CLASS:BCA3rdSem

Batch: 2019-2021

Python

Notes as per IKGPTU Syllabus

Name of Faculty: Ms<Jatinderpal Kaur>

Faculty of IT Department, SBS College. Ludhiana

Unit-I **Introduction to Python Programming Language:** Programming Language, History and Origin of Python Language, Features of Python, Limitations, Major Applications of Python, Getting, Installing Python, Setting up Path and Environment Variables, Running Python, First Python Program, Python Interactive Help Feature, Python differences from other languages. **Python Data Types & Input/Output:** Keywords, Identifiers, Python Statement, Indentation, Documentation, Variables, Multiple Assignment, Understanding Data Type, Data Type Conversion, Python Input and Output Functions, Import command. **Operators and Expressions:** Operators in Python, Expressions, Precedence, Associativity of Operators, Non Associative Operators.	6-34

Unit-II **Control Structures:** Decision making statements, Python loops, Python control statements. **Python Native Data Types:** Numbers, Lists, Tuples, Sets, Dictionary, Functions & Methods of Dictionary, Strings (in detail with their methods and operations).	34-74
Unit-III **Python Functions:** Functions, Advantages of Functions, Built-in Functions, User defined functions, Anonymous functions, Pass by value Vs. Pass by Reference, Recursion, Scope and Lifetime of Variables. **Python Modules:** Module definition, Need of modules, Creating a module, Importing module, Path Searching of a Module, Module Reloading, Standard Modules, Python Packages.	75-92

Unit-IV **Exception Handling:** Exceptions, Built-in exceptions, Exception handling, User defined exceptions in Python. **File Management in Python:** Operations on files (opening, modes, attributes, encoding, closing), read() & write() methods, tell() & seek() methods, renaming & deleting files in Python, directories in Python. **Classes and Objects:** The concept of OOPS in Python, Designing classes, Creating objects, Accessing attributes, Editing class attributes, Built-in class attributes, Garbage collection, Destroying objects.	93-101

Unit- 1

What is Python?

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

It is used for:

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

What can Python do?

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

Why Python?

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

Python History and Versions

- Python laid its foundation in the late 1980s.
- The implementation of Python was started in the December 1989 by **Guido Van Rossum** at CWI in Netherland.
- In February 1991, van Rossum published the code (labeled version 0.9.0) to alt.sources.
- In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce.
- Python 2.0 added new features like: list comprehensions, garbage collection system.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.

- *ABC programming language* is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Python is influenced by following programming languages:
    - ABC language.
    - Modula-3

Python Version List

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.

A list of python versions with its released date is given below.

Python Version	**Released Date**
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000
Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018

.

Python Features

Python provides lots of features that are listed below.

1) Easy to Learn and Use

Python is easy to learn and use. It is developer-friendly and high level programming language.

---

2) Expressive Language

Python language is more expressive means that it is more understandable and readable.

---

3) Interpreted Language

Python is an interpreted language i.e. interpreter executes the code line by line at a time. This makes debugging easy and thus suitable for beginners.

---

4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

---

5) Free and Open Source

Python language is freely available at offical web address.The source-code is also available. Therefore it is open source.

---

6) Object-Oriented Language

Python supports object oriented language and concepts of classes and objects come into existence.

---

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our python code.

---

8) Large Standard Library

Python has a large and broad library and provides rich set of module and functions for rapid application development.

---

9) GUI Programming Support

Graphical user interfaces can be developed using Python.

---

10) Integrated

It can be easily integrated with languages like C, C++, JAVA etc.

What are the drawbacks of Python?

Disadvantages of Python are:

Speed

Python is **slower** than C or C++. But of course, **Python** is a high-level language, unlike C or C++ it's not closer to hardware.

Mobile Development

Python is not a very good language for **mobile development** . It is seen as a **weak language** for mobile computing. This is the reason very few mobile applications are built in it like Carbonnelle.

Memory Consumption

Python is not a good choice for **memory intensive** tasks. Due to the flexibility of the data-types, Python's memory consumption is also high.

