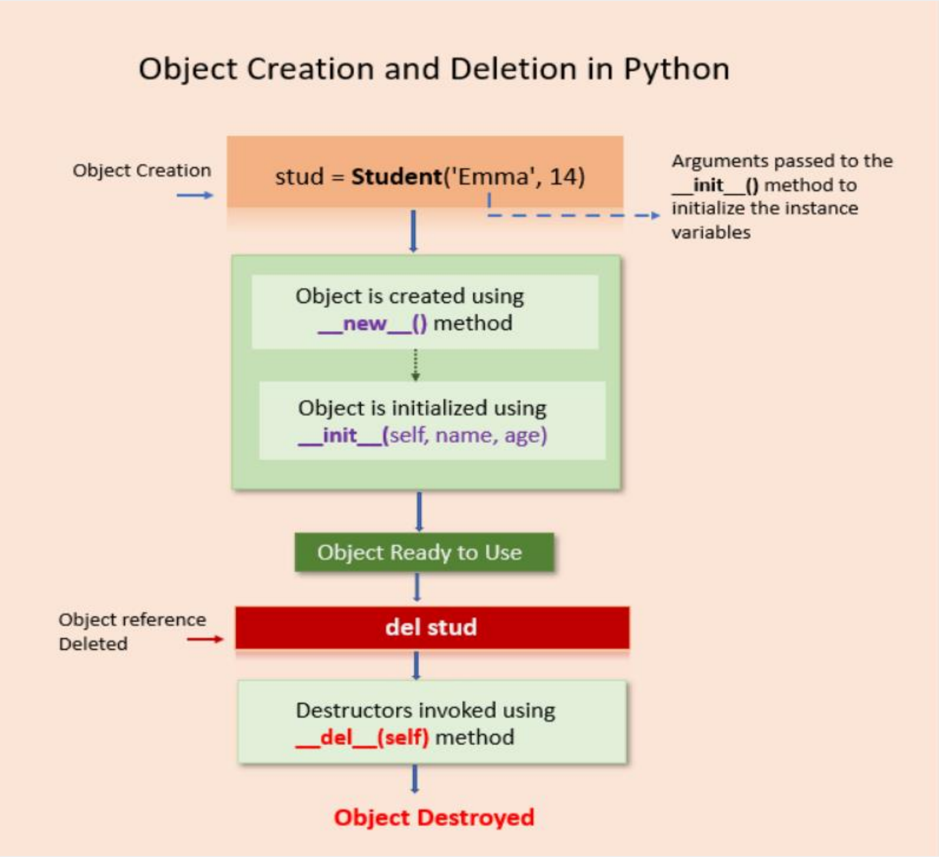

Object Creation and Deletion in Python
Object Creation
stud = Student('Emma', 14)
Arguments passed to the __init__() method to initialize the instance variables
Object is created using __new__() method
Object is initialized using __init__(self, name, age)
Object Ready to Use
Object reference Deleted
del stud
Destructors invoked using __del__(self) method
Object Destroyed

Image source -https://images.app.goo.gl/CPtf96eobgf24M3MA

Database

MySQL Database

- MySQL is one of the most popular database management systems (DBMSs) on the market today.
- It ranked second only to the Oracle DBMS in this year’s DB-Engines Ranking.
- As most software applications need to interact with data in some form, programming languages like Python provide tools for storing and accessing these data sources.

Database

MySQL Database

- Being open source since its inception in 1995, MySQL quickly became a market leader among SQL solutions.
- MySQL is also a part of the Oracle ecosystem.
- While its core functionality is completely free, there are some paid add-ons as well.
- Currently, MySQL is used by all major tech firms, including Google, LinkedIn, Uber, Netflix, Twitter, and others.

Database

MySQL Database

- Ease of installation: MySQL was designed to be user-friendly. It’s quite straightforward to set up a MySQL database, and several widely available third-party tools, like phpMyAdmin, further streamline the setup process. MySQL is available for all major operating systems, including Windows, macOS, Linux, and Solaris.

- Speed: MySQL holds a reputation for being an exceedingly fast database solution. It has a relatively smaller footprint and is extremely scalable in the long run.

- User privileges and security: MySQL comes with a script that allows you to set the password security level, assign admin passwords, and add and remove user account privileges. This script uncomplicates the admin process for a web hosting user management portal. Other DBMSs, like PostgreSQL, use config files that are more complicated to use.

Installing MySQL Connector/Python

MySQL Database

- A database driver is a piece of software that allows an application to connect and interact with a database system. Programming languages like Python need a special driver before they can speak to a database from a specific vendor.

- In Python you need to install a Python MySQL connector to interact with a MySQL database. Many packages follow the DB-API standards, but the most popular among them is MySQL Connector/Python. You can get it with pip:

- pip install mysql-connector-python

Installing MySQL Connector/Python

MySQL Database

- To test if the installation was successful, type the following command on your Python terminal:

- `import mysql.connector`

- If the above code executes with no errors, then mysql.connector is installed and ready to use. If you encounter any errors, then make sure you’re in the correct virtual environment and you’re using the right Python interpreter.

- Make sure that you’re installing the correct mysql-connector-python package, which is a pure-Python implementation. Beware of similarly named but now depreciated connectors like mysql-connector.

Establishing a Connection With MySQL Server

MySQL Database

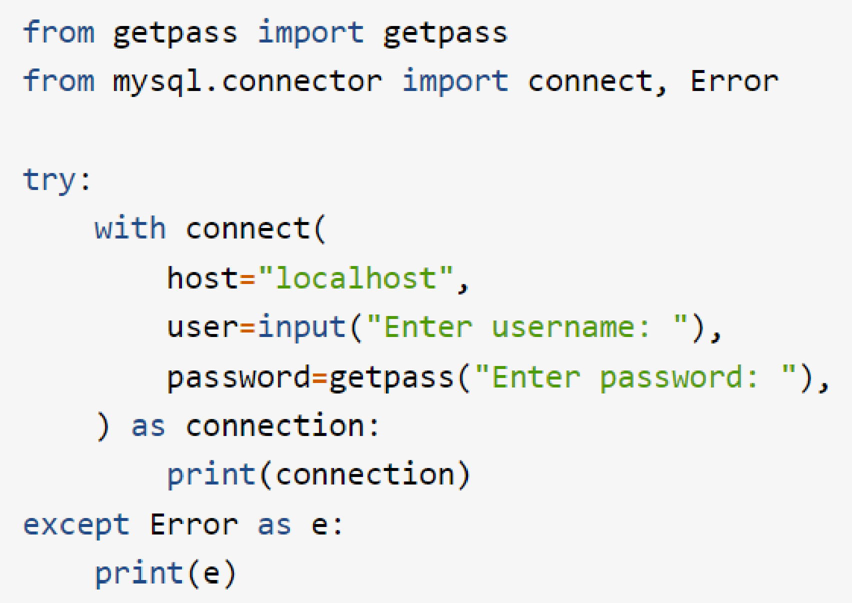
- MySQL is a server-based database management system. One server might contain multiple databases. To interact with a database, you must first establish a connection with the server. The general workflow of a Python program that interacts with a MySQL-based database is as follows:

- Connect to the MySQL server.
- Create a new database.
- Connect to the newly created or an existing database.
- Execute a SQL query and fetch results.
- Inform the database if any changes are made to a table.
- Close the connection to the MySQL server.
- This is a generic workflow that might vary depending on the individual application. But whatever the application might be, the first step is to connect your database with your application.

Establishing a Connection With MySQL Server

MySQL Database

- The first step in interacting with a MySQL server is to establish a connection.
- To do this, you need connect() from the mysql.connector module.
- This function takes in parameters like host, user, and password and returns a MySQLConnection object.
- You can receive these credentials as input from the user and pass them to connect():



```python
from getpass import getpass
from mysql.connector import connect, Error

try:
    with connect(
        host="localhost",
        user=input("Enter username: "),
        password=getpass("Enter password: "),
    ) as connection:
        print(connection)
except Error as e:
    print(e)
```

Establishing a Connection With MySQL Server

MySQL Database

- There are several important things to notice in the code above:
- You should always deal with the exceptions that might be raised while establishing a connection to the MySQL server. This is why you use a try … except block to catch and print any exceptions that you might encounter.
- You should always close the connection after you’re done accessing the database. Leaving unused open connections can lead to several unexpected errors and performance issues.
- You should never hard-code your login credentials, that is, your username and password, directly in a Python script. This is a bad practice for deployment and poses a serious security threat. The code above prompts the user for login credentials. It uses the built-in getpass module to hide the password. While this is better than hard-coding, there are other, more secure ways to store sensitive information, like using environment variables.

Creating a new database

MySQL Database

- To create a new database, you need to execute a SQL statement:

- CREATE DATABASE books_db;
- The above statement will create a new database with the name books_db.

Creating a new database

MySQL Database

- To execute a SQL query in Python, you’ll need to use a cursor, which abstracts away the access to database records.
- MySQL Connector/Python provides you with the MySQLCursor class, which instantiates objects that can execute MySQL queries in Python.
- An instance of the MySQLCursor class is also called a cursor.

- Cursor objects make use of a MySQLConnection object to interact with your MySQL server. To create a cursor, use the .cursor() method of your connection variable:

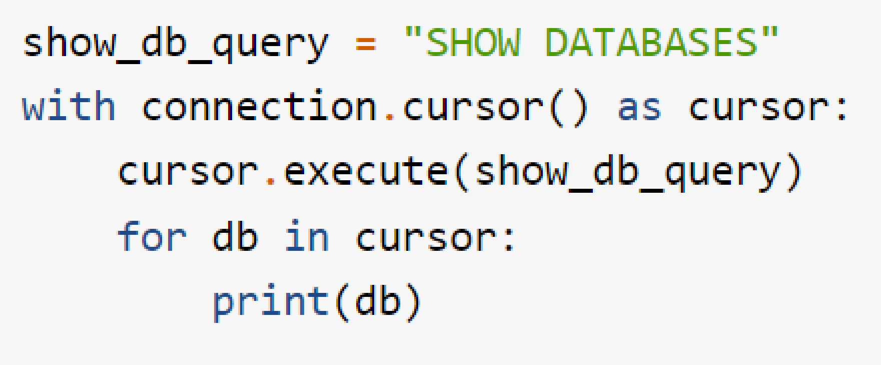
- cursor = connection.cursor()

- The above code gives you an instance of the MySQLCursor class.

Show Database

MySQL Database

- You might receive an error here if a database with the same name already exists in your server.
- To confirm this, you can display the name of all databases in your server.
- Using the same MySQLConnection object from earlier, execute the SHOW DATABASES statement:

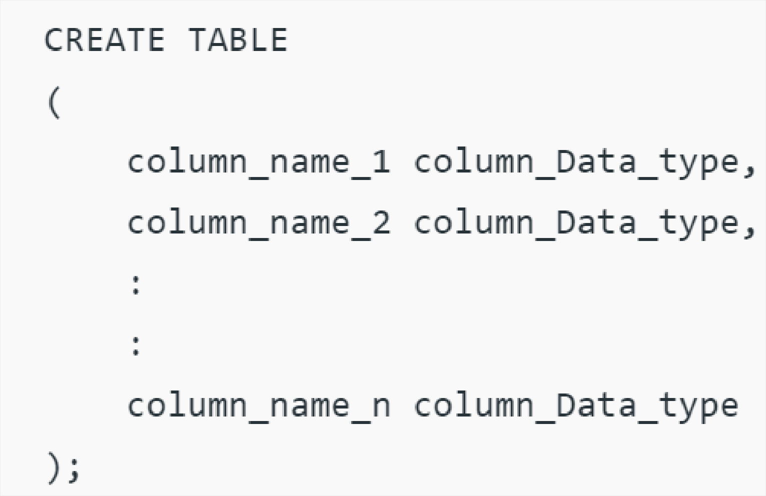


```
show_db_query = "SHOW DATABASES"
with connection.cursor() as cursor:
    cursor.execute(show_db_query)
    for db in cursor:
        print(db)
```

Creating Tables

MySQL Database

- For creating tables we will follow the similar approach of writing the SQL commands as strings and then passing it to the execute() method of the cursor object.
- SQL command for creating a table is –

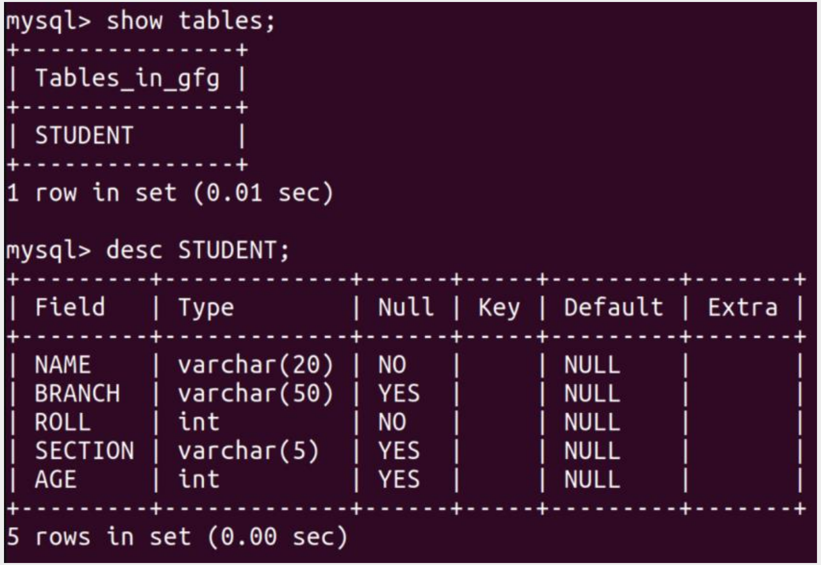


```
CREATE TABLE
(
    column_name_1 column_Data_type,
    column_name_2 column_Data_type,
    :
    :
    column_name_n column_Data_type
);
```

Creating Tables

MySQL Database

```
import mysql.connector
dataBase = mysql.connector.connect(host
="localhost",user ="user",passwd ="password",
database = "gfg")
cursorObject = dataBase.cursor()
studentRecord = """CREATE TABLE STUDENT
(NAME VARCHAR(20) NOT NULL, BRANCH
VARCHAR(50), ROLL INT NOT NULL,
SECTION VARCHAR(5), AGE INT)"""
# table created
cursorObject.execute(studentRecord)
dataBase.close()
```



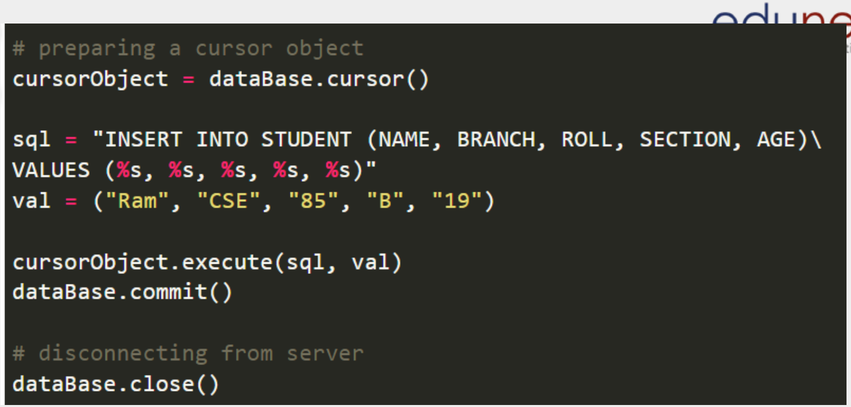
```
mysql> show tables;
+---------------+
| Tables_in_gfg |
+---------------+
| STUDENT       |
+---------------+
1 row in set (0.01 sec)

mysql> desc STUDENT;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| NAME    | varchar(20) | NO   |     | NULL    |       |
| BRANCH  | varchar(50) | YES  |     | NULL    |       |
| ROLL    | int         | NO   |     | NULL    |       |
| SECTION | varchar(5)  | YES  |     | NULL    |       |
| AGE     | int         | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Insert Data into tables

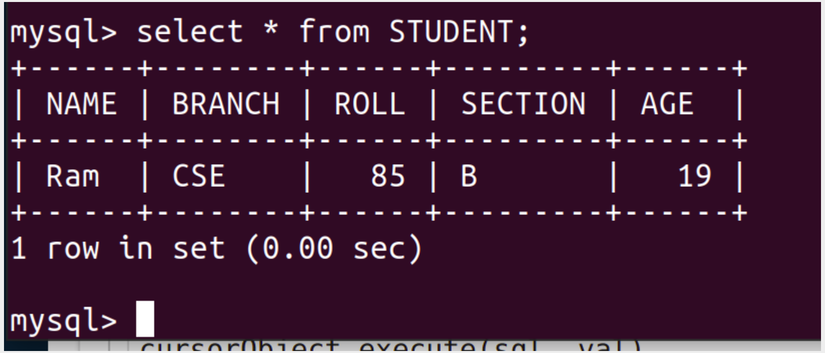
- To insert data into the MySQL table Insert into query is used.

- Syntax:

INSERT INTO table_name (column_names)
VALUES (data)



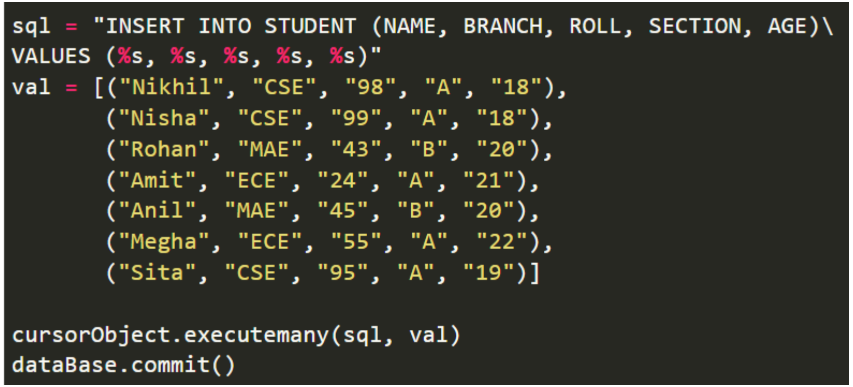
```
# preparing a cursor object
cursorObject = dataBase.cursor()

sql = "INSERT INTO STUDENT (NAME, BRANCH, ROLL, SECTION, AGE)\
VALUES (%s, %s, %s, %s, %s)"
val = ("Ram", "CSE", "85", "B", "19")

cursorObject.execute(sql, val)
dataBase.commit()

# disconnecting from server
dataBase.close()
```



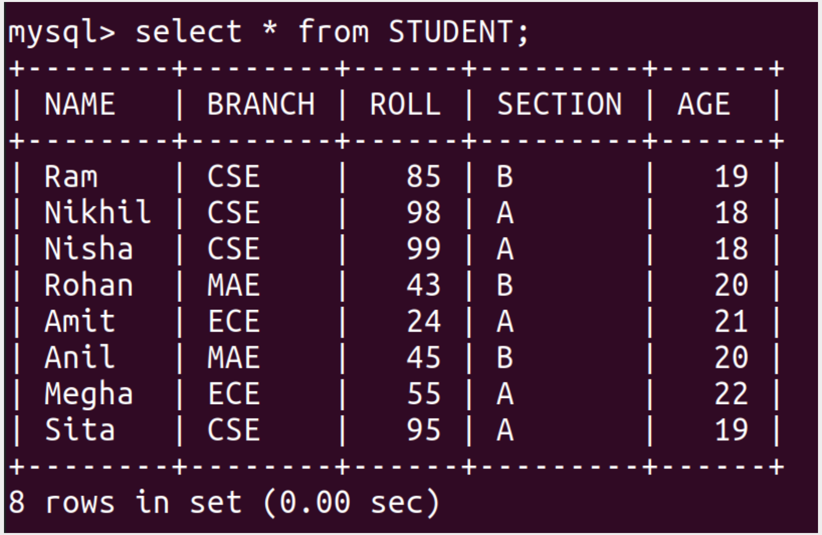
```
mysql> select * from STUDENT;
+------+--------+------+---------+------+
| NAME | BRANCH | ROLL | SECTION | AGE  |
+------+--------+------+---------+------+
| Ram  | CSE    |   85 | B       |   19 |
+------+--------+------+---------+------+
1 row in set (0.00 sec)

mysql> 
```

Inserting Multiple Rows

- To insert multiple values at once, executemany() method is used. This method iterates through the sequence of parameters, passing the current parameter to the execute method.



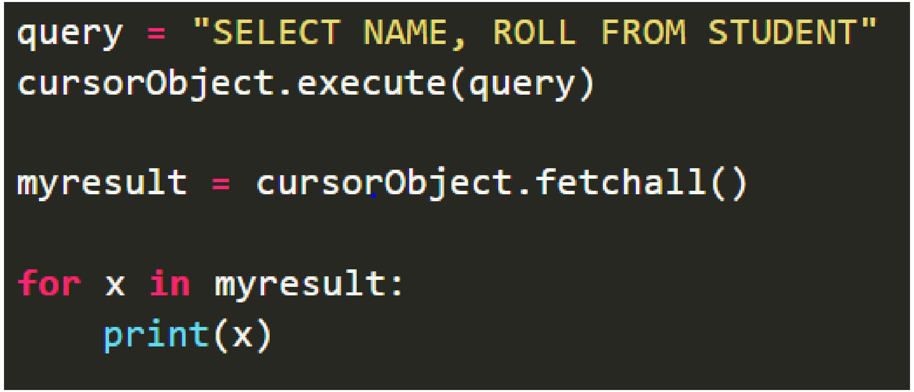
```
sql = "INSERT INTO STUDENT (NAME, BRANCH, ROLL, SECTION, AGE)\
VALUES (%s, %s, %s, %s, %s)"
val = [("Nikhil", "CSE", "98", "A", "18"),
       ("Nisha", "CSE", "99", "A", "18"),
       ("Rohan", "MAE", "43", "B", "20"),
       ("Amit", "ECE", "24", "A", "21"),
       ("Anil", "MAE", "45", "B", "20"),
       ("Megha", "ECE", "55", "A", "22"),
       ("Sita", "CSE", "95", "A", "19")]

cursorObject.executemany(sql, val)
dataBase.commit()
```



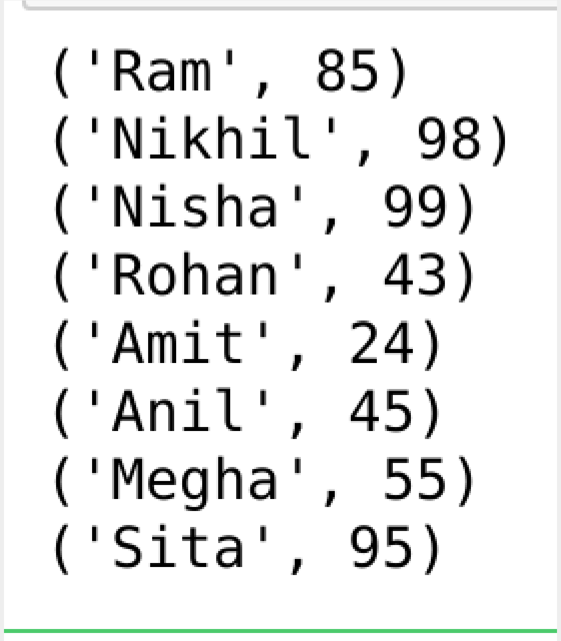
```
mysql> select * from STUDENT;
+--------+--------+------+---------+------+
| NAME   | BRANCH | ROLL | SECTION | AGE  |
+--------+--------+------+---------+------+
| Ram    | CSE    |   85 | B       |   19 |
| Nikhil | CSE    |   98 | A       |   18 |
| Nisha  | CSE    |   99 | A       |   18 |
| Rohan  | MAE    |   43 | B       |   20 |
| Amit   | ECE    |   24 | A       |   21 |
| Anil   | MAE    |   45 | B       |   20 |
| Megha  | ECE    |   55 | A       |   22 |
| Sita   | CSE    |   95 | A       |   19 |
+--------+--------+------+---------+------+
8 rows in set (0.00 sec)
```

Fetching Data

- We can use the select query on the MySQL tables



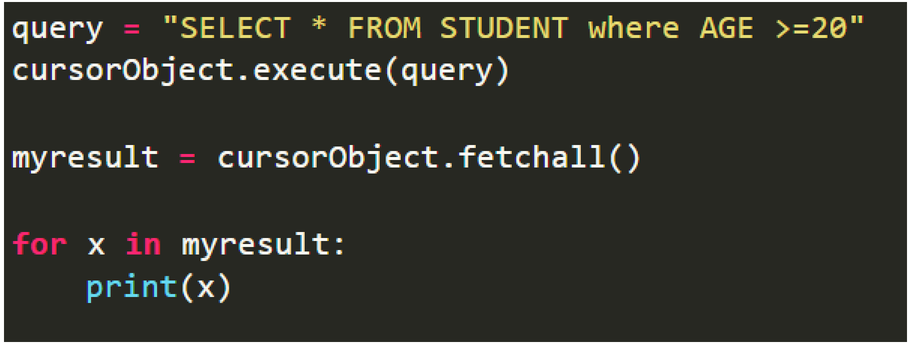
```
query = "SELECT NAME, ROLL FROM STUDENT"
cursorObject.execute(query)

myresult = cursorObject.fetchall()

for x in myresult:
    print(x)
```



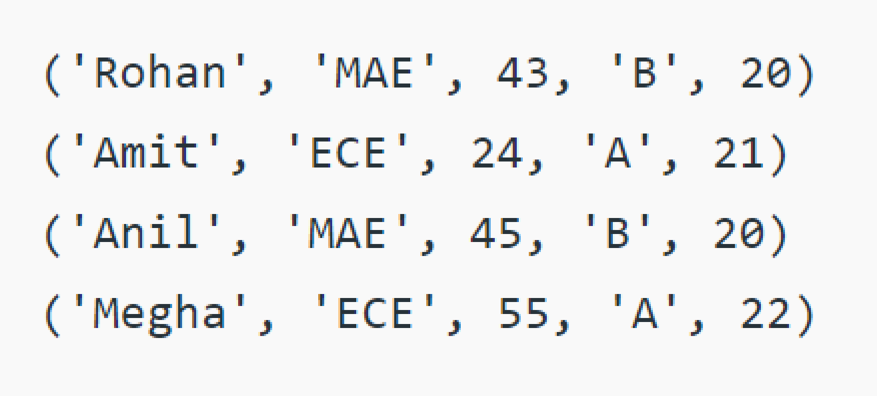
```
('Ram', 85)
('Nikhil', 98)
('Nisha', 99)
('Rohan', 43)
('Amit', 24)
('Anil', 45)
('Megha', 55)
('Sita', 95)
```

Where Clause

- Where clause is used in MySQL database to filter the data as per the condition required.
- You can fetch, delete or update a particular set of data in MySQL database by using where clause.



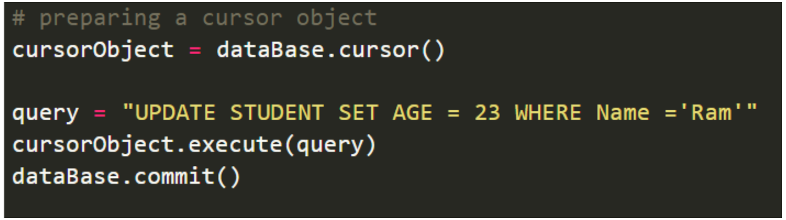
```python
query = "SELECT * FROM STUDENT where AGE >=20"
cursorObject.execute(query)

myresult = cursorObject.fetchall()

for x in myresult:
    print(x)
```



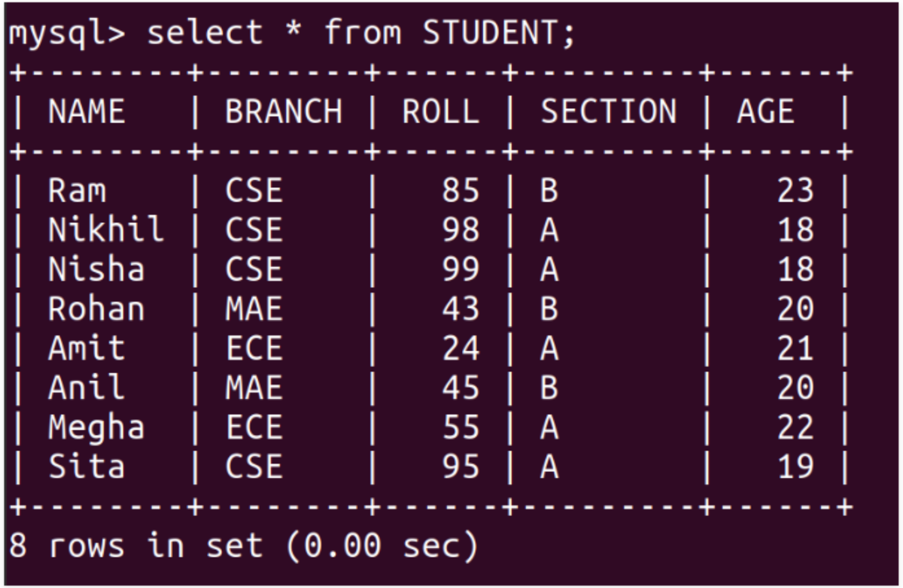
```
('Rohan', 'MAE', 43, 'B', 20)
('Amit', 'ECE', 24, 'A', 21)
('Anil', 'MAE', 45, 'B', 20)
('Megha', 'ECE', 55, 'A', 22)
```

Update Data

- The update query is used to change the existing values in a database. By using update a specific value can be corrected or updated. It only affects the data and not the structure of the table. The basic advantage provided by this command is that it keeps the table accurate.



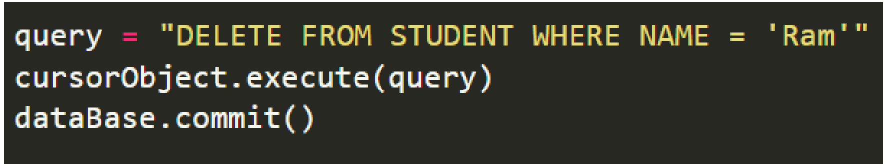
```
# preparing a cursor object
cursorObject = dataBase.cursor()

query = "UPDATE STUDENT SET AGE = 23 WHERE Name ='Ram'"
cursorObject.execute(query)
dataBase.commit()
```



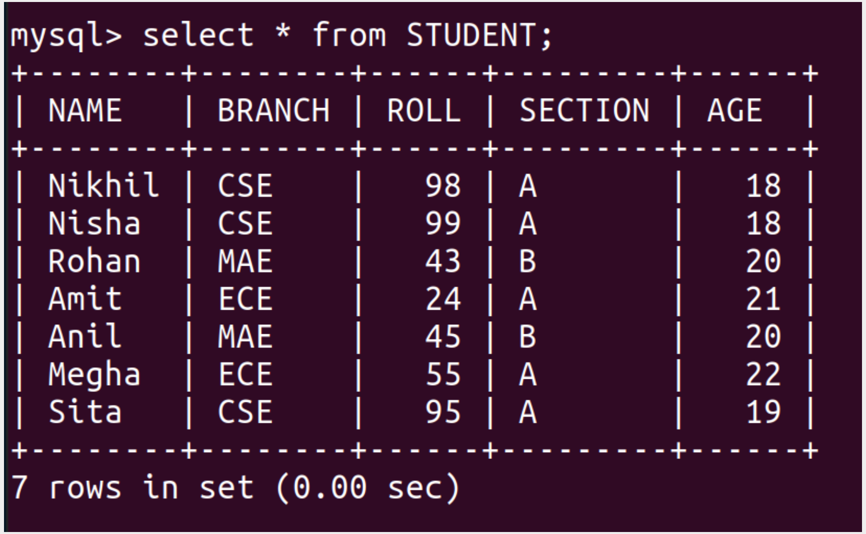
```
mysql> select * from STUDENT;
+--------+--------+------+---------+------+
| NAME   | BRANCH | ROLL | SECTION | AGE  |
+--------+--------+------+---------+------+
| Ram    | CSE    |   85 | B       |   23 |
| Nikhil | CSE    |   98 | A       |   18 |
| Nisha  | CSE    |   99 | A       |   18 |
| Rohan  | MAE    |   43 | B       |   20 |
| Amit   | ECE    |   24 | A       |   21 |
| Anil   | MAE    |   45 | B       |   20 |
| Megha  | ECE    |   55 | A       |   22 |
| Sita   | CSE    |   95 | A       |   19 |
+--------+--------+------+---------+------+
8 rows in set (0.00 sec)
```

Delete Data from Table

- We can use the Delete query to delete data from the table in MySQL.



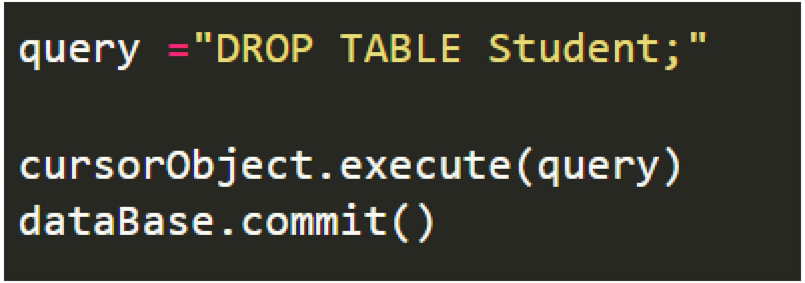
```
query = "DELETE FROM STUDENT WHERE NAME = 'Ram'"
cursorObject.execute(query)
dataBase.commit()
```



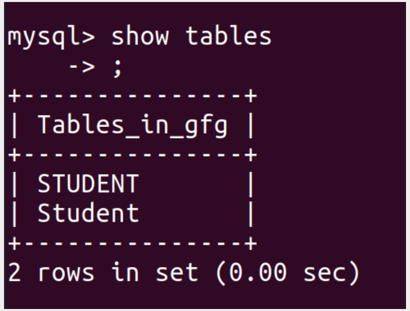
```
mysql> select * from STUDENT;
+--------+--------+------+---------+------+
| NAME   | BRANCH | ROLL | SECTION | AGE  |
+--------+--------+------+---------+------+
| Nikhil | CSE    |   98 | A       |   18 |
| Nisha  | CSE    |   99 | A       |   18 |
| Rohan  | MAE    |   43 | B       |   20 |
| Amit   | ECE    |   24 | A       |   21 |
| Anil   | MAE    |   45 | B       |   20 |
| Megha  | ECE    |   55 | A       |   22 |
| Sita   | CSE    |   95 | A       |   19 |
+--------+--------+------+---------+------+
7 rows in set (0.00 sec)
```

Drop Tables

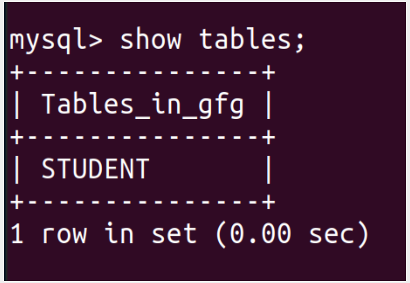
- Drop command affects the structure of the table and not data. It is used to delete an already existing table. For cases where you are not sure if the table to be dropped exists or not DROP TABLE IF EXISTS command is used.



```
query ="DROP TABLE Student;"

cursorObject.execute(query)
dataBase.commit()
```



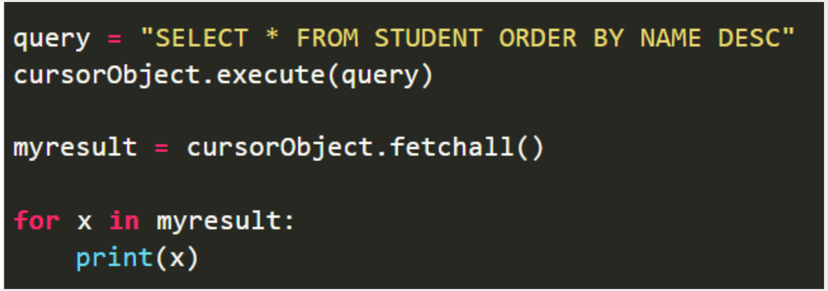
```
mysql> show tables
    -> ;
+---------------+
| Tables_in_gfg |
+---------------+
| STUDENT       |
| Student       |
+---------------+
2 rows in set (0.00 sec)
```



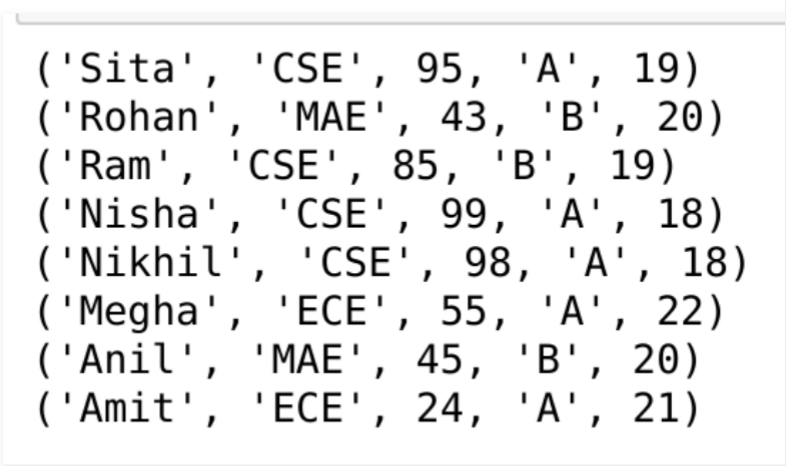
```
mysql> show tables;
+---------------+
| Tables_in_gfg |
+---------------+
| STUDENT       |
+---------------+
1 row in set (0.00 sec)
```

Orberby Clause

- OrderBy is used to arrange the result set in either ascending or descending order.
- By default, it is always in ascending order unless “DESC” is mentioned, which arranges it in descending order.
- “ASC” can also be used to explicitly arrange it in ascending order. But, it is generally not done this way since default already does that.



```
query = "SELECT * FROM STUDENT ORDER BY NAME DESC"
cursorObject.execute(query)

myresult = cursorObject.fetchall()

for x in myresult:
    print(x)
```



```
('Sita', 'CSE', 95, 'A', 19)
('Rohan', 'MAE', 43, 'B', 20)
('Ram', 'CSE', 85, 'B', 19)
('Nisha', 'CSE', 99, 'A', 18)
('Nikhil', 'CSE', 98, 'A', 18)
('Megha', 'ECE', 55, 'A', 22)
('Anil', 'MAE', 45, 'B', 20)
('Amit', 'ECE', 24, 'A', 21)
```

Web Development in Python

Django

features

- Can be used to generate HTML, CSV, Email or any other format
- Supports many databases – Postgresql, MySQL, Oracle, SQLite
- Middleware, csrf protections, sessions, caching, authentication are also included

Image source - http://www.djangoproject.com

Python Flask

Image source -https://images.app.goo.gl/XTCDpKH23xj7DQrk7

Python Flask

- Flask is a lightweight web application framework.
- It is designed to make getting started quick and easy.
- Able to scale ,up to complex applications.
- Flask supports Python 3.7 and newer.

Image source -https://images.app.goo.gl/XTCDpKH23xj7DQrk7

Installation

Create a project folder and a venv folder within:

```
mkdir myproject
cd myproject
py -3 -m venv venv
```

Activate the environment

```
venv\Scripts\activate
```

Install Flask

```
pip install Flask
```

Image source -https://images.app.goo.gl/XTCDpKH23xj7DQrk7

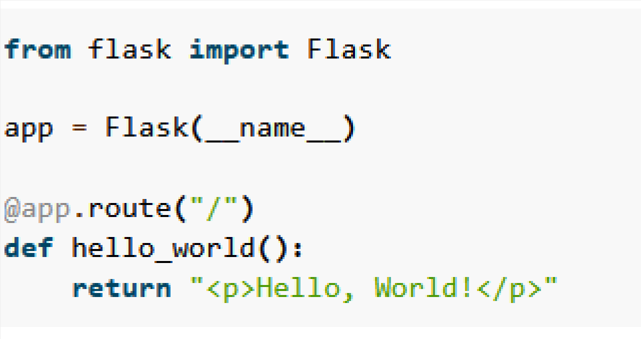
Simple Application

- First you need to import the Flask class.

- After that, we make an instance of the class. __name__ is a handy shortcut for this that works well in most cases.

- The route() decorator is then used to tell Flask which URL should be used to call our function.

- Because HTML is the default content type, the browser will render HTML in the string.



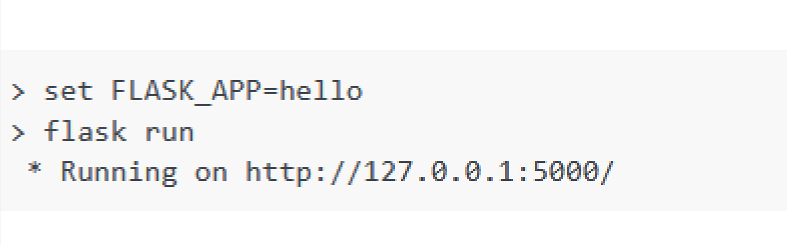
```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Run the Application

We can run the application on command prompt

- ➢set FLASK_APP=hello
- ➢ flask run

  * Running on http://127.0.0.1:5000



```
> set FLASK_APP=hello
> flask run
 * Running on http://127.0.0.1:5000/
```

Python for Web-Django

(60 hours)

Disclaimer: The content is curated for educational purposes only.

In this module, student will learn about:

- Web Framework, Django Introduction, Django Architecture
- Django MVC, MVT (Model View Template)
- Views and URL mapping, HttpRequest and HttpResponse , GET and POST Method
- Template, Render, Views, Context
- Template Editing
- SQL operation in django
- Handling sessions, cookies and working with JSON and AJAX

Web Framework, Django Introduction, Django Architecture

Disclaimer: The content is curated for educational purposes only.

In this sub-section, we will discuss:

- Web Framework
- Django Introduction
- Django Architecture

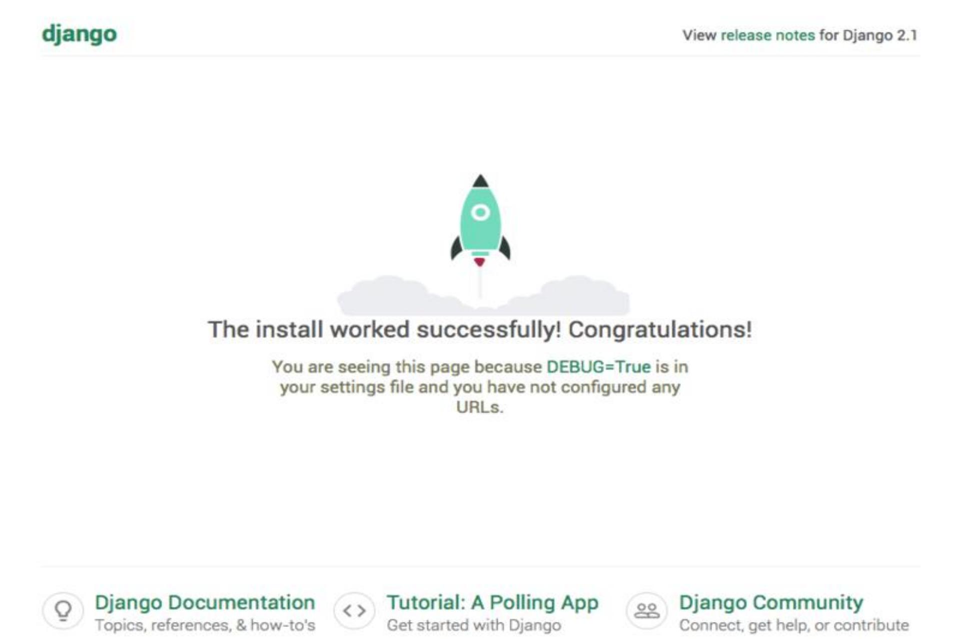
Web Framework

- A web framework (WF) or web application framework (WAF) is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs.
- Web frameworks provide a standard way to build and deploy web applications on the World Wide Web.
- Web frameworks aim to automate the overhead associated with common activities performed in web development.