Database Access

Python has limitations with **database access** . As compared to the popular technologies like **JDBC and ODBC**, the Python's database access layer is found to be bit underdeveloped and **primitive** . However, it cannot be applied in the enterprises that need smooth interaction of complex **legacy data** .

Runtime Errors

Python programmers cited several issues with the **design** of the language. Because the language is **dynamically typed** , it requires more testing and has errors that only show up at **runtime** .

Add Python to the Windows Path

If you've installed Python in Windows using the default installation options, the path to the Python executable wasn't added to the Windows **Path variable**. The Path variable lists the directories that will be searched for executables when you type a command in the command prompt. By adding the path to the Python executable, you will be able to access **python.exe** by typing the **python** keyword (you won't need to specify the full path to the program).

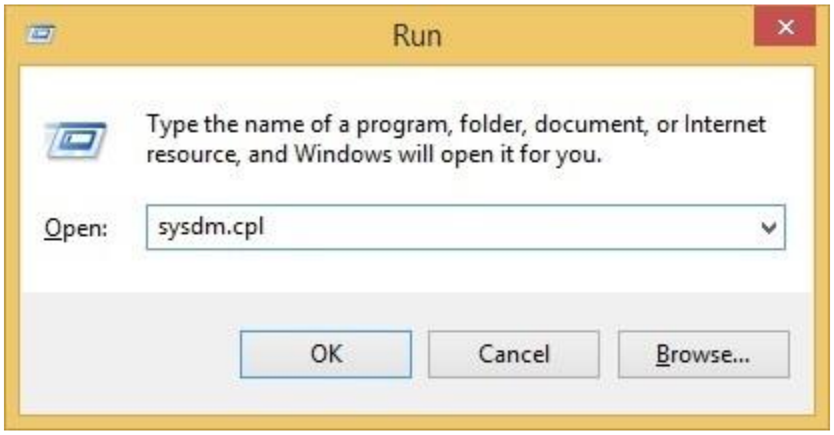
Consider what happens if we enter the **python** command in the command prompt and the path to that executable is not added to the Path variable:

```
C:\>python

'python' is not recognized as an internal or external command,

operable program or batch file.
```

As you can see from the output above, the command was not found. To run **python.exe**, you need to specify the full path to the executable:

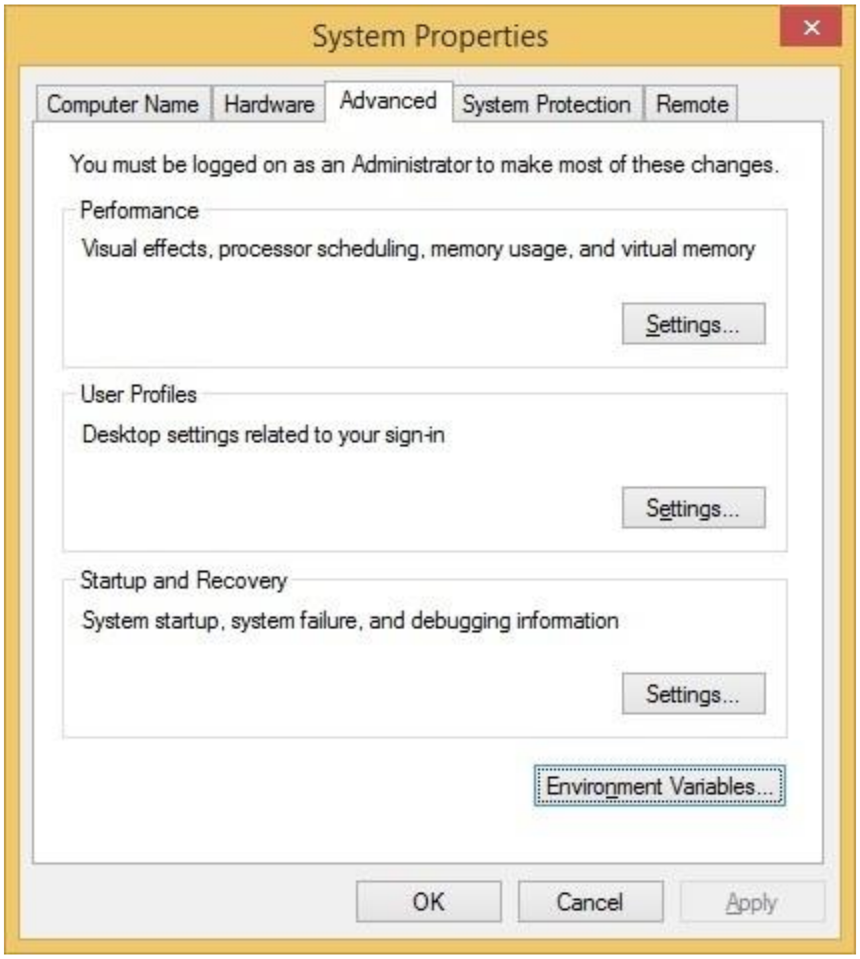
```
C:\>C:\Python34\python --version

Python 3.4.3
```

To add the path to the **python.exe** file to the Path variable, start the **Run** box and enter **sysdm.cpl**:

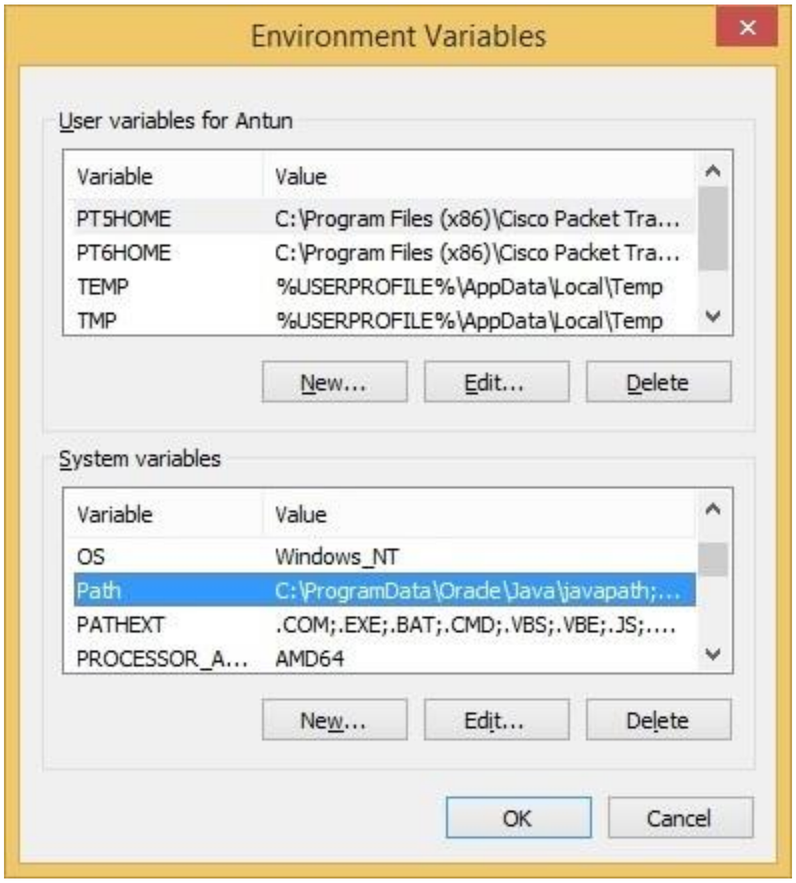

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
OK
Cancel
Browse...

sysdm.cpl

This should open up the **System Properties** window. Go to the **Advanced** tab and click the **Environment Variables** button:


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

In the **System variable** window, find the **Path** variable and click **Edit**:



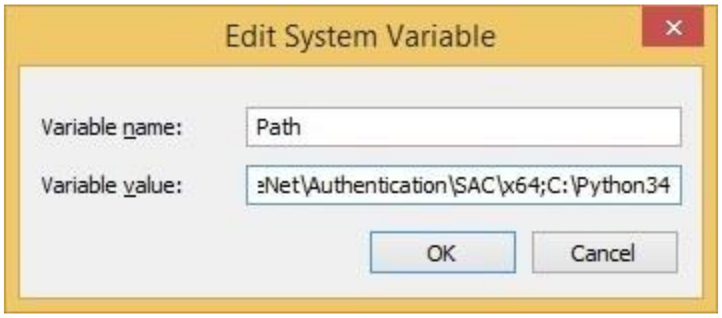
User variables for Antun

Variable	Value
PT5HOME	C:\Program Files (x86)\Cisco Packet Tra...
PT6HOME	C:\Program Files (x86)\Cisco Packet Tra...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