Image Source:https://www.scnsoft.com/blog/web-application-framework

Introduction to Django

- Django is a Python-based free and open-source web framework that follows the model–views-template(MVT) architectural pattern.
- Django's primary goal is to ease the creation of complex, database-driven websites.
- The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself.



django

View release notes for Django 2.1

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Image Source:
https://en.wikipedia.org/wiki/File:Django_2.1_landing_page.png

Django Features

- Helps you to define patterns for the URLs in your application
- Simple but powerful URL system
- Built-in authentication system
- Object-oriented programming language database which offers best in class data storage and retrieval
- Automatic admin interface feature allows the functionality of adding, editing and deleting items. You can customize the admin panel as per your need.
- It is used for Rapid Development
- Secure
- Open Source
- Vast and Supported Community

Django Architecture

Model-View-Template (MVT) Architecture

- Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application.
- M stands for Model
- V stands for View
- T stands for Template

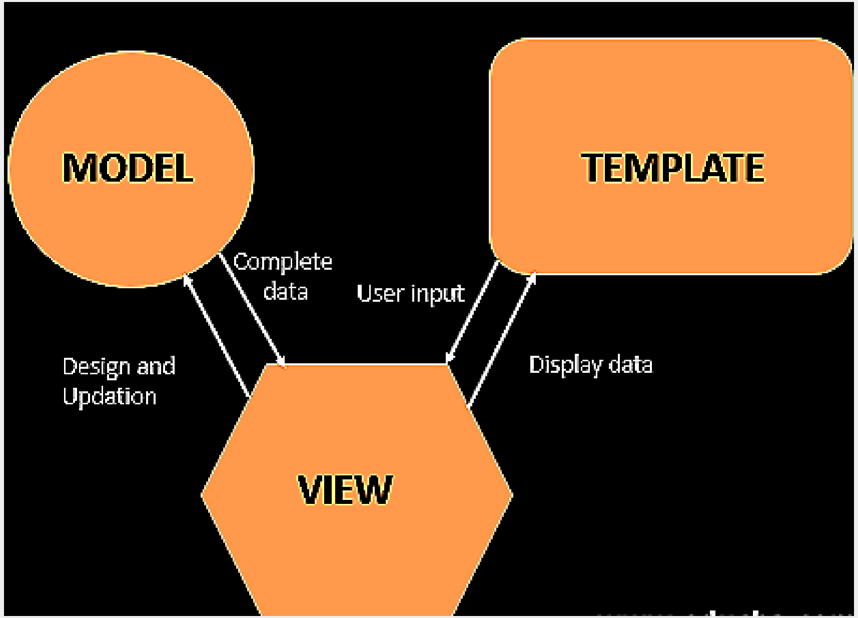


Image Source: https://media.geeksforgeeks.org/wp-content/uploads/20210606092225/image.png

Model-View-Template (MVT) Architecture (Continued)

- MVT Structure has the following three parts –
- Model: The model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).
- View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.
- Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

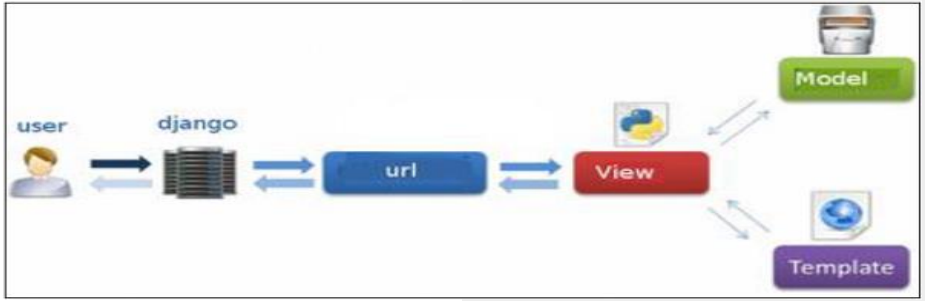


Image Source: https://www.tutorialspoint.com/django/images/django_mvc_mvt_pattern.jpg

Model-View-Controller (MVC) Architecture

- Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications.
- A Model View Controller pattern is made up of the following three parts −
- Model − The lowest level of the pattern which is responsible for maintaining data.
- View − This is responsible for displaying all or a portion of the data to the user.
- Controller − Software Code that controls the interactions between the Model and View.

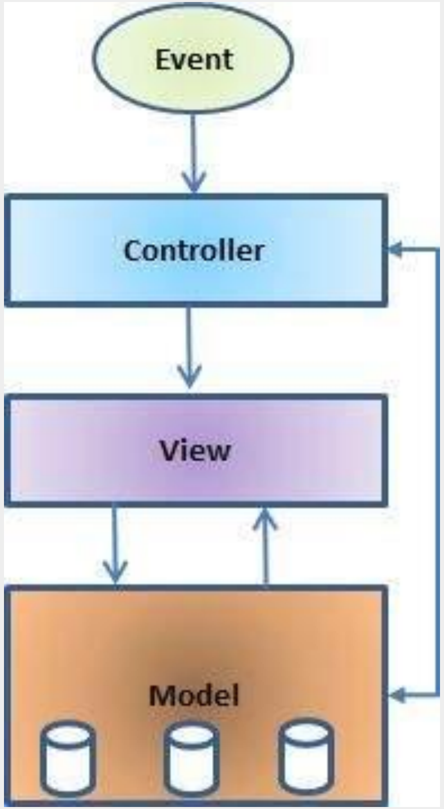


Image Source: https://www.tutorialspoint.com/struts_2/images/struts-mvc.jpg

Model-View-Controller (MVC) Architecture (Continued)

MVC is popular as it isolates the application logic from the user interface layer and supports separation of concerns. Here the Controller receives all requests for the application and then works with the Model to prepare any data needed by the View. The View then uses the data prepared by the Controller to generate a final presentable response. The MVC abstraction can be graphically represented as shown in Fig1.

The Model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The View

It means presentation of data in a particular format, triggered by a controller's decision to present the data. They are script-based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The Controller

The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

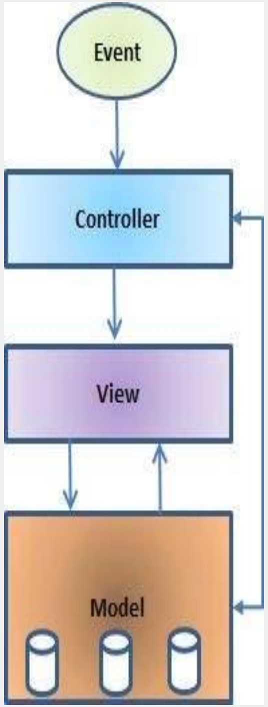


Fig1.

Image Source: https://www.tutorialspoint.com/struts_2/images/struts-mvc.jpg

In this section, Let us work practically. Lets get your hands dirty with code

- Installation of Django
- Creating the first project with Django

Installation of Django

- Creating environment for Django project
  - Install latest version of python
  - Check installed version
  - Install pipenev
  - Install visual studio code editor
- Install Django
  - Create virtual environment
  - Install django

Installation of Django

Step 1) Creating environment for Django project (Continued)

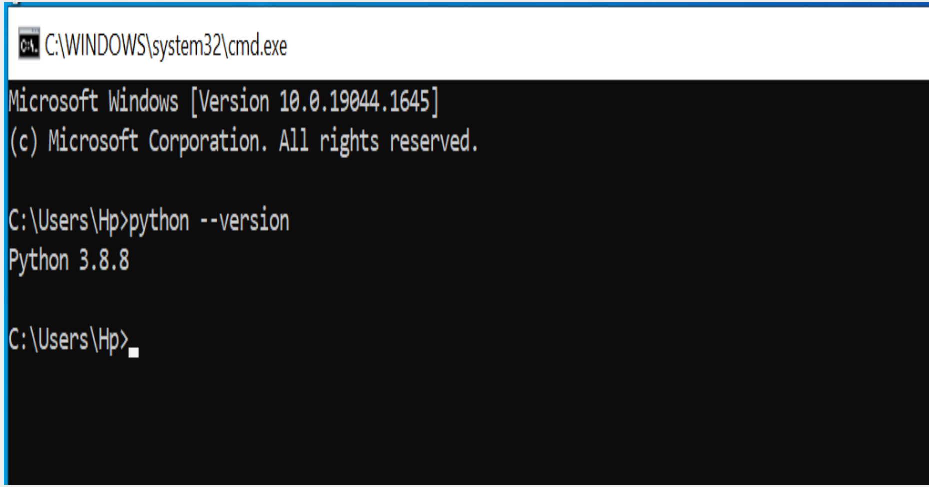
A) Install latest version of python

1. Download and install latest version of python from the url **https://www.python.org/downloads/**

B) Check for installed version of python

1. Press Window + R to open command prompt
2. Type cmd in open box and press ok button
3. Command prompt will open
4. On command prompt type following command, it will display the current python version installed on your laptop

> **python --version**

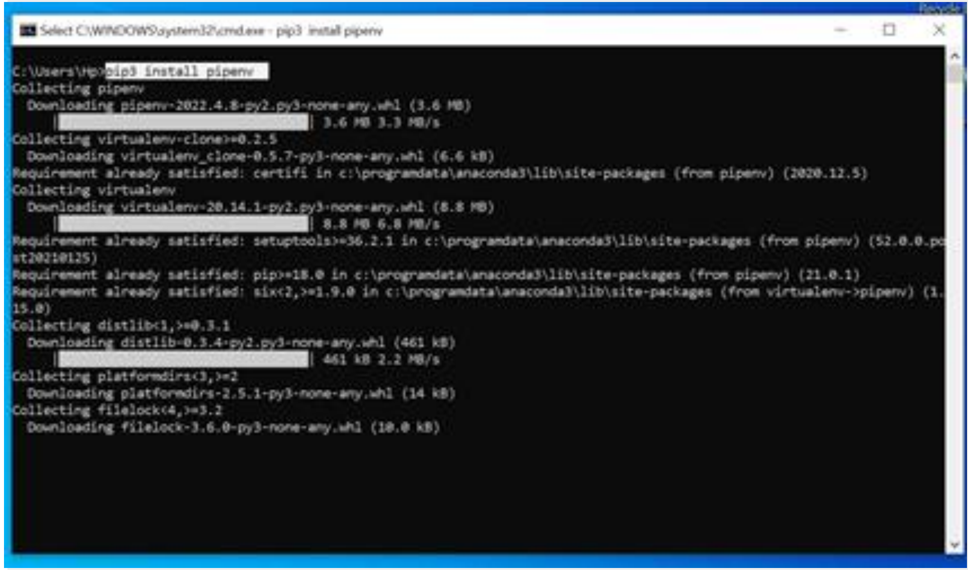


```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Hp>python --version
Python 3.8.8

C:\Users\Hp>_
```

Step 1) Creating environment for Django project (Continued)

C) Install **pipenev**

> pip3 install pipenv



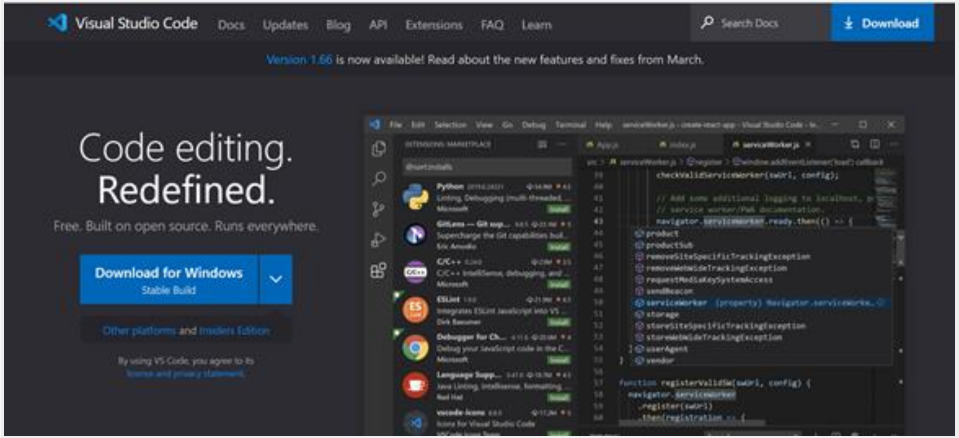
Select C:\WINDOWS\system32\cmd.exe - pip3 install pipenv

```
C:\Users\Hp>pip3 install pipenv
Collecting pipenv
  Downloading pipenv-2022.4.8-py2.py3-none-any.whl (3.6 MB)
     |████████████████████████████████| 3.6 MB 3.3 MB/s
Collecting virtualenv-clone>=0.2.5
  Downloading virtualenv_clone-0.5.7-py3-none-any.whl (6.6 kB)
Requirement already satisfied: certifi in c:\programdata\anaconda3\lib\site-packages (from pipenv) (2020.12.5)
Collecting virtualenv
  Downloading virtualenv-20.14.1-py2.py3-none-any.whl (8.8 MB)
     |████████████████████████████████| 8.8 MB 6.8 MB/s
Requirement already satisfied: setuptools>=36.2.1 in c:\programdata\anaconda3\lib\site-packages (from pipenv) (52.0.0.po
st20210125)
Requirement already satisfied: pip>=18.0 in c:\programdata\anaconda3\lib\site-packages (from pipenv) (21.0.1)
Requirement already satisfied: six<2,>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from virtualenv->pipenv) (1.
15.0)
Collecting distlib<1,>=0.3.1
  Downloading distlib-0.3.4-py2.py3-none-any.whl (461 kB)
     |████████████████████████████████| 461 kB 2.2 MB/s
Collecting platformdirs<3,>=2
  Downloading platformdirs-2.5.1-py3-none-any.whl (14 kB)
Collecting filelock<4,>=3.2
  Downloading filelock-3.6.0-py3-none-any.whl (10.0 kB)
```

Step 1) Creating environment for Django project (Continued)

D) Install **visual studio code editor**

- **Visit URI and download https://code.visualstudio.com/**



Version 1.66 is now available! Read about the new features and fixes from March.

Code editing.
Redefined.

Free. Built on open source. Runs everywhere.

By using VS Code, you agree to its license and privacy statement.

Step 2) Creating the first Project with django

A) Switch to Desktop

> cd Desktop

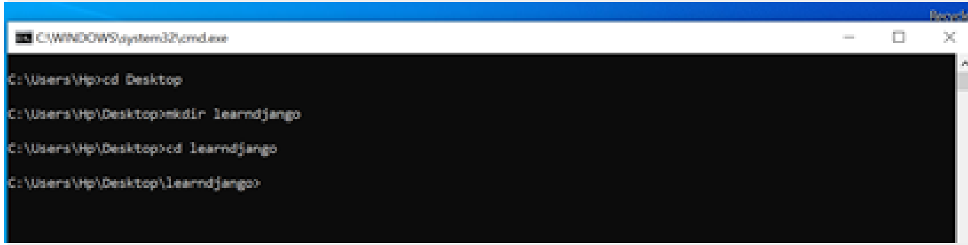
B) Create Project folder

> mkdir <<projectname_folder>>

> mkdir learndjango

Move to learndjango   directory

>cd learndjango



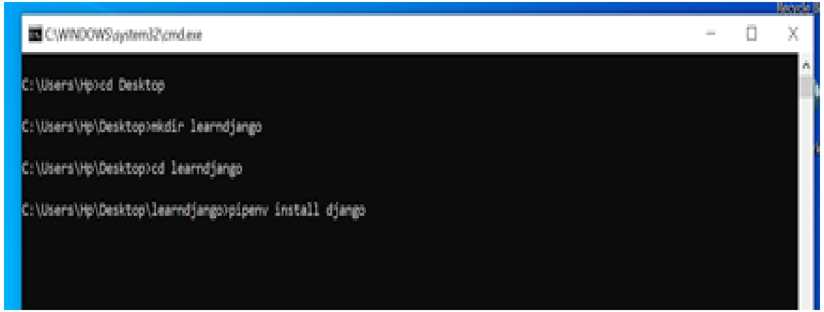
```
C:\Users\Hp>cd Desktop

C:\Users\Hp\Desktop>mkdir learndjango

C:\Users\Hp\Desktop>cd learndjango

C:\Users\Hp\Desktop\learndjango>
```

Creating the first Project with django

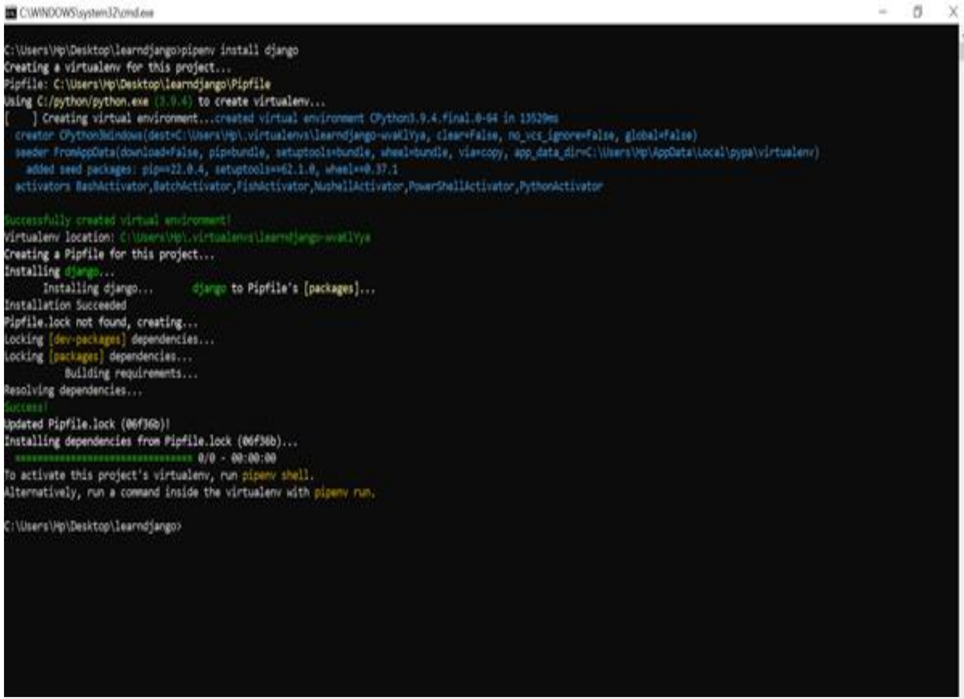
C) Install django

>pipenv install django

Successfully created the virtual environment



```
C:\Users\Hp>cd Desktop

C:\Users\Hp\Desktop>mkdir learndjango

C:\Users\Hp\Desktop>cd learndjango

C:\Users\Hp\Desktop\learndjango>pipenv install django
```

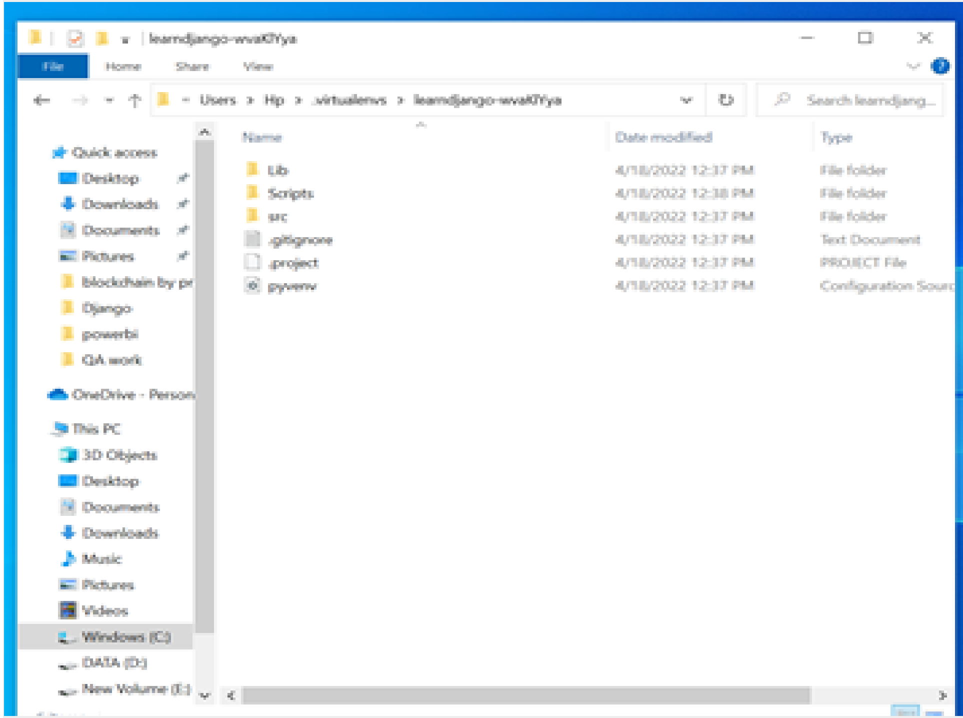


```
C:\Users\Hp\Desktop\learndjango>pipenv install django
Creating a virtualenv for this project...
Pipfile: C:\Users\Hp\Desktop\learndjango\Pipfile
Using C:/python/python.exe (3.9.4) to create virtualenv...
[    ] Creating virtual environment...created virtual environment CPython3.9.4.final.0-64 in 13529ms
  creator CPython3Windows(dest=C:\Users\Hp\.virtualenvs\learndjango-wvaKlYya, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\Hp\AppData\Local\pypa\virtualenv)
    added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator

Successfully created virtual environment!
Virtualenv location: C:\Users\Hp\.virtualenvs\learndjango-wvaKlYya
Creating a Pipfile for this project...
Installing django...
       Installing django...       django to Pipfile's [packages]...
Installation Succeeded
Pipfile.lock not found, creating...
Locking [dev-packages] dependencies...
Locking [packages] dependencies...
           Building requirements...
Resolving dependencies...
Success!
Updated Pipfile.lock (06f36b)!
Installing dependencies from Pipfile.lock (06f36b)...
  ================================ 0/0 - 00:00:00
To activate this project's virtualenv, run pipenv shell.
Alternatively, run a command inside the virtualenv with pipenv run.

C:\Users\Hp\Desktop\learndjango>
```

Creating the first Project with django

C) Install django (Continued)

- Virtual environment location is shown in above image C:\Users\Hp\.virtualenvs\learndjango-wvaKlYya