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

Position your cursor at the end of the **Variable value** line and add the path to the **python.exe** file, preceeded with the semicolon character (**;**). In our example, we have added the following value: **;C:\Python34**



Variable name: Path

Variable value: :Net\Authentication\SAC\x64;C:\Python34

Close all windows. Now you can run **python.exe** without specifying the full path to the file:

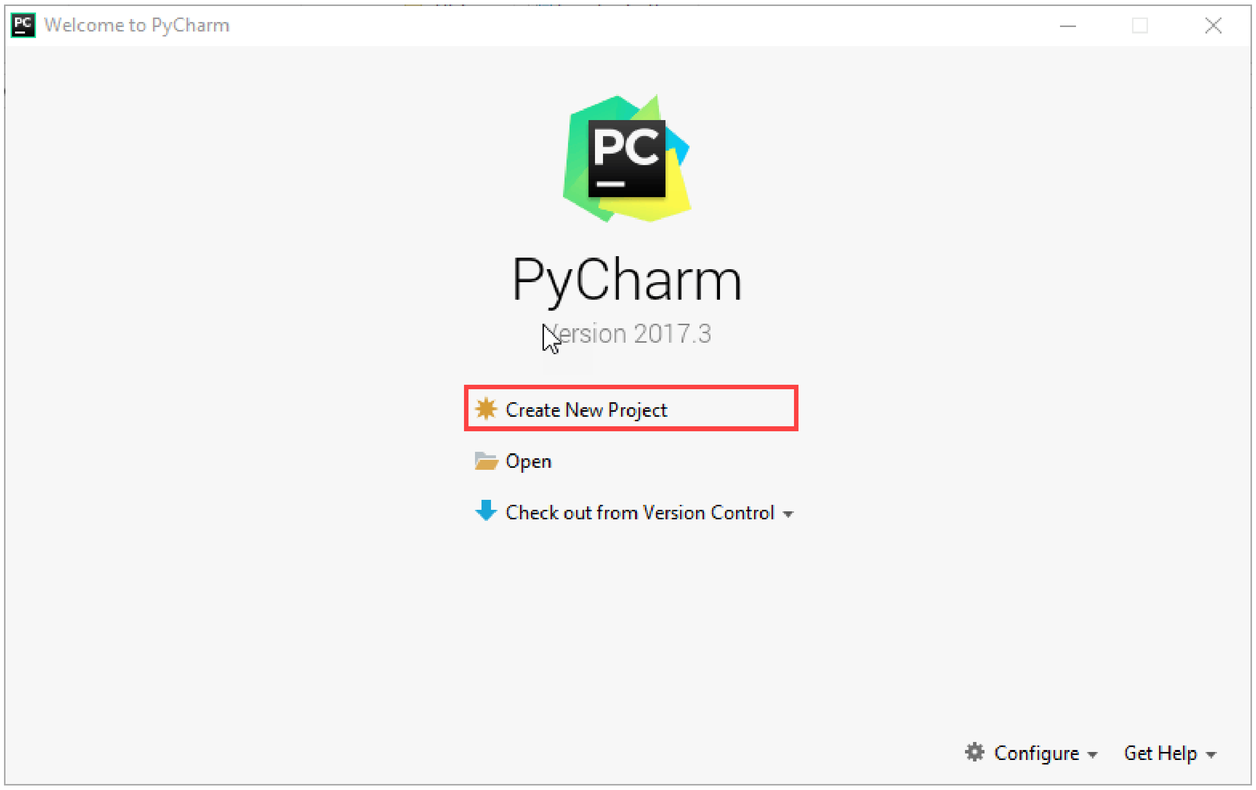
C:>python --version

Python 3.4.3

<u>Hello World: Create your First Python Program</u>

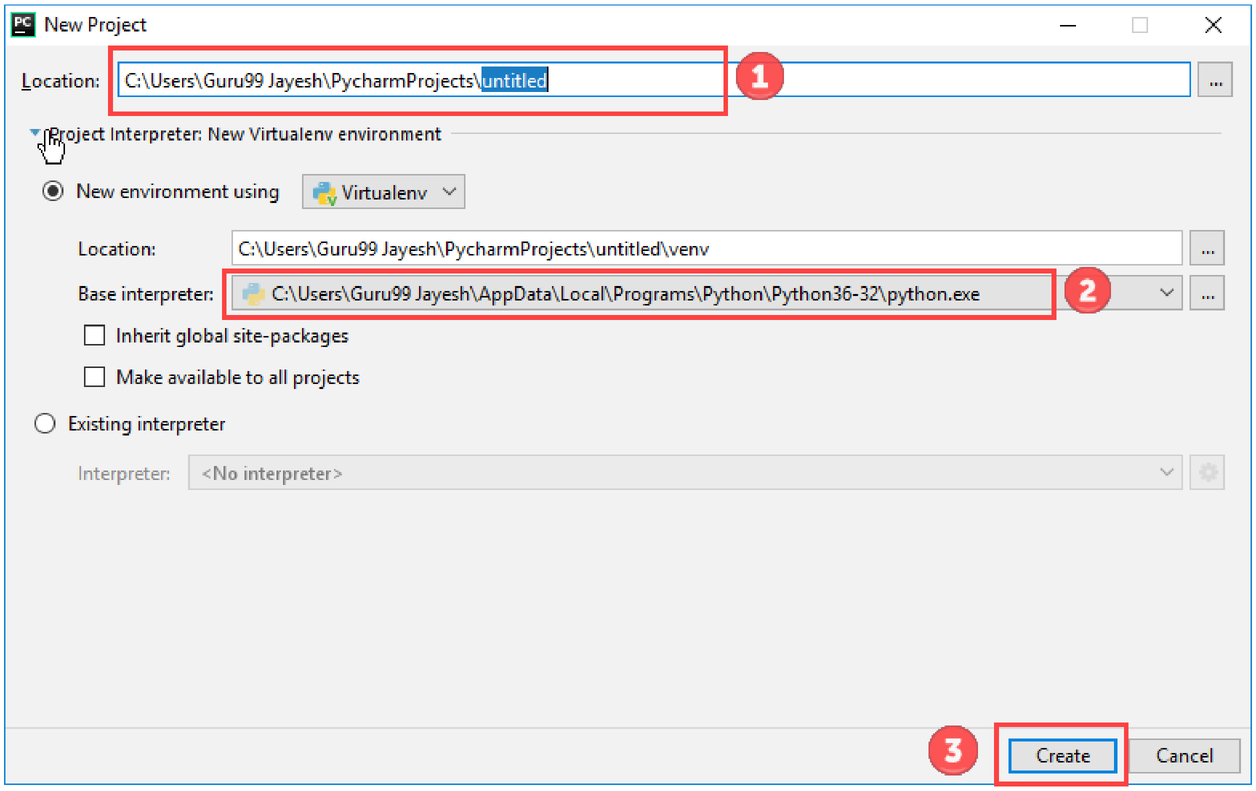
<u>Creating First Program</u>

<u>**Step 1)** Open PyCharm Editor. You can see the introductory screen for PyCharm. To create a new project, click on "Create New Project".</u>



Step 2) You will need to select a location.

1. You can select the location where you want the project to be created. If you don’t want to change location than keep it as it is but at least change the name from “untitled” to something more meaningful, like “FirstProject”.
2. PyCharm should have found the Python interpreter you installed earlier.
3. Next Click the “Create” Button.