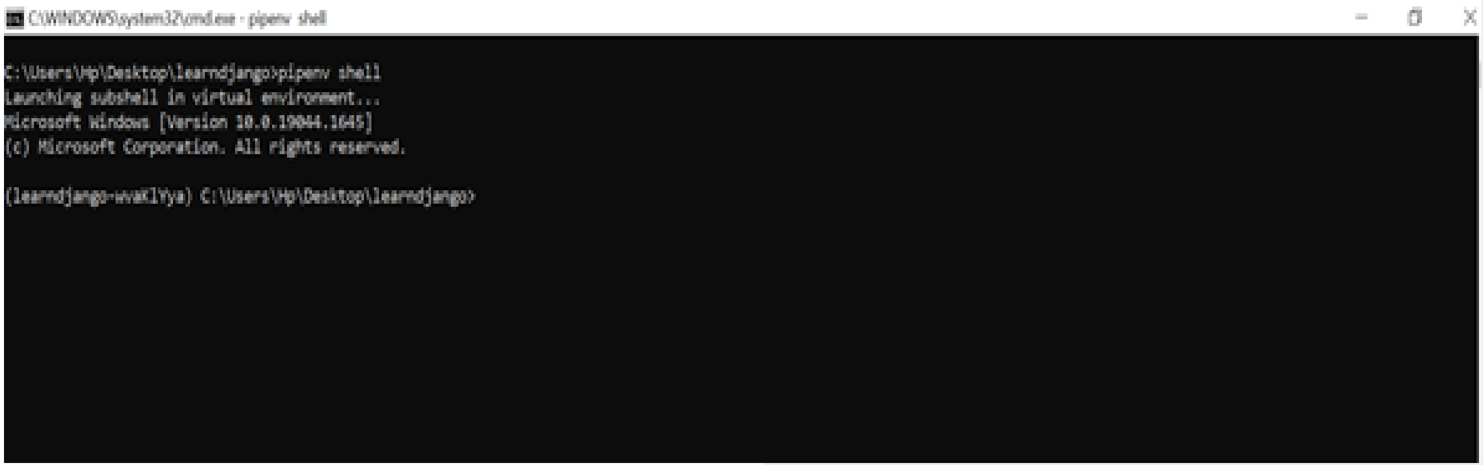
Name	Date modified	Type
Lib	4/18/2022 12:37 PM	File folder
Scripts	4/18/2022 12:38 PM	File folder
src	4/18/2022 12:37 PM	File folder
.gitignore	4/18/2022 12:37 PM	Text Document
.project	4/18/2022 12:37 PM	PROJECT File
pyvenv	4/18/2022 12:37 PM	Configuration Sourc

Creating the first Project with django

C)Install django (Continued)

- Activate python interpreter under this virtual environment

>pipenv shell



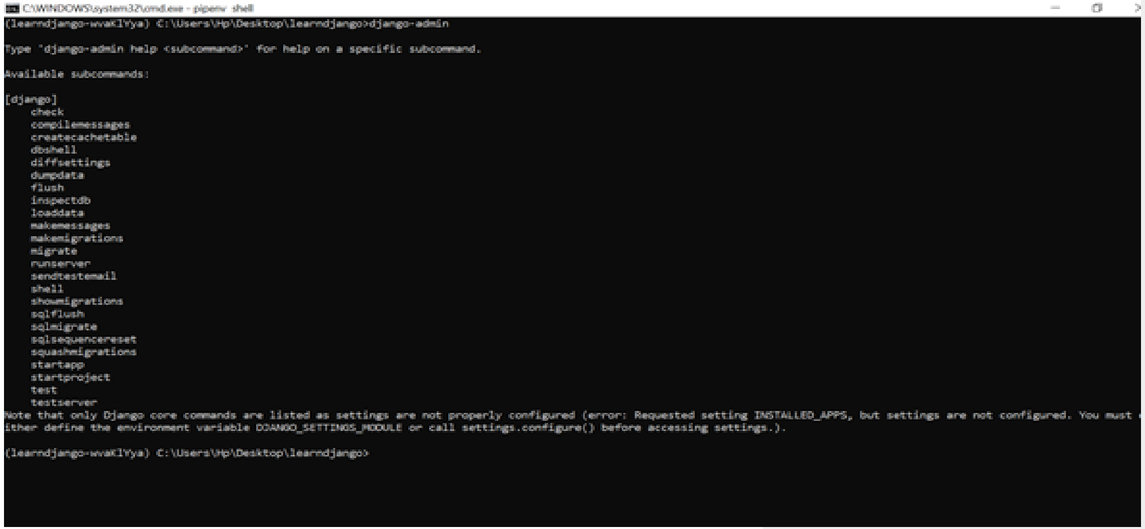
```
C:\Users\Hp\Desktop\learndjango>pipenv shell
Launching subshell in virtual environment...
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>
```

Creating the first Project with django

C) Install django (Continued)

- Run django-admin to start new project.
- django admin is utility comes along with django

>django-admin

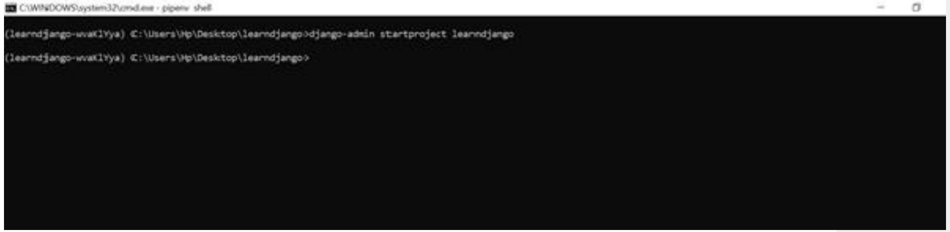


```
(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>django-admin

Type 'django-admin help <subcommand>' for help on a specific subcommand.

Available subcommands:

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
    makemigrations
    migrate
    runserver
    sendtestemail
    shell
    showmigrations
    sqlflush
    sqlmigrate
    sqlsequencereset
    squashmigrations
    startapp
    startproject
    test
    testserver
Note that only Django core commands are listed as settings are not properly configured (error: Requested setting INSTALLED_APPS, but settings are not configured. You must 
ither define the environment variable DJANGO_SETTINGS_MODULE or call settings.configure() before accessing settings.).

(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>
```

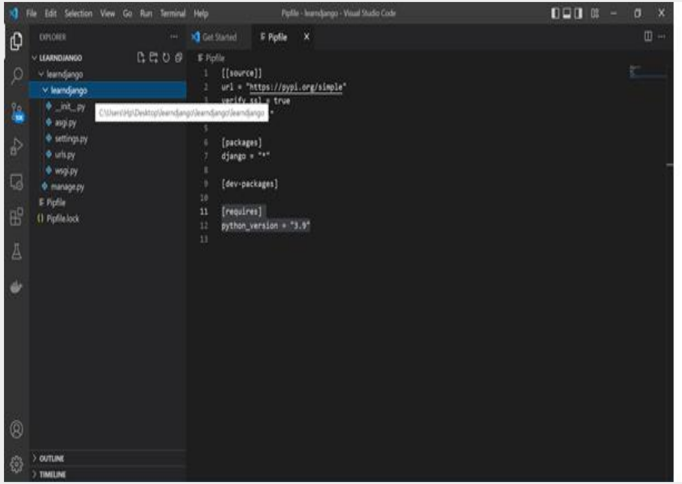
Creating the first Project with django

D) start our project learndjango

> django-admin startproject <<project_name>>
> django-admin startproject learndjango



```
(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>django-admin startproject learndjango

(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>
```



C:\Users\Hp\Desktop\learndjango\learndjango\learndjango

```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true


[packages]
django = "*"

[dev-packages]

[requires]
python_version = "3.9"
```

-It creates the two directories with the same name learndjango

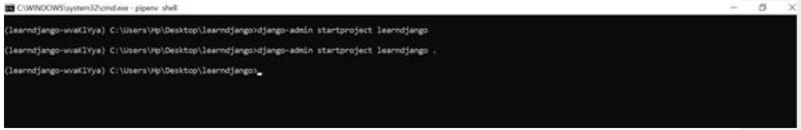
-First directory learndjango is the project directory and second directory learndjango is the django application directory

-delete the first directory learndjango and go to the command prompt

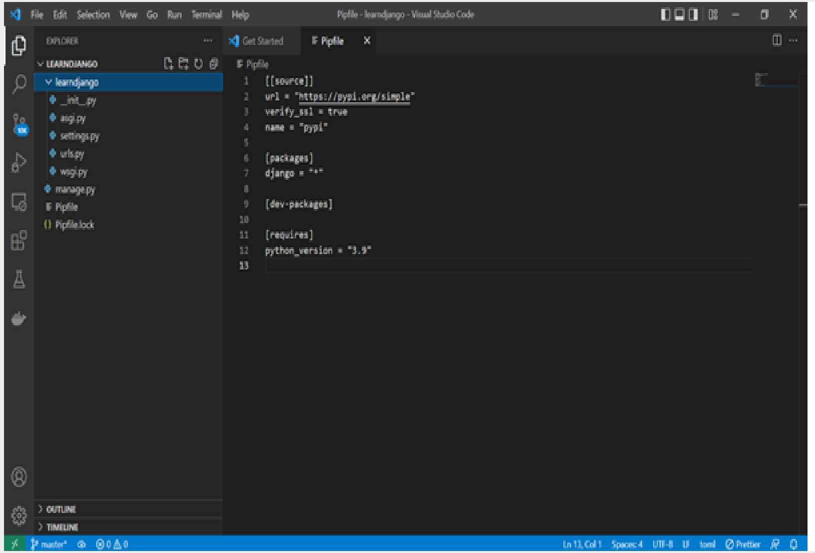
Creating the first Project with django

E) Go Back to the terminal and execute the command

> django-admin startproject learndjango .



```
(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>django-admin startproject learndjango

(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>django-admin startproject learndjango .

(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>_
```



```
[[source]]
url = "https://pypi.org/simple"
verify_ssl = true
name = "pypi"

[packages]
django = "*"

[dev-packages]

[requires]
python_version = "3.9"

```

It will use current directory as our project directory .It will now not going to create the additional directory.

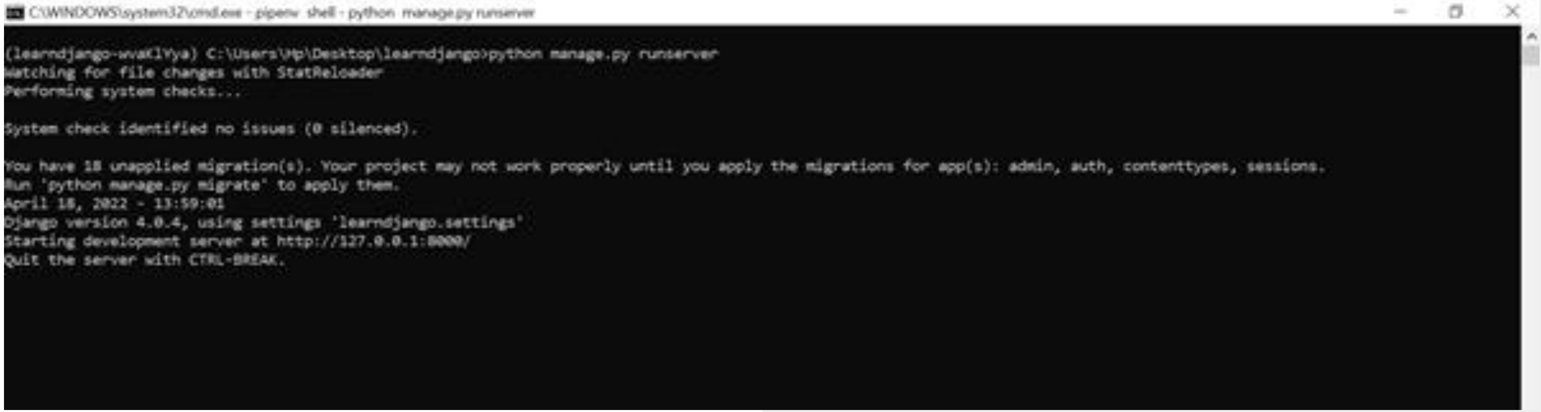
Creating the first Project with django

F) Start webserver

- Start webserver - Run the command

>python manage.py runserver

It will start server at http://127.0.0.1:8000/



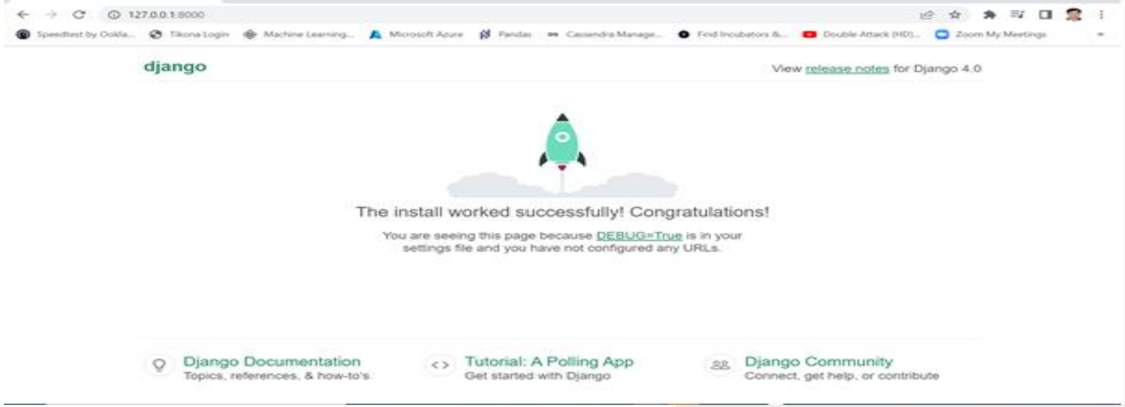
C:\WINDOWS\system32\cmd.exe - pipenv shell - python manage.py runserver

```
(learndjango-wvaKlYya) C:\Users\Hp\Desktop\learndjango>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
April 18, 2022 - 13:59:01
Django version 4.0.4, using settings 'learndjango.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Creating the first Project with django

G) Load the landing page/welcome page of django

- Now open browser and type address http://127.0.0.1:8000/ to open home page of our Django project learndjango



django

View release notes for Django 4.0

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Congratulation! You have installed and run the Django home page successfully.

Writing your first Django app: basic poll application

Throughout this tutorial section, we’ll walk you through the creation of a basic poll application. It’ll consist of two parts:

A public site that lets people view polls and vote in them.
An admin site that lets you add, change, and delete polls.

We’ll assume you have Django installed already. You can tell Django is installed and which version by running the following command in a command prompt

```
> python -m django --version
```

If Django is installed, you should see the version of your installation. If it isn’t, you’ll get an error telling “No module named django”.

This tutorial is written for Django 4.0, which supports Python 3.8 and later.

Writing your first Django app: basic poll application

Creating your project

If this is your first time using Django, you’ll have to take care of some initial setup. Namely, you’ll need to auto-generate some code that establishes a Django project – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.

From the command line, cd into a directory where you’d like to store your code, then run the following command:

```
> django-admin startproject mysite
```

This will create a mysite directory in your current directory.

Writing your first Django app: basic poll application

Creating your project (Continued)

Let’s look at what startproject created:

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

We will dicuss on these files in next slide.

Writing your first Django app: basic poll application

Creating your project (Continued)

Let us discuss on different files created by Django . These files are as follows:

The outer mysite/ root directory is a container for your project. Its name doesn’t matter to Django; you can rename it to anything you like.

manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin and manage.py.

The inner mysite/ directory is the actual Python package for your project. Its name is the Python package name you’ll need to use to import anything inside it (e.g. mysite.urls).

mysite/__init__.py: An empty file that tells Python that this directory should be considered a Python package.

Writing your first Django app: basic poll application

Creating your project (Continued)

Let us discuss on different files created by Django (Continued)

mysite/urls.py: The URL declarations for this Django project; a “table of contents” of your Django-powered site. You can read more about URLs in URL dispatcher.

mysite/asgi.py: An entry-point for ASGI-compatible web servers to serve your project. See How to deploy with ASGI for more details.

mysite/wsgi.py: An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

Writing your first Django app: basic poll application

The development server : Run the server

- Let’s verify your Django project works. Change into the outer mysite directory, if you haven’t already, and run the following commands:

```
>python manage.py runserver
```

- You’ll see the following output on the command line:

```
Performing system checks...
System check identified no issues (0 silenced).
You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.
April 20, 2022 - 15:50:53
Django version 4.0, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

- Ignore the warning about unapplied database migrations for now; we’ll deal with the database shortly.

Writing your first Django app: basic poll application

The development server : Changing the port :

By default, the runserver command starts the development server on the internal IP at port 8000.

If you want to change the server’s port, pass it as a command-line argument. For instance, this command starts the server on port 8080:

```
>python manage.py runserver 8080
```

If you want to change the server’s IP, pass it along with the port. For example, to listen on all available public IPs (which is useful if you are running Vagrant or want to show off your work on other computers on the network), use:

```
>python manage.py runserver 0:8000
```

0 is a shortcut for 0.0.0.0.

Automatic reloading of runserver

Writing your first Django app: basic poll application

The development server : Automatic reloading of runserver

The development server automatically reloads Python code for each request as needed. You don’t need to restart the server for code changes to take effect. However, some actions like adding files don’t trigger a restart, so you’ll have to restart the server in these cases.

Writing your first Django app: basic poll application

Creating the poll app

Projects vs. apps

What’s the difference between a project and an app? An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

Writing your first Django app: basic poll application

Creating the poll app

Now that your environment – a “project” – is set up, you’re set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

Your apps can live anywhere on your Python path. In this tutorial, we’ll create our poll app in the same directory as your manage.py file so that it can be imported as its own top-level module, rather than a submodule of mysite.

To create your app, make sure you’re in the same directory as manage.py and type this command:

```
>python manage.py startapp polls
```

Writing your first Django app: basic poll application

Creating the poll app

- Now that your environment – a “project” – is set up, you’re set to start doing work.
- That’ll create a directory polls, which is laid out like this:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

This directory structure will house the poll application.

Writing your first Django app: basic poll application

Creating the poll app: Write your first view

- Now that your environment – a “project” – is set up, you’re set to start doing work.
- Let’s write the first view. Open the file polls/views.py and put the following Python code in it:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

This is the simplest view possible in Django. To call the view, we need to map it to a URL - and for this we need a URLconf.

Writing your first Django app: basic poll application

Creating the poll app: Write your first view

- To create a URLconf in the polls directory, create a file called urls.py. Your app directory should now look like:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

Writing your first Django app: basic poll application

Creating the poll app: Write your first view

- In the polls/urls.py file include the following code:

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Writing your first Django app: basic poll application

Creating the poll app: Write your first view

- The next step is to point the root URLconf at the polls.urls module. In **mysite/urls.py**, add an import for django.urls.include and insert an include() in the urlpatterns list, so you have:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]
```

- The include() function allows referencing other URLconfs. Whenever Django encounters include(), it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLconf for further processing.

Writing your first Django app: basic poll application

Creating the poll app: Write your first view

The idea behind include() is to make it easy to plug-and-play URLs. Since polls are in their own URLconf (polls/urls.py), they can be placed under “/polls/”, or under “/fun_polls/”, or under “/content/polls/”, or any other path root, and the app will still work

Note: When to use include()

You should always use include() when you include other URL patterns. admin.site.urls is the only exception to this.

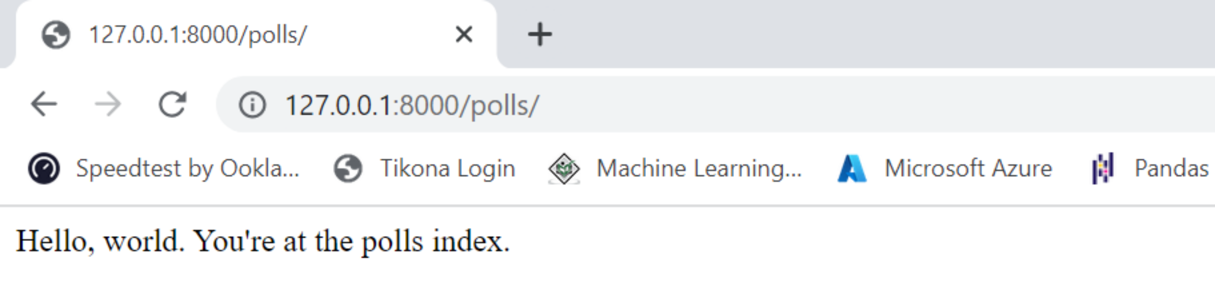
Writing your first Django app: basic poll application

Creating the poll app: Write your first view

You have now wired an index view into the URLconf. Verify it’s working with the following command:

```
>python manage.py runserver
```

Go to http://localhost:8000/polls/ in your browser, and you should see the text “Hello, world. You’re at the polls index.”, which you defined in the index view.



Hello, world. You're at the polls index.

Let us add database into the poll app

Database

Disclaimer: The content is curated for educational purposes only.

Writing your first Django app: Part 2

Migrating the database

We'll set up the database, create your first model, and get a quick introduction to Django's automatically-generated admin site.

Database setup

Now, open up mysite/settings.py. It's a normal Python module with module-level variables representing Django settings.

By default, the configuration uses SQLite. SQLite is included in Python, so you won't need to install anything else to support your database.

Writing your first Django app: Part 2

Migrating the database: Database setup (Continued)

If you wish to use another database, install the appropriate database bindings and change the following keys in the DATABASES 'default' item to match your database connection settings:

ENGINE – Either 'django.db.backends.sqlite3', 'django.db.backends.postgresql', 'django.db.backends.mysql', or 'django.db.backends.oracle'. Other backends are also available.

NAME – The name of your database. If you’re using SQLite, the database will be a file on your computer; in that case, NAME should be the full absolute path, including filename, of that file. The default value, BASE_DIR / 'db.sqlite3', will store the file in your project directory.

If you are not using SQLite as your database, additional settings such as USER, PASSWORD, and HOST must be added.

Writing your first Django app: Part 2

Setup Timezone , INSTALLED_APPS

While you’re editing mysite/settings.py, set TIME_ZONE to your time zone.

Also, note the INSTALLED_APPS setting at the top of the file. That holds the names of all Django applications that are activated in this Django instance. Apps can be used in multiple projects, and you can package and distribute them for use by others in their projects.

By default, INSTALLED_APPS contains the following apps, all of which come with Django:

django.contrib.admin – The admin site. You’ll use it shortly.

django.contrib.auth – An authentication system.

django.contrib.contenttypes – A framework for content types.

django.contrib.sessions – A session framework.

django.contrib.messages – A messaging framework.

django.contrib.staticfiles – A framework for managing static files.

These applications are included by default as a convenience for the common case.

Writing your first Django app: Part 2

Setup Timezone , INSTALLED_APPS (Continued)

Some of these applications mentioned in previous slide make use of at least one database table, though, so we need to create the tables in the database before we can use them. To do that, run the following command:

```
>python manage.py migrate
```

The migrate command looks at the INSTALLED_APPS setting and creates any necessary database tables according to the database settings in your mysite/settings.py file and the database migrations shipped with the app

You'll see a message for each migration it applies.

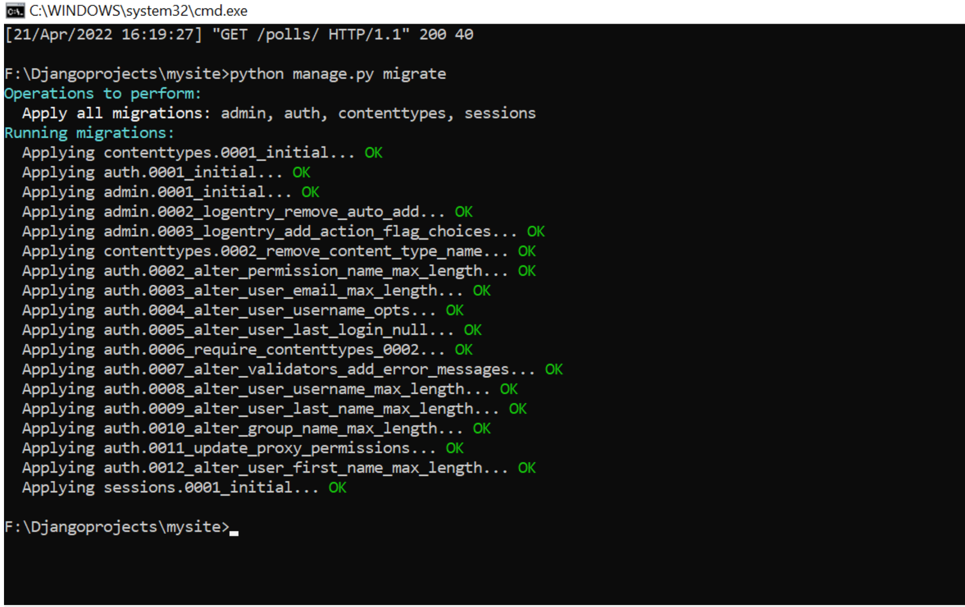
If you're interested, run the command-line client for your database and type \dt (PostgreSQL), SHOW TABLES; (MariaDB, MySQL), .tables (SQLite), or SELECT TABLE_NAME FROM USER_TABLES; (Oracle) to display the tables Django created.

Writing your first Django app: Part 2

INSTALLED_APPS (Continued): Applying Migration

After Output after executing the command

> python manage.py migrate


C:\WINDOWS\system32\cmd.exe

```
[21/Apr/2022 16:19:27] "GET /polls/ HTTP/1.1" 200 40

F:\Djangoprojects\mysite>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK

F:\Djangoprojects\mysite>
```

Writing your first Django app: Part 2

Creating models

Now we’ll define your models – essentially, your database layout, with additional metadata.

Overview-

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you’re storing. Django follows the DRY Principle. The goal is to define your data model in one place and automatically derive things from it.

This includes the migrations - unlike in Ruby On Rails, for example, migrations are entirely derived from your models file, and are essentially a history that Django can roll through to update your database schema to match your current models.

Writing your first Django app: Part 2

Creating models (Continued)

In our poll app, we’ll create two models: Question and Choice.

A Question has a question and a publication date.

A Choice has two fields: the text of the choice and a vote tally.

Each Choice is associated with a Question.

These concepts are represented by Python classes. Edit the polls/models.py file so it looks like this:

Writing your first Django app: Part 2

Creating models (Continued)

- These concepts are represented by Python classes. Edit the **polls/models.py** file so it looks like this:

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Writing your first Django app: Part 2

Models explanation :

Here, each model is represented by a class that subclasses django.db.models.Model. Each model has a number of class variables, each of which represents a database field in the model.

Each field is represented by an instance of a Field class – e.g., CharField for character fields and DateTimeField for datetimes. This tells Django what type of data each field holds.

The name of each Field instance (e.g. question_text or pub_date) is the field's name, in machine-friendly format. You'll use this value in your Python code, and your database will use it as the column name.

You can use an optional first positional argument to a Field to designate a human-readable name. That's used in a couple of introspective parts of Django, and it doubles as documentation. If this field isn't provided, Django will use the machine-readable name. In this example, we've only defined a human-readable name for Question.pub_date. For all other fields in this model, the field's machine-readable name will suffice as its human-readable name.

Writing your first Django app: Part 2

Models explanation : (Continued)

Some Field classes have required arguments. Char Field, for example, requires that you give it a max_length. That’s used not only in the database schema, but in validation, as we’ll soon see.

A Field can also have various optional arguments; in this case, we’ve set the default value of votes to 0.

Finally, note a relationship is defined, using Foreign Key. That tells Django each Choice is related to a single Question. Django supports all the common database relationships: many-to-one, many-to-many, and one-to-one.

Writing your first Django app: Part 2

Activating models-

That small bit of model code gives Django a lot of information. With it, Django is able to:

Create a database schema (CREATE TABLE statements) for this app.

Create a Python database-access API for accessing Question and Choice objects.

Writing your first Django app: Part 2

Activating models-

First we need to tell our project that the polls app is installed

To include the app in our project, we need to add a reference to its configuration class in the INSTALLED_APPS setting. The PollsConfig class is in the polls/apps.py file, so its dotted path is 'polls.apps.PollsConfig'. Edit the mysite/settings.py file and add that dotted path to the INSTALLED_APPS setting. It’ll look like this:

```
INSTALLED_APPS = [
    'polls.apps.PollsConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Writing your first Django app: Part 2

Activating models-

- Now Django knows to include the polls app. Let’s run another command:

```
>python manage.py makemigrations polls
```

- You should see something similar to the following:

```
Migrations for 'polls':
  polls/migrations/0001_initial.py
    - Create model Question
    - Create model Choice
```

- By running makemigrations, you’re telling Django that you’ve made some changes to your models (in this case, you’ve made new ones) and that you’d like the changes to be stored as a migration.

Writing your first Django app: Part 2

Activating models-

Migrations are how Django stores changes to your models (and thus your database schema) - they’re files on disk. You can read the migration for your new model if you like; it’s the file polls/migrations/0001_initial.py. Don’t worry, you’re not expected to read them every time Django makes one, but they’re designed to be human-editable in case you want to manually tweak how Django changes things.

There’s a command that will run the migrations for you and manage your database schema automatically - that’s called migrate, and we’ll come to it in a moment - but first, let’s see what SQL that migration would run. The sqlmigrate command takes migration names and returns their SQL:

```
>python manage.py sqlmigrate polls 0001
```

Writing your first Django app: Part 2

Activating models-

You should see something similar to the following (we’ve reformatted it for readability):

```
BEGIN;
-- Create model Question
-CREATE TABLE "polls_question" (
    "id" serial NOT NULL PRIMARY KEY,
    "question_text" varchar(200) NOT NULL,
    "pub_date" timestamp with time zone NOT NULL
);
-- Create model Choice
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL,
    "question_id" integer NOT NULL
);
```

Writing your first Django app: Part 2

Activating models- (Continued)

```
ALTER TABLE "polls_choice"
  ADD CONSTRAINT "polls_choice_question_id_c5b4b260_fk_polls_question_id"
    FOREIGN KEY ("question_id")
    REFERENCES "polls_question" ("id")
    DEFERRABLE INITIALLY DEFERRED;
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");

COMMIT;
```

Writing your first Django app: Part 2

Activating models- (Continued)

Note the following:

The exact output will vary depending on the database you are using. The example above is generated for PostgreSQL.

Table names are automatically generated by combining the name of the app (polls) and the lowercase name of the model – question and choice. (You can override this behavior.)

Primary keys (IDs) are added automatically. (You can override this, too.)

By convention, Django appends "_id" to the foreign key field name. (Yes, you can override this, as well.)

The foreign key relationship is made explicit by a FOREIGN KEY constraint. Don’t worry about the DEFERRABLE parts; it’s telling PostgreSQL to not enforce the foreign key until the end of the transaction.

It’s tailored to the database you’re using, so database-specific field types such as auto_increment (MySQL), serial (PostgreSQL), or integer primary key autoincrement (SQLite) are handled for you automatically. Same goes for the quoting of field names – e.g., using double quotes or single quotes.

The sqlmigrate command doesn’t actually run the migration on your database - instead, it prints it to the screen so that you can see what SQL Django thinks is required. It’s useful for checking what Django is going to do or if you have database administrators who require SQL scripts for changes.

Writing your first Django app: Part 2

Migrating the models

- Now, run migrate again to create those model tables in your database:

  > python manage.py migrate

- It will show output like

```
perations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

The migrate command takes all the migrations that haven’t been applied (Django tracks which ones are applied using a special table in your database called django_migrations) and runs them against your database - essentially, synchronizing the changes you made to your models with the schema in the database.

Migrations are very powerful and let you change your models over time, as you develop your project, without the need to delete your database or tables and make new ones - it specializes in upgrading your database live, without losing data.

Writing your first Django app: Part 2

Playing with the API

Now, let’s hop into the interactive Python shell and play around with the free API Django gives you. To invoke the Python shell, use this command:

```
>python manage.py shell
```

We’re using this instead of simply typing “python”, because manage.py sets the DJANGO_SETTINGS_MODULE environment variable, which gives Django the Python import path to your mysite/settings.py file.

Once you’re in the shell, explore the database API:

```
>>> from polls.models import Choice, Question  # Import the model classes we just
wrote.
# No questions are in the system yet.
>>> Question.objects.all()
<QuerySet []>
```

Writing your first Django app: Part 2

Playing with the API(Continued)

Once you’re in the shell, explore the database API:

Create a new Question.

```
# Support for time zones is enabled in the default settings file, so
# Django expects a datetime with tzinfo for pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
# Save the object into the database. You have to call save() explicitly.
>>> q.save()
# Now it has an ID.
>>> q.id
1
```

Writing your first Django app: Part 2

Playing with the API(Continued)

Once you’re in the shell, explore the database API:

```
# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)
# Change values by changing the attributes, then calling save().
>>> q.question_text = "What's up?"
>>> q.save()
# objects.all() displays all the questions in the database.
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

To explore more click here

Writing your first Django app: Part 2

Playing with the API(Continued)

- Wait a minute. <Question: Question object (1)> isn’t a helpful representation of this object. Let’s fix that by editing the Question model (in the polls/models.py file) and adding a __str__() method to both Question and Choice:

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- It’s important to add __str__() methods to your models, not only for your own convenience when dealing with the interactive prompt, but also because objects’ representations are used throughout Django’s automatically-generated admin.
- Let’s also add a custom method to this model: polls/models.py –

```
import datetime

from django.db import models
from django.utils import timezone


class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Note the addition of import datetime and from django.utils import timezone, to reference Python’s standard datetime module and Django’s time-zone-related utilities in django.utils.timezone, respectively. If you aren’t familiar with time zone handling in Python, you can learn more in the time zone support docs.

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
>>> from polls.models import Choice, Question

# Make sure our __str__() addition worked.
>>> Question.objects.all()
<QuerySet [<Question: What's up?>]>

# Django provides a rich database lookup API that's entirely driven by
# keyword arguments.
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
# keyword arguments.
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
# Make sure our custom method worked.
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# Give the Question a couple of Choices. The create call constructs a new
# Choice object, does the INSERT statement, adds the choice to the set
# of available choices and returns the new Choice object. Django creates
# a set to hold the "other side" of a ForeignKey relation
# (e.g. a question's choice) which can be accessed via the API.
>>> q = Question.objects.get(pk=1)
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
# Display any choices from the related object set -- none so far.
>>> q.choice_set.all()
<QuerySet []>

# Create three choices.
>>> q.choice_set.create(choice_text='Not much', votes=0)
<Choice: Not much>
>>> q.choice_set.create(choice_text='The sky', votes=0)
<Choice: The sky>
>>> c = q.choice_set.create(choice_text='Just hacking again', votes=0)
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
# Choice objects have API access to their related Question objects.
>>> c.question
<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>
>>> q.choice_set.count()
3
```

Writing your first Django app: Part 2

Playing with the API(Continued)

- Save these changes and start a new Python interactive shell by running python manage.py shell again:

```
# The API automatically follows relationships as far as you need.
# Use double underscores to separate relationships.
# This works as many levels deep as you want; there's no limit.
# Find all Choices for any question whose pub_date is in this year
# (reusing the 'current_year' variable we created above).
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

# Let's delete one of the choices. Use delete() for that.
>>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()
```

Django Admin

Disclaimer: The content is curated for educational purposes only.

Introducing the Django Admin

Overview:

- Generating admin sites for your staff or clients to add, change, and delete content is tedious work that doesn’t require much creativity. For that reason, Django entirely automates creation of admin interfaces for models.
- The admin isn’t intended to be used by site visitors. It’s for site managers.

Creating an admin user-

- First we’ll need to create a user who can login to the admin site. Run the following command:

```
>python manage.py createsuperuser
```

Enter your desired username and press enter.

Username: admin

You will then be prompted for your desired email address:

Introducing the Django Admin

Creating an admin user-(Continued)

Email address: admin@example.com

The final step is to enter your password. You will be asked to enter your password twice, the second time as a confirmation of the first.

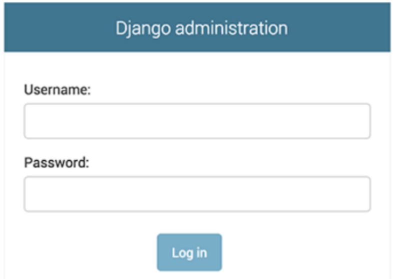
```
Password: **********
Password (again): *********
Superuser created successfully.
```

Introducing the Django Admin

- Start the development server
- The Django admin site is activated by default. Let’s start the development server and explore it.
- If the server is not running start it like so:

```
>python manage.py runserver
```

- Now, open a web browser and go to “/admin/” on your local domain – e.g., http://127.0.0.1:8000/admin/. You should see the admin’s login screen:

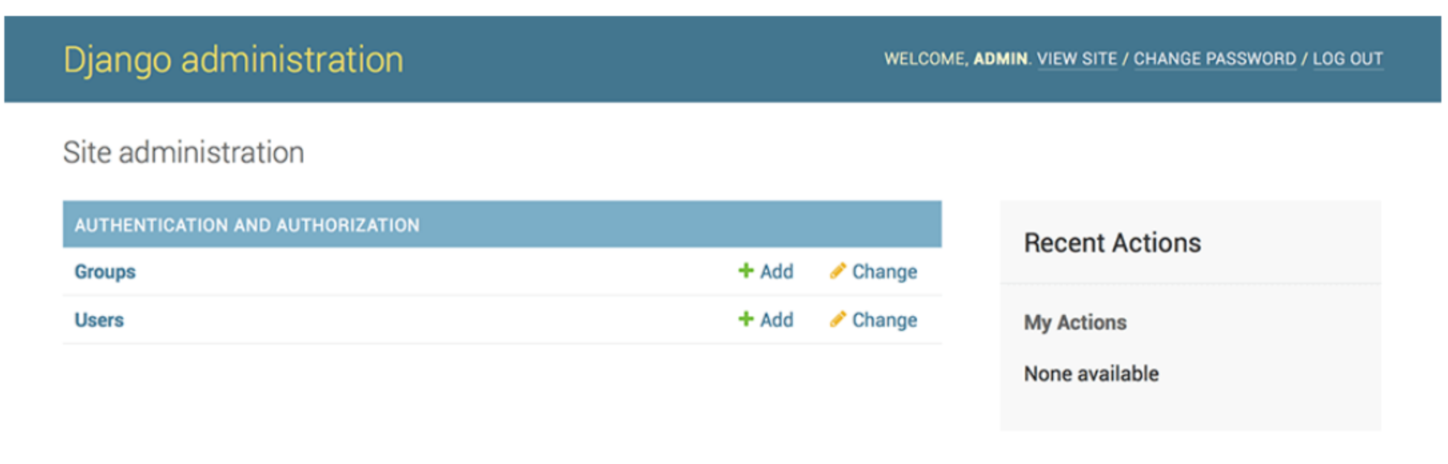


- Since translation is turned on by default, if you set LANGUAGE_CODE, the login screen will be displayed in the given language (if Django has appropriate translations).

edunet foundation

Introducing the Django Admin

- Enter the admin site:
- Now, try logging in with the superuser account you created in the previous step. You should see the Django admin index page:



Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Recent Actions

My Actions

None available

- You should see a few types of editable content: groups and users. They are provided by django.contrib.auth, the authentication framework shipped by Django.

Introducing the Django Admin

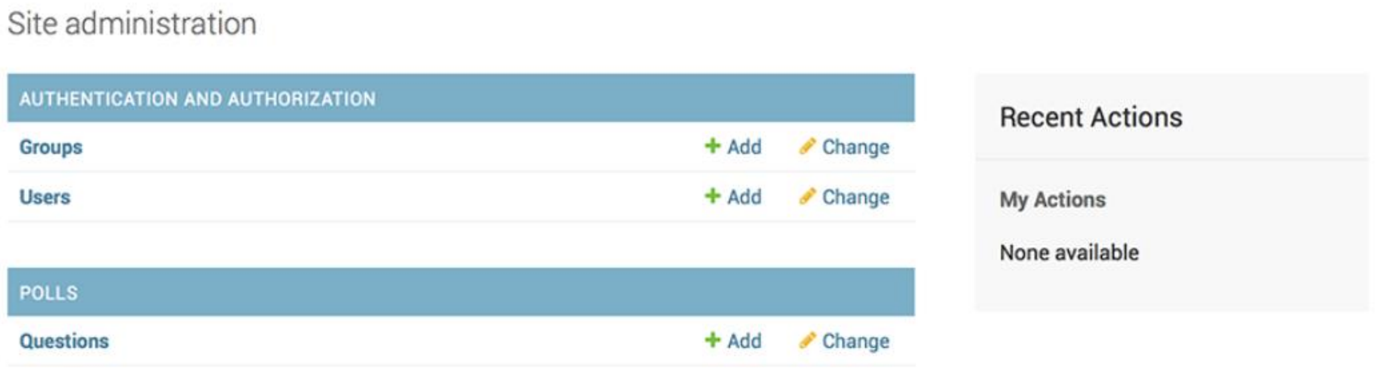
Make the poll app modifiable in the admin:

- But where’s our poll app? It’s not displayed on the admin index page.
- Only one more thing to do: we need to tell the admin that Question objects have an admin interface. To do this, open the polls/admin.py file, and edit it to look like this:

```
from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

Introducing the Django Admin

- Explore the free admin functionality
- Now that we’ve registered Question, Django knows that it should be displayed on the admin index page:



Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

POLLS		
Questions	+ Add	Change

Recent Actions

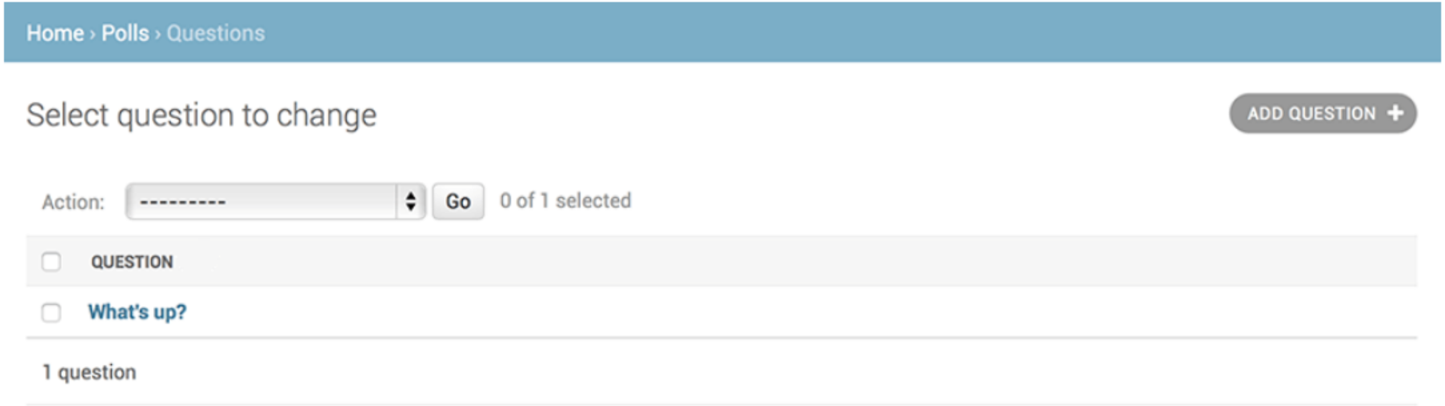
My Actions

None available

- Click “Questions”. Now you’re at the “change list” page for questions.

Introducing the Django Admin

- Explore the free admin functionality
- This page displays all the questions in the database and lets you choose one to change it. There’s the “What’s up?” question we created earlier:



Home › Polls › Questions

Select question to change

Action: --------- Go 0 of 1 selected

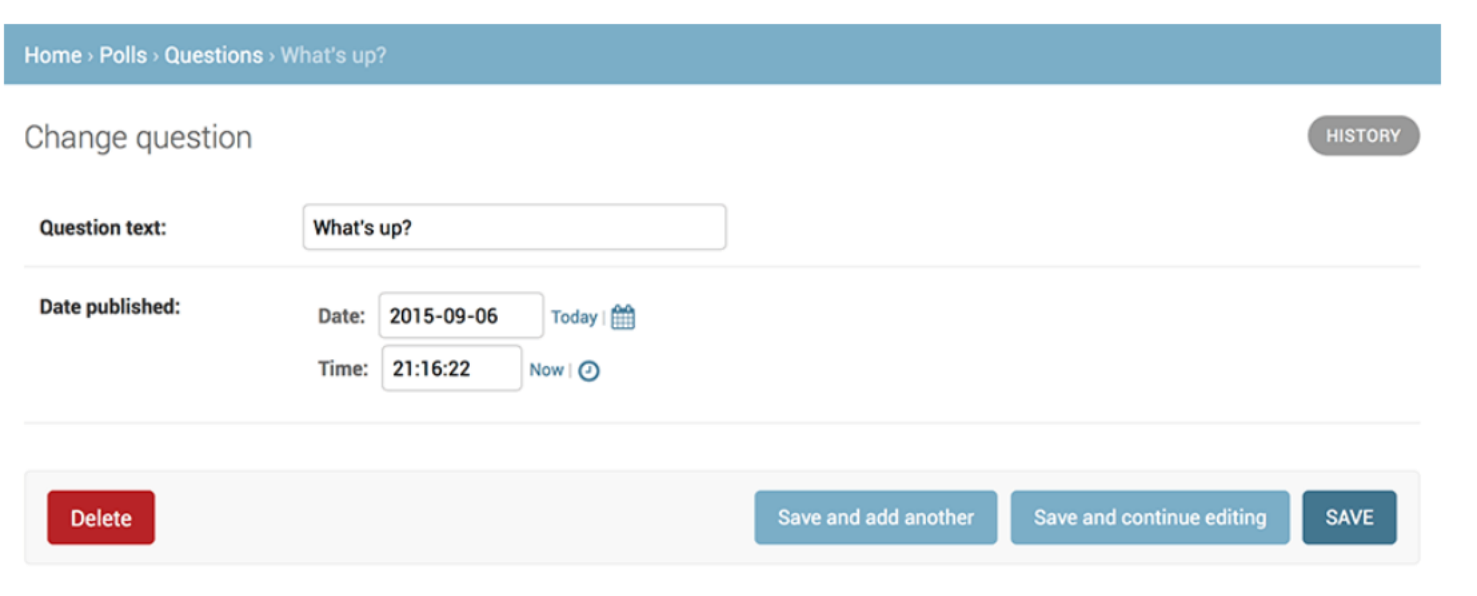
□	QUESTION
□	What’s up?

1 question

- Click the “What’s up?” question to edit it:

Introducing the Django Admin

- Explore the free admin functionality
- Edit question :



Home › Polls › Questions › What's up?

Change question

Question text: What's up?

Date published:

Date: 2015-09-06 Today |

Time: 21:16:22 Now |

Introducing the Django Admin

- Things to note here:
- The form is automatically generated from the Question model.
- The different model field types (DateTimeField, CharField) correspond to the appropriate HTML input widget. Each type of field knows how to display itself in the Django admin.
- Each DateTimeField gets free JavaScript shortcuts. Dates get a “Today” shortcut and calendar popup, and times get a “Now” shortcut and a convenient popup that lists commonly entered times.

Introducing the Django Admin

- things to note here (Continued)-
- The bottom part of the page gives you a couple of options:
- Save – Saves changes and returns to the change-list page for this type of object.
- Save and continue editing – Saves changes and reloads the admin page for this object.
- Save and add another – Saves changes and loads a new, blank form for this type of object.
- Delete – Displays a delete confirmation page.

Views and URL mapping, HttpRequest & HttpResponse,
GET & POST Method

Disclaimer: The content is curated for educational purposes only.

Views

- A view function, or view for short, is a Python function that takes a web request and returns a web response.
- This response can be the HTML contents of a web page, or a redirect, or a 404 error, or an XML document, or an image , or anything that a web browser can display.
- The convention is to put views in a file called views.py file in your project or application directory.

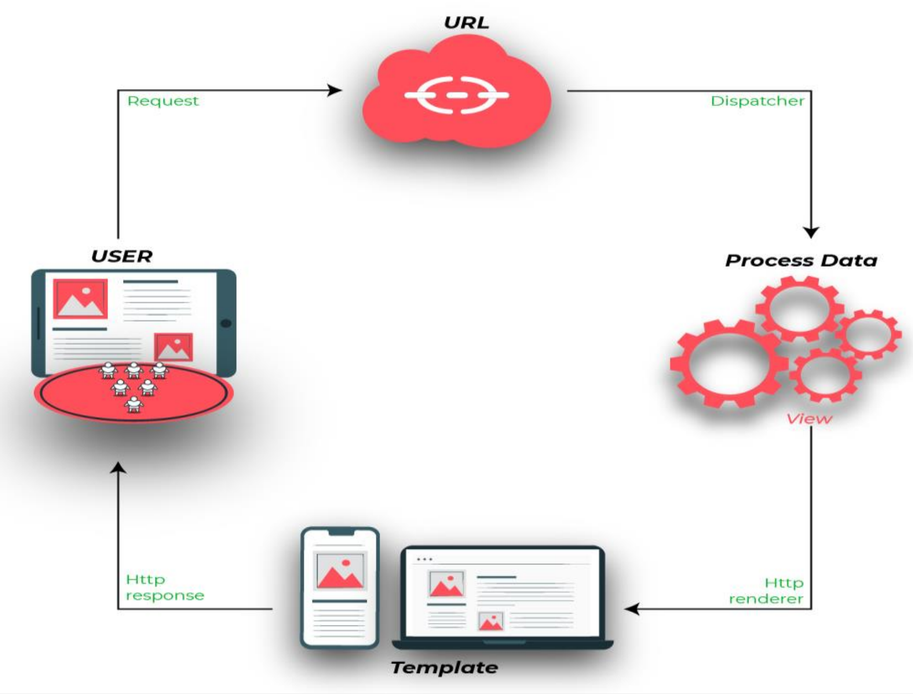


Image Source: https://media.geeksforgeeks.org/wp-content/uploads/20200124153519/django-views.jpg

Creating simple View : Example

Here’s a view that returns the current date and time, as an HTML document: learndjango/ views.py

```
# import Http Response from django
from django.http import HttpResponse
# get datetime
import datetime
 # create a function
def date_view(request):
    # fetch date and time
    now = datetime.datetime.now()
    # convert to string
    html = "Time is {}".format(now)
    # return response
    return HttpResponse(html)
```

Creating simple View: Example Explanation

- •First, we import the class HttpResponse from the django.http module, along with Python’s datetime library.
- Next, we define a function called date_view. This is the view function. Each view function takes an HttpRequest object as its first parameter, which is typically named request.
- The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

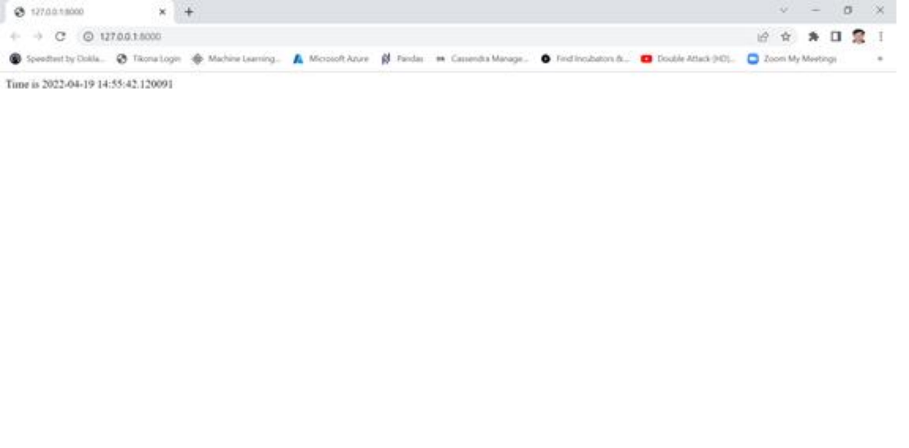
URL Mapping : Example

Let’s get this view to working, in learndjango/urls.py

```
from django.contrib import admin
from django.urls import path,include

from .views import date_view

urlpatterns = [
    #path('admin/', admin.site.urls),
    path('', date_view),
]
```

Output for DateTime example

- • Now, visit http://127.0.0.1:8000/



```
Time is 2022-04-19 14:55:42.120091
```

HttpRequest & HttpResponse,

Disclaimer: The content is curated for educational purposes only.

HttpRequest & HttpResponse

Request and response objects : Overview

- Django uses request and response objects to pass state through the system.

- When a page is requested, Django creates an HttpRequest object that contains metadata about the request. Then Django loads the appropriate view, passing the HttpRequest as the first argument to the view function. Each view is responsible for returning an HttpResponse object.

HttpRequest

HttpRequest objects

- class HttpRequest

- Attributes
- All attributes should be considered read-only, unless stated otherwise.

- HttpRequest.scheme
- A string representing the scheme of the request (http or https usually).

HttpRequest

HttpRequest objects (Continued)

HttpRequest.body
The raw HTTP request body as a bytestring. This is useful for processing data in different ways than conventional HTML forms: binary images, XML payload etc. For processing conventional form data, use HttpRequest.POST.

You can also read from an HttpRequest using a file-like interface with HttpRequest.read() or HttpRequest.readline(). Accessing the body attribute after reading the request with either of these I/O stream methods will produce a RawPostDataException.

HttpRequest.path
A string representing the full path to the requested page, not including the scheme, domain, or query string.
Example: "/music/bands/the_beatles/"

HttpRequest

HttpRequest objects (Continued)

HttpRequest.path_info

Under some web server configurations, the portion of the URL after the host name is split up into a script prefix portion and a path info portion. The path_info attribute always contains the path info portion of the path, no matter what web server is being used. Using this instead of path can make your code easier to move between test and deployment servers.

For example, if the WSGIScriptAlias for your application is set to "/minfo", then path might be "/minfo/music/bands/the_beatles/" and path_info would be "/music/bands/the_beatles/".

HttpRequest

HttpRequest objects (Continued)

HttpRequest.method

A string representing the HTTP method used in the request. This is guaranteed to be uppercase. For example:

```
if request.method == 'GET':
    do_something()
elif request.method == 'POST':
    do_something_else()
```

HttpRequest

HttpRequest objects (Continued)

HttpRequest.encoding
A string representing the current encoding used to decode form submission data (or None, which means the DEFAULT_CHARSET setting is used). You can write to this attribute to change the encoding used when accessing the form data. Any subsequent attribute accesses (such as reading from GET or POST) will use the new encoding value. Useful if you know the form data is not in the DEFAULT_CHARSET encoding.

HttpRequest.content_type
A string representing the MIME type of the request, parsed from the CONTENT_TYPE header.

HttpRequest

HttpRequest objects (Continued)

HttpRequest.POST
A dictionary-like object containing all given HTTP POST parameters, providing that the request contains form data. See the QueryDict documentation below. If you need to access raw or non-form data posted in the request, access this through the HttpRequest.body attribute instead.

It’s possible that a request can come in via POST with an empty POST dictionary – if, say, a form is requested via the POST HTTP method but does not include form data. Therefore, you shouldn’t use if request.POST to check for use of the POST method; instead, use if request.method == "POST" (see HttpRequest.method).

POST does not include file-upload information. See FILES.

HttpRequest

HttpRequest objects (Continued)

HttpRequest.content_params
A dictionary of key/value parameters included in the CONTENT_TYPE header.

HttpRequest.GET
A dictionary-like object containing all given HTTP GET parameters. See the QueryDict documentation below.

HttpRequest.COOKIES
A dictionary containing all cookies. Keys and values are strings.

HttpRequest

HttpRequest objects (Continued)

HttpRequest.FILES
A dictionary-like object containing all uploaded files. Each key in FILES is the name from the <input type="file" name="">. Each value in FILES is an UploadedFile.

See Managing files for more information.

FILES will only contain data if the request method was POST and the <form> that posted to the request had enctype="multipart/form-data". Otherwise, FILES will be a blank dictionary-like object.

HttpRequest

HttpRequest objects (Continued)

HttpRequest.META

A dictionary containing all available HTTP headers. Available headers depend on the client and server, but here are some examples:

CONTENT_LENGTH – The length of the request body (as a string).
CONTENT_TYPE – The MIME type of the request body.
HTTP_ACCEPT – Acceptable content types for the response.
HTTP_ACCEPT_ENCODING – Acceptable encodings for the response.
HTTP_ACCEPT_LANGUAGE – Acceptable languages for the response.
HTTP_HOST – The HTTP Host header sent by the client.
HTTP_REFERER – The referring page, if any.
HTTP_USER_AGENT – The client’s user-agent string.
QUERY_STRING – The query string, as a single (unparsed) string.
REMOTE_ADDR – The IP address of the client.
REMOTE_HOST – The hostname of the client.
REMOTE_USER – The user authenticated by the web server, if any.
REQUEST_METHOD – A string such as "GET" or "POST".
SERVER_NAME – The hostname of the server.
SERVER_PORT – The port of the server (as a string).

HttpRequest

HttpRequest objects (Continued)

HttpRequest.headers
A case insensitive, dict-like object that provides access to all HTTP-prefixed headers (plus Content-Length and Content-Type) from the request.

HttpRequest.resolver_match
An instance of ResolverMatch representing the resolved URL. This attribute is only set after URL resolving took place, which means it’s available in all views but not in middleware which are executed before URL resolving takes place (you can use it in process_view() though).

HttpRequest

HttpRequest objects : Attributes set by application code

Django doesn’t set these attributes itself but makes use of them if set by your application.

HttpRequest.current_app
The url template tag will use its value as the current_app argument to reverse().

HttpRequest.urlconf
This will be used as the root URLconf for the current request, overriding the ROOT_URLCONF setting. See How Django processes a request for details.

urlconf can be set to None to revert any changes made by previous middleware and return to using the ROOT_URLCONF.

HttpRequest

HttpRequest objects : Attributes set by application code (Continued)

HttpRequest.exception_reporter_filter
This will be used instead of DEFAULT_EXCEPTION_REPORTER_FILTER for the current request. See Custom error reports for details.

HttpRequest.exception_reporter_class
This will be used instead of DEFAULT_EXCEPTION_REPORTER for the current request. See Custom error reports for details.

HttpRequest

HttpRequest objects :Methods

HttpRequest.get_host()
Returns the originating host of the request using information from the HTTP_X_FORWARDED_HOST (if USE_X_FORWARDED_HOST is enabled) and HTTP_HOST headers, in that order. If they don't provide a value, the method uses a combination of SERVER_NAME and SERVER_PORT as detailed in PEP 3333.

Example: "127.0.0.1:8000“

HttpRequest.get_port()
Returns the originating port of the request using information from the HTTP_X_FORWARDED_PORT (if USE_X_FORWARDED_PORT is enabled) and SERVER_PORT META variables, in that order.

HttpRequest

HttpRequest objects :Methods (Continued)

HttpRequest.get_full_path()
Returns the path, plus an appended query string, if applicable.

Example: "/music/bands/the_beatles/?print=true"

HttpRequest.get_full_path_info()
Like get_full_path(), but uses path_info instead of path.

Example: "/minfo/music/bands/the_beatles/?print=true"

HttpRequest

HttpRequest objects :Methods (Continued)

HttpRequest.build_absolute_uri(location=None)
Returns the absolute URI form of location. If no location is provided, the location will be set to request.get_full_path().

If the location is already an absolute URI, it will not be altered. Otherwise the absolute URI is built using the server variables available in this request.

HttpRequest

HttpRequest objects :Methods (Continued)

HttpRequest.get_signed_cookie(key, default=RAISE_ERROR, salt='', max_age=None)
Returns a cookie value for a signed cookie, or raises a django.core.signing.BadSignature exception if the signature is no longer valid. If you provide the default argument the exception will be suppressed and that default value will be returned instead.

The optional salt argument can be used to provide extra protection against brute force attacks on your secret key. If supplied, the max_age argument will be checked against the signed timestamp attached to the cookie value to ensure the cookie is not older than max_age seconds.

HttpRequest

HttpRequest objects :Methods (Continued)

HttpRequest.is_secure()
Returns True if the request is secure; that is, if it was made with HTTPS.

HttpRequest.accepts(mime_type)
Returns True if the request Accept header matches the mime_type argument

HttpResponse

HttpResponse objects

class **HttpResponse**

In contrast to HttpRequest objects, which are created automatically by Django, HttpResponse objects are your responsibility. Each view you write is responsible for instantiating, populating, and returning an HttpResponse.

The HttpResponse class lives in the django.http module.

HttpResponse

HttpResponse objects

Usage

Passing strings

Typical usage is to pass the contents of the page, as a string, bytestring, or memoryview, to the HttpResponse constructor:

```
>>> from django.http import HttpResponse
>>> response = HttpResponse("Here's the text of the web page.")
```

But if you want to add content incrementally, you can use response as a file-like object:

```
>>> response = HttpResponse()
>>> response.write("<p>Here's the text of the web page.</p>")
```

HttpResponse

HttpResponse objects

Usage

Passing iterators

Finally, you can pass HttpResponse an iterator rather than strings. HttpResponse will consume the iterator immediately, store its content as a string, and discard it. Objects with a close() method such as files and generators are immediately closed.

If you need the response to be streamed from the iterator to the client, you must use the StreamingHttpResponse class instead.

HttpResponse

HttpResponse objects

Setting header fields

To set or remove a header field in your response, use HttpResponse.headers:

```
>>> response = HttpResponse()
>>> response.headers['Age'] = 120
>>> del response.headers['Age’]
```

You can also manipulate headers by treating your response like a dictionary:

```
>>> response = HttpResponse()
>>> response['Age'] = 120
>>> del response['Age’]
```

This proxies to HttpResponse.headers, and is the original interface offered by HttpResponse.

HttpResponse

HttpResponse objects

Setting header fields (continued)

When using this interface, unlike a dictionary, del doesn’t raise KeyError if the header field doesn’t exist.

You can also set headers on instantiation:

```
>>> response = HttpResponse(headers={'Age': 120})
```

HttpResponse

HttpResponse objects

Telling the browser to treat the response as a file attachment

To tell the browser to treat the response as a file attachment, set the Content-Type and Content-Disposition headers. For example, this is how you might return a Microsoft Excel spreadsheet:

```
>>> response = HttpResponse(my_data, headers={
...     'Content-Type': 'application/vnd.ms-excel',
...     'Content-Disposition': 'attachment; filename="foo.xls"',
... })
```

HttpResponse

HttpResponse objects : Attributes

HttpResponse.content

A bytestring representing the content, encoded from a string if necessary.

HttpResponse.headers

New in Django 3.2.

A case insensitive, dict-like object that provides an interface to all HTTP headers on the response. See Setting header fields.

HttpResponse.charset

A string denoting the charset in which the response will be encoded. If not given at HttpResponse instantiation time, it will be extracted from content_type and if that is unsuccessful, the DEFAULT_CHARSET setting will be used.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.status_code
The HTTP status code for the response.

HttpResponse.reason_phrase
The HTTP reason phrase for the response. It uses the HTTP standard’s default reason phrases.

HttpResponse.streaming
This is always False.

HttpResponse.closed
True if the response has been closed.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.__setitem__(header, value)
Sets the given header name to the given value. Both header and value should be strings.

HttpResponse.__delitem__(header)
Deletes the header with the given name. Fails silently if the header doesn’t exist. Case-insensitive.

HttpResponse.__getitem__(header)
Returns the value for the given header name. Case-insensitive.

HttpResponse.get(header, alternate=None)
Returns the value for the given header, or an alternate if the header doesn’t exist.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.has_header(header)
Returns True or False based on a case-insensitive check for a header with the given name.

HttpResponse.items()
Acts like dict.items() for HTTP headers on the response.

HttpResponse.setdefault(header, value)
Sets a header unless it has already been set.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.set_cookie(key, value='', max_age=None, expires=None, path='/', domain=None, secure=False, httponly=False, samesite=None)¶
Sets a cookie.
max_age should be an integer number of seconds, or None (default) if the cookie should last only as long as the client’s browser session. If expires is not specified, it will be calculated.
expires should either be a string in the format "Wdy, DD-Mon-YY HH:MM:SS GMT" or a datetime.datetime object in UTC. If expires is a datetime object, the max_age will be calculated.
Use domain if you want to set a cross-domain cookie. For example, domain="example.com" will set a cookie that is readable by the domains www.example.com, blog.example.com, etc. Otherwise, a cookie will only be readable by the domain that set it.
Use secure=True if you want the cookie to be only sent to the server when a request is made with the https scheme.
Use httponly=True if you want to prevent client-side JavaScript from having access to the cookie.
HttpOnly is a flag included in a Set-Cookie HTTP response header. It’s part of the RFC 6265 standard for cookies and can be a useful way to mitigate the risk of a client-side script accessing the protected cookie data.
Use samesite='Strict' or samesite='Lax' to tell the browser not to send this cookie when performing a cross-origin request. SameSite isn’t supported by all browsers, so it’s not a replacement for Django’s CSRF protection, but rather a defense in depth measure.

HttpResponse

HttpResponse objects : Methods

HttpResponse.__init__(content=b'', content_type=None, status=200, reason=None, charset=None, headers=None)
Instantiates an HttpResponse object with the given page content, content type, and headers.

content is most commonly an iterator, bytestring, memoryview, or string. Other types will be converted to a bytestring by encoding their string representation. Iterators should return strings or bytestrings and those will be joined together to form the content of the response.

content_type is the MIME type optionally completed by a character set encoding and is used to fill the HTTP Content-Type header. If not specified, it is formed by 'text/html' and the DEFAULT_CHARSET settings, by default: "text/html; charset=utf-8".

status is the HTTP status code for the response. You can use Python’s http.HTTPStatus for meaningful aliases, such as HTTPStatus.NO_CONTENT.

reason is the HTTP response phrase. If not provided, a default phrase will be used. charset is the charset in which the response will be encoded. If not given it will be extracted from content_type, and if that is unsuccessful, the DEFAULT_CHARSET setting will be used.headers is a dict of HTTP headers for the response.TTP status code for the response.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.delete_cookie(key, path='/', domain=None, samesite=None)
Deletes the cookie with the given key. Fails silently if the key doesn’t exist.

Due to the way cookies work, path and domain should be the same values you used in set_cookie() – otherwise the cookie may not be deleted.

HttpResponse.close()
This method is called at the end of the request directly by the WSGI server.

HttpResponse.write(content)
This method makes an HttpResponse instance a file-like object.

HttpResponse.flush()
This method makes an HttpResponse instance a file-like object.

HttpResponse

HttpResponse objects : Attributes

HttpResponse.tell()
This method makes an HttpResponse instance a file-like object.

HttpResponse.getvalue()
Returns the value of HttpResponse.content. This method makes an HttpResponse instance a stream-like object.

HttpResponse.readable()
Always False. This method makes an HttpResponse instance a stream-like object.

HttpResponse.seekable()
Always False. This method makes an HttpResponse instance a stream-like object.

HttpResponse.writable()
Always True. This method makes an HttpResponse instance a stream-like object.

HttpResponse

HttpResponse subclasses

Django includes a number of HttpResponse subclasses that handle different types of HTTP responses. Like HttpResponse, these subclasses live in django.http.

class HttpResponseRedirect
The first argument to the constructor is required – the path to redirect to. This can be a fully qualified URL (e.g. 'https://www.yahoo.com/search/'), an absolute path with no domain (e.g. '/search/'), or even a relative path (e.g. 'search/'). In that last case, the client browser will reconstruct the full URL itself according to the current path. See HttpResponse for other optional constructor arguments. Note that this returns an HTTP status code 302.

url
This read-only attribute represents the URL the response will redirect to (equivalent to the Location response header).

HttpResponse

HttpResponse subclasses

class HttpResponsePermanentRedirect
Like HttpResponseRedirect, but it returns a permanent redirect (HTTP status code 301) instead of a “found” redirect (status code 302).

class HttpResponseNotModified
The constructor doesn’t take any arguments and no content should be added to this response. Use this to designate that a page hasn’t been modified since the user’s last request (status code 304).

class HttpResponseBadRequest
Acts just like HttpResponse but uses a 400 status code.

HttpResponse

HttpResponse subclasses

class HttpResponseNotFound
Acts just like HttpResponse but uses a 404 status code.

class HttpResponseForbidden
Acts just like HttpResponse but uses a 403 status code.

class HttpResponseNotAllowed
Like HttpResponse, but uses a 405 status code. The first argument to the constructor is required: a list of permitted methods (e.g. ['GET', 'POST']).

HttpResponse

HttpResponse subclasses

class HttpResponseGone

Acts just like HttpResponse but uses a 410 status code.

class HttpResponseServerError

Acts just like HttpResponse but uses a 500 status code.

GET & POST Method

Disclaimer: The content is curated for educational purposes only.

Get and Post Method

GET and POST

Django’s login form is returned using the POST method, in which the browser bundles up the form data, encodes it for transmission, sends it to the server, and then receives back its response.

GET, by contrast, bundles the submitted data into a string, and uses this to compose a URL. The URL contains the address where the data must be sent, as well as the data keys and values. You can see this in action if you do a search in the Django documentation, which will produce a URL of the form https://docs.djangoproject.com/search/?q=forms&release=1.

GET and POST are typically used for different purposes.

Get and Post Method

GET and POST

Any request that could be used to change the state of the system - for example, a request that makes changes in the database - should use POST. GET should be used only for requests that do not affect the state of the system.

GET would also be unsuitable for a password form, because the password would appear in the URL, and thus, also in browser history and server logs, all in plain text. Neither would it be suitable for large quantities of data, or for binary data, such as an image. A web application that uses GET requests for admin forms is a security risk: it can be easy for an attacker to mimic a form’s request to gain access to sensitive parts of the system. POST, coupled with other protections like Django’s CSRF protection offers more control over access.

Get and Post Method

HTML Form

In HTML, a form is a collection of elements inside <form>...</form> that allow a visitor to do things like enter text, select options, manipulate objects or controls, and so on, and then send that information back to the server.

Some of these form interface elements - text input or checkboxes - are built into HTML itself. Others are much more complex; an interface that pops up a date picker or allows you to move a slider or manipulate controls will typically use JavaScript and CSS as well as HTML form <input> elements to achieve these effects.

As well as its <input> elements, a form must specify two things:

where: the URL to which the data corresponding to the user’s input should be returned
how: the HTTP method the data should be returned by

Get and Post Method

HTML Form (Continued)

As an example, the login form for the Django admin contains several <input> elements: one of type="text" for the username, one of type="password" for the password, and one of type="submit" for the “Log in” button. It also contains some hidden text fields that the user doesn’t see, which Django uses to determine what to do next.

It also tells the browser that the form data should be sent to the URL specified in the <form>’s action attribute - /admin/ - and that it should be sent using the HTTP mechanism specified by the method attribute - post.

When the <input type="submit" value="Log in"> element is triggered, the data is returned to /admin/.

Get and Post Method

Django’s role in forms

Handling forms is a complex business. Consider Django’s admin, where numerous items of data of several different types may need to be prepared for display in a form, rendered as HTML, edited using a convenient interface, returned to the server, validated and cleaned up, and then saved or passed on for further processing.

Django’s form functionality can simplify and automate vast portions of this work, and can also do it more securely than most programmers would be able to do in code they wrote themselves.

Django handles three distinct parts of the work involved in forms:

preparing and restructuring data to make it ready for rendering
creating HTML forms for the data
receiving and processing submitted forms and data from the client

It is possible to write code that does all of this manually, but Django can take care of it all for you.

Get and Post Method

Forms in Django-

We’ve described HTML forms briefly, but an HTML <form> is just one part of the machinery required.

In the context of a web application, ‘form’ might refer to that HTML <form>, or to the Django Form that produces it, or to the structured data returned when it is submitted, or to the end-to-end working collection of these parts.

Get and Post Method

The Django Form class

At the heart of this system of components is Django’s Form class. In much the same way that a Django model describes the logical structure of an object, its behavior, and the way its parts are represented to us, a Form class describes a form and determines how it works and appears.

In a similar way that a model class’s fields map to database fields, a form class’s fields map to HTML form <input> elements. (A ModelForm maps a model class’s fields to HTML form <input> elements via a Form; this is what the Django admin is based upon.)

A form’s fields are themselves classes; they manage form data and perform validation when a form is submitted. A DateField and a FileField handle very different kinds of data and have to do different things with it.

A form field is represented to a user in the browser as an HTML “widget” - a piece of user interface machinery. Each field type has an appropriate default Widget class, but these can be overridden as required.

Get and Post Method

Instantiating, processing, and rendering forms

When rendering an object in Django, we generally:

get hold of it in the view (fetch it from the database, for example)
pass it to the template context
expand it to HTML markup using template variables

Rendering a form in a template involves nearly the same work as rendering any other kind of object, but there are some key differences.

In the case of a model instance that contained no data, it would rarely if ever be useful to do anything with it in a template. On the other hand, it makes perfect sense to render an unpopulated form - that's what we do when we want the user to populate it.

So when we handle a model instance in a view, we typically retrieve it from the database. When we're dealing with a form we typically instantiate it in the view.

Get and Post Method

Instantiating, processing, and rendering forms (Continued)

When we instantiate a form, we can opt to leave it empty or pre-populate it, for example with:

data from a saved model instance (as in the case of admin forms for editing)
data that we have collated from other sources
data received from a previous HTML form submission

The last of these cases is the most interesting, because it’s what makes it possible for users not just to read a website, but to send information back to it too.

Get and Post Method

Building a form

The work that needs to be done

Suppose you want to create a simple form on your website, in order to obtain the user’s name. You’d need something like this in your template:

```
<form action="/your-name/" method="post">
        <label for="your_name">Your name: </label>
        <input id="your_name" type="text" name="your_name" value="{{ current_name }}">
        <input type="submit" value="OK">
 </form>
```

This tells the browser to return the form data to the URL /your-name/, using the POST method. It will display a text field, labeled “Your name:”, and a button marked “OK”. If the template context contains a current_name variable, that will be used to pre-fill the your_name field.

Get and Post Method

Building a form (Continued)

You’ll need a view that renders the template containing the HTML form, and that can supply the current_name field as appropriate.
When the form is submitted, the POST request which is sent to the server will contain the form data.
Now you’ll also need a view corresponding to that /your-name/ URL which will find the appropriate key/value pairs in the request, and then process them.

This is a very simple form. In practice, a form might contain dozens or hundreds of fields, many of which might need to be pre-populated, and we might expect the user to work through the edit-submit cycle several times before concluding the operation.

We might require some validation to occur in the browser, even before the form is submitted; we might want to use much more complex fields, that allow the user to do things like pick dates from a calendar and so on.
At this point it’s much easier to get Django to do most of this work for us.

Get and Post Method

Building a form in Django

The Form class

We already know what we want our HTML form to look like. Our starting point for it in Django is this:

forms.py

```
from django import forms
class NameForm(forms.Form):
            your_name = forms.CharField(label='Your name', max_length=100)
```

This defines a **Form** class with a single field **(your_name).** We've applied a human-friendly label to the field, which will appear in the **<label>** when it's rendered (although in this case, the **label** we specified is actually the same one that would be generated automatically if we had omitted it).

Get and Post Method

Building a form in Django (Continued)

The field’s maximum allowable length is defined by **max_length**. This does two things. It puts a **maxlength="100**" on the HTML **<input>** (so the browser should prevent the user from entering more than that number of characters in the first place). It also means that when Django receives the form back from the browser, it will validate the length of the data.

A **Form** instance has an **is_valid()** method, which runs validation routines for all its fields. When this method is called, if all fields contain valid data, it will:

- return **True**
- place the form’s data in its **cleaned_data** attribute.

The whole form, when rendered for the first time, will look like:

```
<label for="your_name">Your name: </label>
<input id="your_name" type="text" name="your_name" maxlength="100" required>
```

Note that it does not include the <form> tags, or a submit button. We’ll have to provide those ourselves in the template.

Get and Post Method

Building a form in Django (Continued) : The view

Form data sent back to a Django website is processed by a view, generally the same view which published the form. This allows us to reuse some of the same logic.
To handle the form we need to instantiate it in the view for the URL where we want it to be published:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

from .forms import NameForm

def get_name(request):
    # if this is a POST request we need to process the form data
    if request.method == 'POST':
        # create a form instance and populate it with data from the request:
        form = NameForm(request.POST)
        # check whether it's valid:
        if form.is_valid():
            # process the data in form.cleaned_data as required
            # ...
            # redirect to a new URL:
            return HttpResponseRedirect('/thanks/')

    # if a GET (or any other method) we'll create a blank form
    else:
        form = NameForm()
    return render(request, 'name.html', {'form': form})
```

Get and Post Method

Building a form in Django : The view (Continued)

If we arrive at this view with a GET request, it will create an empty form instance and place it in the template context to be rendered. This is what we can expect to happen the first time we visit the URL.

If the form is submitted using a POST request, the view will once again create a form instance and populate it with data from the request: form = NameForm(request.POST) This is called “binding data to the form” (it is now a bound form).

We call the form’s is_valid() method; if it’s not True, we go back to the template with the form. This time the form is no longer empty (unbound) so the HTML form will be populated with the data previously submitted, where it can be edited and corrected as required.

If is_valid() is True, we’ll now be able to find all the validated form data in its cleaned_data attribute. We can use this data to update the database or do other processing before sending an HTTP redirect to the browser telling it where to go next.

Get and Post Method

Building a form in Django (Continued) : The template

We don’t need to do much in our name.html template:

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

All the form’s fields and their attributes will be unpacked into HTML markup from that {{ form }} by Django’s template language.
We now have a working web form, described by a Django Form, processed by a view, and rendered as an HTML <form>.

Get and Post Method

Working with form templates

All you need to do to get your form into a template is to place the form instance into the template context. So if your form is called **form** in the context, {{ form }} will render its **<label>** and **<input>** elements appropriately.

Form rendering options:
There are other output options though for the <label>/<input> pairs:

- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags

- Note that you'll have to provide the surrounding <table> or <ul> elements yourself.

Template, Render, Views, Context Template Editing

Disclaimer: The content is curated for educational purposes only.

Templates

- Templates are the third and most important part of Django's MVT Structure. A template in Django is basically written in HTML, CSS, and JavaScript in a .html file. Django framework efficiently handles and generates dynamically HTML web pages that are visible to the end-user.
- Django mainly functions with a backend so, in order to provide a frontend and provide a layout to our website, we use templates.
- There are two methods of adding the template to our website depending on our needs.

  1) We can use a single template directory which will be spread over the entire project.

  2) For each app of our project, we can create a different template directory.

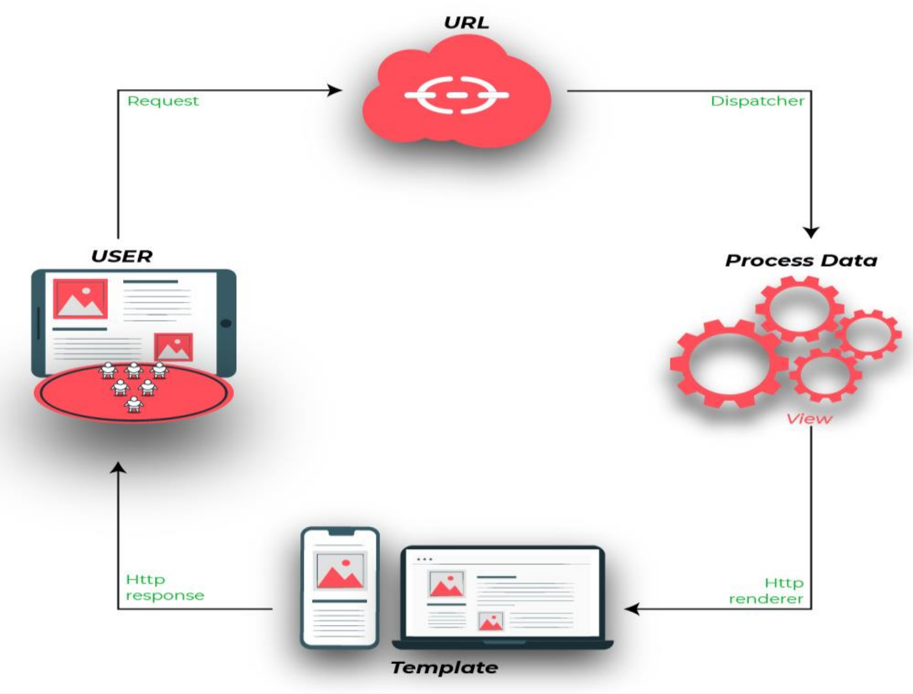


Image Source: https://media.geeksforgeeks.org/wp-content/uploads/20200124153519/django-views.jpg

The Django template language

- A Django template is a text document or a Python string marked-up using the Django template language.
- Some constructs are recognized and interpreted by the template engine.
- The main ones are variables and tags.
- The main characteristics of Django Template language are Variables, Tags, Filters, and Comments.

Let us discuss main characteristics one by one

Variables

Variables output a value from the context, which is a dict-like object mapping keys to values. The context object we sent from the view can be accessed in the template using variables of Django Template.

Syntax

{{ variable_name }}

Example

-Variables are surrounded by {{ and }} like this:

Eg.-My first name is {{ first_name }}. My last name is {{ last_name }}.

With a context of {‘first_name’: ‘Pawan’, ‘last_name’: ‘Kumar’}, this template renders to:

My first name is Pawan. My last name is Kumar.

Let us discuss main characteristics one by one (Continued)

Tags

It provide arbitrary logic in the rendering process

Syntax

{% tag_name %}

Example

Tags are surrounded by {% and %} like this:

{% csrf_token %}

Most tags accept arguments, for example :

{% cycle 'odd' 'even' %}

Let us discuss main characteristics one by one (Continued)

Filters

Django Template Engine provides filters that are used to transform the values of variables and tag arguments.

Tags can’t modify the value of a variable whereas filters can be used for incrementing the value of a variable or modifying it to one’s own need.

Syntax

{{ variable_name | filter_name }}

Filters can be “chained.” The output of one filter is applied to the next.

{{ text|escape|linebreaks }} is a common idiom for escaping text contents, then converting line breaks to <p> tags.

Example

{{ value | length }}

If value is [‘a’, ‘b’, ‘c’, ‘d’], the output will be 4.

Let us discuss main characteristics one by one (Continued)

Comments

Template ignores everything between {% comment %} and {% end comment %}.

An optional note may be inserted in the first tag.

For example, this is useful when commenting out code for documenting why the code was disabled.

Syntax

```
{% comment 'comment_name' %}
{% endcomment %}
```

Example :

```
{% comment "Optional note" %}
    Commented out text with {{ create_date|date:"c" }}
{% endcomment %}
```

Let us discuss main characteristics one by one (Continued)

Template Inheritance

Template inheritance allows you to build a base “skeleton” template that contains all the common elements of your site and defines blocks that child templates can override. extends tag is used for the inheritance of templates in Django. One needs to repeat the same code again and again. Using extends we can inherit templates as well as variables.

Syntax

```
{% extends 'template_name.html' %}
```

Let us discuss main characteristics one by one (Continued)

Template Inheritance

Example :

assume the following directory structure:

```
dir1/
    template.html
    base2.html
    my/
                    base3.html
base1.html
```

In template.html, the following paths would be valid:

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./my/base3.html" %}
```

Example on Django Template : Let us create one template and render it .

Step1) Create view.py

```
# import Http Response from django
from django.shortcuts import render

# create a function
def learn_view(request):
    # create a dictionary to pass
    # data to the template
    context ={
        "data":"Updating from the list",
        "list":['Data Science', 'Python', 'Django', 'HTML5','JavaScript']
    }
    # return response with template and context
    return render(request, "learn.html", context)
```

Example on Django Template : Let us create one template and render it .

Step2) URL Mapping: open urls.py

```
from django.urls import path

# importing views from views..py
from .views import learn_view

urlpatterns = [
    path('', learn_view),
]
```

Example on Django Template : Let us create one template and render it .

Step3) Create template

Create folder named as template and create new file names learn.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Homepage</title>
</head>
<body>
    <h1>Welcome to Learn Django.</h1>
    <p> Data is {{ data }}</p>
    <h4>List is </h4>
    <ul>
    {% for i in list %}
    <li>{{ i }}</li>
    {% endfor %}
</body>
</html>
```

Example on Django Template : Let us create one template and render it .

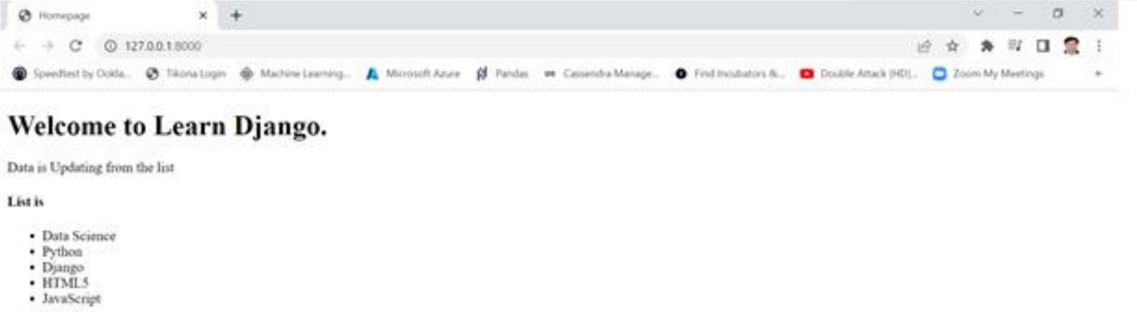
Step4) Open settings.py and Configure TEMPLATES section

-Copy path of your template directory and paste under TEMPLATES section in seetings.py

```
'DIRS': ['C:\Users\Hp\Desktop\learndjango\learndjango\templates'],
```

Example on Django Template : Let us create one template and render it .

Step5) check output : Visit http://127.0.0.1:8000/



Welcome to Learn Django.

Data is Updating from the list

List is

- Data Science
- Python
- Django
- HTML5
- JavaScript

SQL operations in Django
Django Models

Disclaimer: The content is curated for educational purposes only.

Django Models

Disclaimer: The content is curated for educational purposes only.

Django Models

- A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.
- A Django model is the built-in feature that Django uses to create tables, their fields, and various constraints. In short, Django Models is the SQL of Database one uses with Django.
- SQL (Structured Query Language) is complex and involves a lot of different queries for creating, deleting, updating or any other stuff related to database. Django models simplify the tasks and organize tables into models. Generally, each model maps to a single database table.
- Django models provide simplicity, consistency, version control and advanced metadata handling.
- Basics of a model include –
  - Each model is a Python class that subclasses django.db.models.Model.
  - Each attribute of the model represents a database field.
  - With all of this, Django gives you an automatically-generated database-access API

Quick Example : Creating Models

- This example model defines a **Person**, which has a first_name and last_name:

```
from django.db import models
class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

first_name and last_name are fields of the model. Each field is specified as a class attribute, and each attribute maps to a database column.

The **Person** model would create a database table like this:

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Using Models

- Once you have defined your models, you need to tell Django you're going to use those models. Do this by editing your settings file and changing the INSTALLED_APPS setting to add the name of the module that contains your models.py.
- For example, if the models for your application live in the module myapp.models (the package structure that is created for an application by the manage.py startapp script), INSTALLED_APPS should read, in part:

```
INSTALLED_APPS = [
    #...
    'myapp',
    #...
]
```

- When you add new apps to INSTALLED_APPS, be sure to run following commands for making migrations

```
> manage.py makemigrations
> manage.py migrate
```

Fields : -creating model Fields

- The most important part of a model – and the only required part of a model – is the list of database fields it defines. Fields are specified by class attributes. Be careful not to choose field names that conflict with the models API like clean, save, or delete.
- Example:

```
from django.db import models
class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)
class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

Fields types : -creating model Fields

- Each field in your model should be an instance of the appropriate Field class. Django uses the field class types to determine a few things:

The column type, which tells the database what kind of data to store (e.g. INTEGER, VARCHAR, TEXT).
The default HTML widget to use when rendering a form field (e.g. <input type="text">, <select>).
The minimal validation requirements, used in Django’s admin and in automatically-generated forms.