Location: C:\Users\Guru99 Jayesh\PycharmProjects\untitled

Project Interpreter: New Virtualenv environment

New environment using Virtualenv

Location: C:\Users\Guru99 Jayesh\PycharmProjects\untitled\venv

Base interpreter: C:\Users\Guru99 Jayesh\AppData\Local\Programs\Python\Python36-32\python.exe

Inherit global site-packages

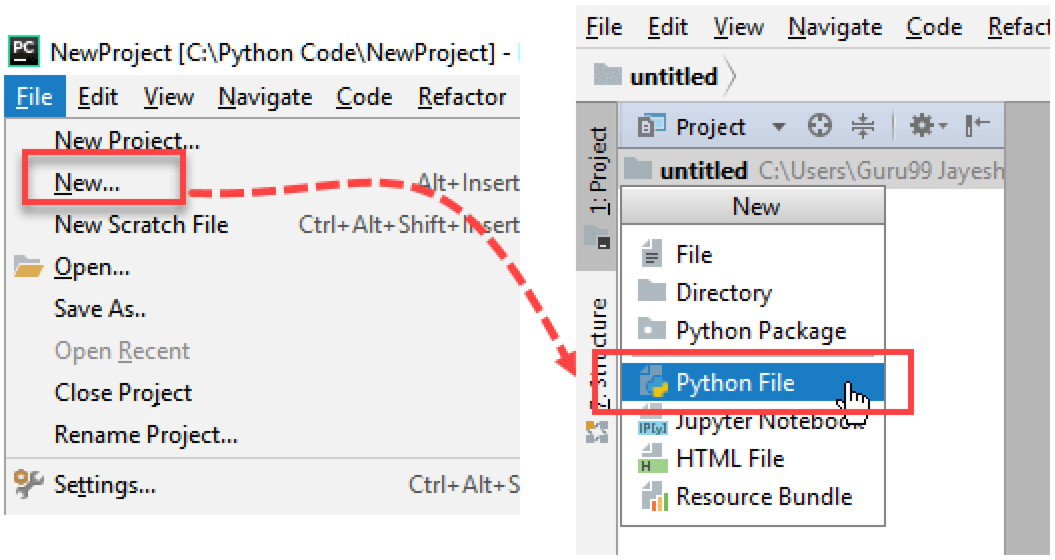
Make available to all projects

Existing interpreter

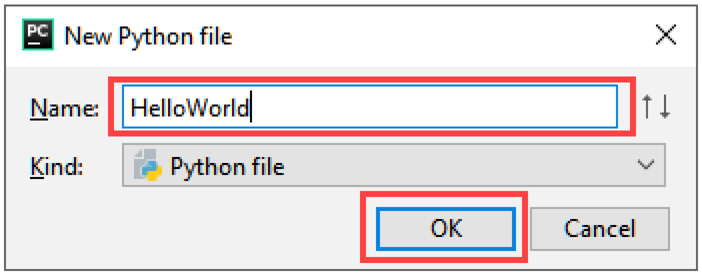
Interpreter: <No interpreter>

Step 3) Now Go up to the “File” menu and select “New”. Next, select “Python File”.

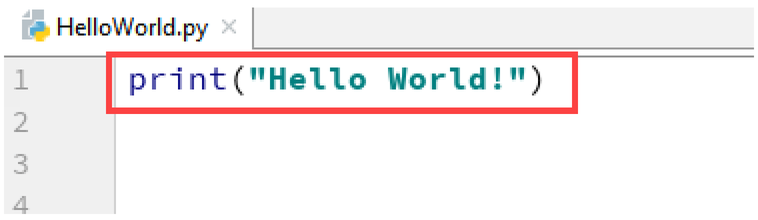
Step 3) Now Go up to the “File” menu and select “New”. Next, select “Python File”.



Step 4) A new pop up will appear. Now type the name of the file you want (Here we give "HelloWorld") and hit "OK".