- Django ships with dozens of built-in field types; you can find the complete list in the model field reference.
- Few Filed types are listed below:
- AutoField
- CharField
- DateField
- ImageField
- DecimalField

IntegerField

TextField

FileField

EmailField

JSONField

Fields options : -creating model Fields

- Each field takes a certain set of field-specific arguments. For example, CharField (and its subclasses) require a max_length argument which specifies the size of the VARCHAR database field used to store the data.
- Here’s a quick summary of the most often-used ones:

1) null -If True, Django will store empty values as NULL in the database. Default is False.
2) blank -If True, the field is allowed to be blank. Default is False.
3) choices- A sequence of 2-tuples to use as choices for this field. If this is given, the default form widget will be a select box instead of the standard text field and will limit choices to the choices given.

A choices list looks like this:

```
YEAR_IN_SCHOOL_CHOICES = [
                    ('FR', 'Freshman'),
                    ('SO', 'Sophomore'),
                     ('JR', 'Junior'),
                      ('SR', 'Senior'),
                       ('GR', 'Graduate’), ]
```

SQL operations in Django

Disclaimer: The content is curated for educational purposes only.

Making queries :create model

- Once you’ve created your data models, Django automatically gives you a database-abstraction API that lets you create, retrieve, update and delete objects. we’ll refer to the following models

```
from datetime import date
from django.db import models
class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()
    def __str__(self):
        return self.name
class Author(models.Model):
    name = models.CharField(max_length=200)
    email = models.EmailField()
    def __str__(self):
        return self.name
class Entry(models.Model):
    blog = models.ForeignKey(Blog, on_delete=models.CASCADE)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
    mod_date = models.DateField(default=date.today)
    authors = models.ManyToManyField(Author)
    number_of_comments = models.IntegerField(default=0)
    number_of_pingbacks = models.IntegerField(default=0)
    rating = models.IntegerField(default=5)
    def __str__(self):
        return self.headline
```

Making queries : Creating objects

- A model class represents a database table, and an instance of that class represents a particular record in the database table.
- To create an object, instantiate it using keyword arguments to the model class, then call save() to save it to the database.
- Assuming models live in a file mysite/blog/models.py, here's an example:

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')
>>> b.save()
```

This performs an INSERT SQL statement behind the scenes. Django doesn't hit the database until you explicitly call save().

The save() method has no return value.

- To create and save an object in a single step, use the create() method.

Making queries : Saving changes to objects

- To save changes to an object that’s already in the database, use save().
- Given a Blog instance b5 that has already been saved to the database, this example changes its name and updates its record in the database:

```
>>> b5.name = 'New name'
>>> b5.save()
```

- This performs an UPDATE SQL statement behind the scenes. Django doesn’t hit the database until you explicitly call save().

Making queries : Retrieving objects

- To retrieve objects from your database, construct a QuerySet via a Manager on your model class.
- A QuerySet represents a collection of objects from your database. It can have zero, one or many filters. Filters narrow down the query results based on the given parameters. In SQL terms, a QuerySet equates to a SELECT statement, and a filter is a limiting clause such as WHERE or LIMIT.
- You get a QuerySet by using your model's Manager. Each model has at least one Manager, and it's called objects by default. Access it directly via the model class, like so:
- >>> Blog.objects
- <django.db.models.manager.Manager object at ...>
- >>> b = Blog(name='Foo', tagline='Bar')
- >>> b.objects
- Traceback:
- ...
- AttributeError: "Manager isn't accessible via Blog instances.“
- The Manager is the main source of QuerySets for a model. For example, Blog.objects.all() returns a QuerySet that contains all Blog objects in the database.
- Retrieving all objects¶
- The simplest way to retrieve objects from a table is to get all of them. To do this, use the all() method on a Manager:
- >>> all_entries = Entry.objects.all()
- The all() method returns a QuerySet of all the objects in the database.

Making queries : Retrieving specific objects with filters

- The QuerySet returned by all() describes all objects in the database table. Usually, though, you'll need to select only a subset of the complete set of objects.

- To create such a subset, you refine the initial QuerySet, adding filter conditions. The two most common ways to refine a QuerySet are:
- filter(**kwargs)
- -Returns a new QuerySet containing objects that match the given lookup parameters.
-
- exclude(**kwargs)
- -Returns a new QuerySet containing objects that do not match the given lookup parameters.
- The lookup parameters (**kwargs in the above function definitions) should be in the format described in Field lookups below:
- For example, to get a QuerySet of blog entries from the year 2006, use filter() like so:
- Entry.objects.filter(pub_date__year=2006)

- With the default manager class, it is the same as:
- Entry.objects.all().filter(pub_date__year=2006)

Making queries :Retrieving a single object with get()

- filter() will always give you a QuerySet, even if only a single object matches the query - in this case, it will be a QuerySet containing a single element.

- If you know there is only one object that matches your query, you can use the get() method on a Manager which returns the object directly:

- >>> one_entry = Entry.objects.get(pk=1)

Making queries : Limiting QuerySets

- Use a subset of Python’s array-slicing syntax to limit your QuerySet to a certain number of results. This is the equivalent of SQL’s LIMIT and OFFSET clauses.

- For example, this returns the first 5 objects (LIMIT 5):
- >>> Entry.objects.all()[:5]

- This returns the sixth through tenth objects (OFFSET 5 LIMIT 5):
- >>> Entry.objects.all()[5:10]
- Negative indexing (i.e. Entry.objects.all()[-1]) is not supported

Performing raw SQL queries

- Django gives you two ways of performing raw SQL queries: you can use Manager.raw() to perform raw queries and return model instances, or you can avoid the model layer entirely and execute custom SQL directly.

- The raw() manager method can be used to perform raw SQL queries that return model instances:
- Manager.raw(raw_query, params=(), translations=None)
- This method takes a raw SQL query, executes it, and returns a django.db.models.query.RawQuerySet instance. This RawQuerySet instance can be iterated over like a normal QuerySet to provide object instances.

- This is best illustrated with an example. Suppose you have the following model:

```
class Person(models.Model):
    first_name = models.CharField(...)
    last_name = models.CharField(...)
    birth_date = models.DateField(...)
```

Performing raw SQL queries (Continued)

- You could then execute custom SQL like so:

```
>>> for p in Person.objects.raw('SELECT * FROM myapp_person'):
...     print(p)
John Smith
Jane Jones
```

- it’s exactly the same as running Person.objects.all().
- Index lookups:
- raw() supports indexing, so if you need only the first result you can write:
- >>> first_person = Person.objects.raw('SELECT * FROM myapp_person')[0]

Django CRUD (Create, Retrieve, Update, Delete) Function Based Views

- Django is a Python-based web framework which allows you to quickly create web application without all of the installation or dependency problems that you normally will find with other frameworks.
- Django is based on MVT (Model View Template) architecture and revolves around CRUD (Create, Retrieve, Update, Delete) operations.
- CRUD means performing Create, Retrieve, Update and Delete operations on a table in a database.

Django CRUD operations

- Create – create or add new entries in a table in the database.
- Retrieve – read, retrieve, search, or view existing entries as a list(List View) or retrieve a particular entry in detail (Detail View)
- Update – update or edit existing entries in a table in the database
- Delete – delete, deactivate, or remove existing entries in a table in the database

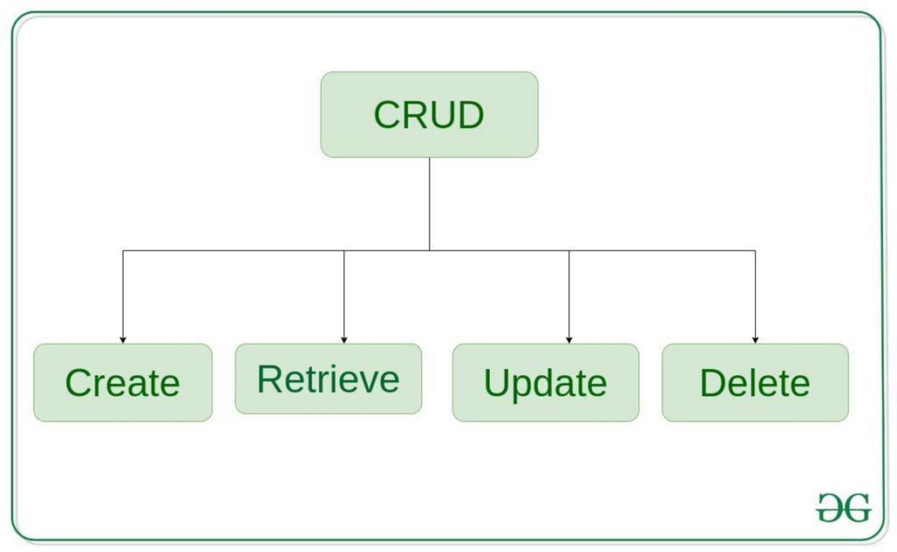


Image Source: https://media.geeksforgeeks.org/wp-content/uploads/20200114185631/Untitled-Diagram-316-1024x630.jpg

Handling sessions, cookies and Working with JSON and AJAX

Disclaimer: The content is curated for educational purposes only.

Handling sessions

Disclaimer: The content is curated for educational purposes only.

What is session and How to use sessions:

- The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the sending and receiving of cookies. Cookies contain a session ID – not the data itself (unless you're using the cookie based backend).
- Django provides full support for anonymous sessions.

Enabling sessions

- Sessions are implemented via a piece of middleware.
- To enable session functionality, do the following:
- Edit the MIDDLEWARE setting and make sure it contains 'django.contrib.sessions.middleware.SessionMiddleware'. The default settings.py created by django-admin startproject has SessionMiddleware activated.
- If you don't want to use sessions, you might as well remove the SessionMiddleware line from MIDDLEWARE and 'django.contrib.sessions' from your INSTALLED_APPS. It'll save you a small bit of overhead.

Configuring the session engine

- By default, Django stores sessions in your database (using the model django.contrib.sessions.models.Session). Though this is convenient, in some setups it’s faster to store session data elsewhere, so Django can be configured to store session data on your filesystem or in your cache.

Using database-backed sessions

- If you want to use a database-backed session, you need to add 'django.contrib.sessions' to your INSTALLED_APPS setting.
- Once you have configured your installation, run manage.py migrate to install the single database table that stores session data.

Using file-based sessions

- To use file-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.file".
- You might also want to set the SESSION_FILE_PATH setting (which defaults to output from tempfile.gettempdir(), most likely /tmp) to control where Django stores session files. Be sure to check that your web server has permissions to read and write to this location.

Using cookie-based sessions

- To use cookies-based sessions, set the SESSION_ENGINE setting to "django.contrib.sessions.backends.signed_cookies".
- The session data will be stored using Django’s tools for cryptographic signing and the SECRET_KEY setting.

Using sessions in views

- When SessionMiddleware is activated, each HttpRequest object – the first argument to any Django view function – will have a session attribute, which is a dictionary-like object.
- You can read it and write to request.session at any point in your view. You can edit it multiple times.

  class backends.base.SessionBase

- This is the base class for all session objects. It has the following standard dictionary methods:

  __getitem__(key)

  Example: fav_color = request.session['fav_color']

- __setitem__(key, value)

  Example: request.session['fav_color'] = 'blue'’

- __delitem__(key)¶

  Example: del request.session['fav_color']. This raises KeyError if the given key isn’t already in the session.

Using sessions in views (Continued)

- ___contains__(key)

  Example: 'fav_color' in request.session

- get(key, default=None)

  Example: fav_color = request.session.get('fav_color', 'red')

- pop(key, default=__not_given)

  Example: fav_color = request.session.pop('fav_color', 'blue’)

- keys()
- items()
- setdefault()
- clear()

Using sessions in views (Continued)

It also has these methods:

flush()

Deletes the current session data from the session and deletes the session cookie. This is used if you want to ensure that the previous session data can't be accessed again from the user's browser (for example, the django.contrib.auth.logout() function calls it).

set_test_cookie()

Sets a test cookie to determine whether the user's browser supports cookies. Due to the way cookies work, you won't be able to test this until the user's next page request.

test_cookie_worked()

Returns either True or False, depending on whether the user's browser accepted the test cookie. Due to the way cookies work, you'll have to call set_test_cookie() on a previous, separate page request.

Using sessions in views (Continued)

delete_test_cookie()

Deletes the test cookie. Use this to clean up after yourself.

get_session_cookie_age()

Returns the value of the setting SESSION_COOKIE_AGE. This can be overridden in a custom session backend.

set_expiry(value)

Sets the expiration time for the session. You can pass a number of different values:

- If value is an integer, the session will expire after that many seconds of inactivity. For example, calling request.session.set_expiry(300) would make the session expire in 5 minutes.
- If value is a datetime or timedelta object, the session will expire at that specific date/time. Note that datetime and timedelta values are only serializable if you are using the PickleSerializer.
- If value is 0, the user's session cookie will expire when the user's web browser is closed.
- If value is None, the session reverts to using the global session expiry policy.
- Reading a session is not considered activity for expiration purposes. Session expiration is computed from the last time the session was modified.

Using sessions in views (Continued)

get_expiry_age()

Returns the number of seconds until this session expires. For sessions with no custom expiration (or those set to expire at browser close), this will equal SESSION_COOKIE_AGE.

This function accepts two optional keyword arguments:

- modification: last modification of the session, as a datetime object. Defaults to the current time.
- expiry: expiry information for the session, as a datetime object, an int (in seconds), or None. Defaults to the value stored in the session by set_expiry(), if there is one, or None.

get_expiry_date()

Returns the date this session will expire. For sessions with no custom expiration (or those set to expire at browser close), this will equal the date SESSION_COOKIE_AGE seconds from now.

This function accepts the same keyword arguments as get_expiry_age(), and similar notes on usage apply.

Using sessions in views (Continued)

get_expire_at_browser_close()

Returns either True or False, depending on whether the user’s session cookie will expire when the user’s web browser is closed.

clear_expired()

Removes expired sessions from the session store. This class method is called by clearsessions.

cycle_key()

Creates a new session key while retaining the current session data. django.contrib.auth.login() calls this method to mitigate against session fixation.

Working with JSON and AJAX

Disclaimer: The content is curated for educational purposes only.

Working with JSON

What is JSON

- JSON (JavaScript Object Notation) is a lightweight data-interchange format. The official Internet media type for JSON is application/json. The JSON filename extension is .json. It is easy for humans to read and write and for machines to parse and generate.

Django JsonResponse

- JsonResponse is an HttpResponse subclass that helps to create a JSON-encoded response. Its default Content-Type header is set to application/json. The first parameter, data, should be a dict instance. If the safe parameter is set to False, any object can be passed for serialization; otherwise only dict instances are allowed

JsonResponse objects

class JsonResponse(data, encoder=DjangoJSONEncoder, safe=True, json_dumps_params=None, **kwargs)

An HttpResponse subclass that helps to create a JSON-encoded response. It inherits most behavior from its superclass with a couple differences:

Its default Content-Type header is set to application/json.

The first parameter, data, should be a dict instance. If the safe parameter is set to False (see below) it can be any JSON-serializable object.

The encoder, which defaults to django.core.serializers.json.DjangoJSONEncoder, will be used to serialize the data. See JSON serialization for more details about this serializer.

The safe boolean parameter defaults to True. If it’s set to False, any object can be passed for serialization (otherwise only dict instances are allowed). If safe is True and a non-dict object is passed as the first argument, a TypeError will be raised.

The json_dumps_params parameter is a dictionary of keyword arguments to pass to the json.dumps() call used to generate the response.

JsonResponse objects (Continued)

Usage

Typical usage could look like:

```
>>> from django.http import JsonResponse
>>> response = JsonResponse({'foo': 'bar'})
>>> response.content
b'{"foo": "bar"}’
```

Serializing non-dictionary objects

In order to serialize objects other than dict you must set the safe parameter to False:

```
>>> response = JsonResponse([1, 2, 3], safe=False)
```

Without passing safe=False, a TypeError will be raised.

JsonResponse objects (Continued)

Changing the default JSON encoder

If you need to use a different JSON encoder class you can pass the encoder parameter to the constructor method:

```
>>> response = JsonResponse(data, encoder=MyJSONEncoder)
```

Django JsonResponse example :

This example demonstrate how to send JSON data in Django

- Step 1) open command prompt and execute the following commands

```
> mkdir jsonresponse
> cd jsonresponse
> mkdir src
> cd src
```

We create the project and the and src directories. Then we locate to the src directory.

- step 2) run the command

```
> django-admin startproject jsonresponse .
```

We create a new Django project in the src directory.

Note: If the optional destination is provided, Django will use that existing directory as the project directory. If it is omitted, Django creates a new directory based on the project name. We use the dot (.) to create a project inside the current working directory.

We locate to the project directory.

Django JsonResponse example : (Continued)

- Step 3) open command prompt and execute the following command to show the tree structure

```
> tree /f
```

```
src
│   manage.py
│
└───jsonresponse
        settings.py
        urls.py
        views.py
        wsgi.py
        __init__.py
```

- Note: The Django way is to put functionality into apps, which are created with django-admin startapp. In this tutorial, we do not use an app to make the example simpler. We focus on demonstrating how to send JSON response

Django JsonResponse example :(Continued)

- Step 4) open file src/jsonresponse/urls.py and add following code

```
from django.contrib import admin
from django.urls import path
from .views import send_json

urlpatterns = [
   path('admin/', admin.site.urls),
   path('sendjson/', send_json, name='send_json'),
]
```

We add a new route page; it calls the send_json() function from the views.py module.

Django JsonResponse example :(Continued)

- Step 5) create new views.py under src/jsonresponse/views.py and add following code

```
from django.http import JsonResponse


def send_json(request):


    data = [{'name': 'Peter', 'email': 'peter@example.org'},
          {'name': 'Julia', 'email': 'julia@example.org'}]


    return JsonResponse(data, safe=False)
```

- Inside send_json(), we define a list of dictionaries. Since it is a list, we set safe to False. If we did not set this parameter, we would get a TypeError with the following message:

  In order to allow non-dict objects to be serialized set the safe parameter to False.

Django JsonResponse example :(Continued)

- Step 6) open command prompt and run the command

  > python manage.py runserver

- Step 7) We run the server and navigate to http://127.0.0.1:8000/sendjson/
- Step 8) We use the curl tool to make the GET request .open command prompt and run command

  > curl localhost:8000/sendjson/

  It shows output as follows:

```
[{"name": "Peter", "email": "peter@example.org"},
{"name": "Julia", "email": "julia@example.org"}]
```

Working with AJAX

Disclaimer: The content is curated for educational purposes only.

In this module, we learnt about

- Web Framework, Django Introduction, Django Architecture
- Django MVC, MVT (Model View Template)
- Views and URL mapping, HttpRequest and HttpResponse , GET and POST Method
- Template, Render, Views, Context
- Template Editing
- SQL operation in django
- Handling sessions, cookies and working with JSON and AJAX

Working with AJAX

What is AJAX

- AJAX stands for Asynchronous JavaScript And XML, which allows web pages to update asynchronously by exchanging data to and from the server.
- This means you can update parts of a web page without reloading the complete web page.
- It involves a combination of a browser built-in XMLHttpRequest object, JavaScript, and HTML DOM.

How AJAX Works

- An event occurs on a web page, such as an initial page load, form submission, link or button click, etc.
- An XMLHttpRequest object is created and sends the request to the server .
- The server responds to the request.
- The response is captured and then server respond back with response data.
- There are many scenarios where you may want to make GET and POST requests to load and post data from the server asynchronously, back and forth. Additionally, this enables web applications to be more dynamic and reduces page load time.

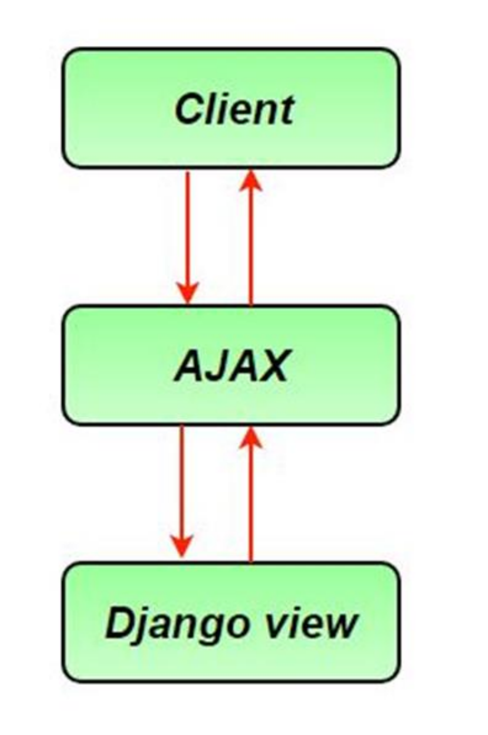


Image Source: https://media.geeksforgeeks.org/wp-content/uploads/ajax.jpg

Web References :

- Web framework : https://en.wikipedia.org/wiki/Web_framework
- What is Django : https://en.wikipedia.org/wiki/Django_(web_framework)
- MVC : https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm
- MVT: https://www.geeksforgeeks.org/django-project-mvt-structure/
- Views : https://docs.djangoproject.com/en/4.0/topics/http/views/
- https://www.geeksforgeeks.org/views-in-django-python/
- SQL : https://docs.djangoproject.com/en/4.0/topics/db/sql/
- Sessions : https://docs.djangoproject.com/en/4.0/topics/http/sessions/
- AJAX : https://www.pluralsight.com/guides/work-with-ajax-Django
- Django : https://www.guru99.com/django-tutorial.html#5
- Django : https://www.javatpoint.com/django-tutorial
- Request and Response objects : https://docs.djangoproject.com/en/4.0/ref/request-response/
- Get and post method : https://docs.djangoproject.com

Web References :

- Django project https://docs.djangoproject.com/en/4.0/intro/tutorial01/
- https://docs.djangoproject.com/en/4.0/intro/tutorial02/

Thank you

Disclaimer: The content is curated for educational purposes only.