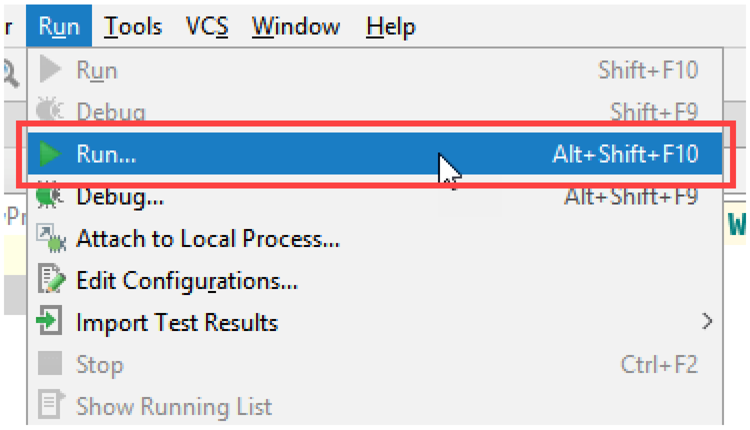


Step 5) Now type a simple program - print ('Hello World!').

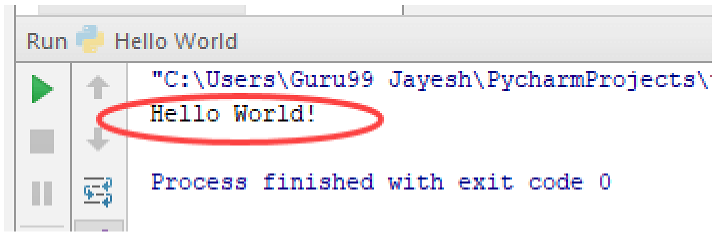


```
print("Hello World!")
```

Step 6) Now Go up to the "Run" menu and select "Run" to run your program.



Step 7) You can see the output of your program at the bottom of the screen.



```
"C:\Users\Guru99 Jayesh\PycharmProjects\
Hello World!

Process finished with exit code 0
```

Help function in Python

The python help function is used to display the documentation of modules, functions, classes, keywords etc.
The help function has the following syntax:

help([object])

If the help function is passed without an argument, then the interactive help utility starts up on the console.

Let us check the documentation of the print function in python console.

Difference between python to other languages

1. Python programs are generally expected to run slower than Java programs.
2. Python supports a programming style that uses simple functions and variables.
3. Python development is much quicker than having to write and debug a C or C++.
4. Python shines as a glue language, used to combine components written in C++
5. Python is one of the popular high-level programming languages used in an extensive variety of application domains.
6. Python provides the ability to ‘write once, run anywhere’ that enables it to run on all the operating systems which have Python installed.
7. Python has inbuilt garbage collection and dynamic memory allocation process that enables efficient memory management.
8. Python is used as a scripting language, and at times it is also used for the non-scripting purpose.
9. It is easier to write a code in Python as the number of lines is less comparatively.
10. Python is an interpreted language and it runs through an interpreter during compilation.

Python Statement, Indentation and Comments

1.Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is an assignment statement. **if statement**, **for statement**, **while statement** etc. are other kinds of statements which will be discussed later.

2.Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

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

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

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

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

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

We could also put multiple statements in a single line using semicolons, as follows

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

3.if statement
4.while statement
5for statement
6.input statement
7.print Statement ‘

Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
A code block (body of a function, loop etc.) starts with indentation and ends with the first unindented line. The amount of indentation is up to you, but it must be consistent throughout that block.

Generally four whitespaces are used for indentation and is preferred over tabs.

Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment.

It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

For Example

1. #This is a long comment
2. #and it extends
3. #to multiple lines

Python Keywords

Keywords are the reserved words in Python.

We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.

In Python, keywords are case sensitive.

There are 33 keywords in Python 3.7. This number can vary slightly in the course of time.

All the keywords except True, False and None are in lowercase and they must be written as it is. The list of all the keywords is given below.

Keywords in Python

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	

Python Identifiers

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

Rules for writing identifiers

1. Identifiers can be a combination of letters in lowercase **(a to z)** or uppercase **(A to Z)** or digits **(0 to 9)** or an underscore _. Names like myClass, var_1 and print_this_to_screen, all are valid example.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.

Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

1. number = 10

Assigning a value to a Variable in Python

As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

Example 1: Declaring and assigning a value to a variable

website = "String"

print(website)

Example 3: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, "Hello"


print (a)

print (b)

print (c)
```

Constants

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.types (int,float,double,char,strings)

Example 3: Declaring and assigning value to a constant

Create a constant.py

1. PI = 3.14
2. GRAVITY = 9.8

Create a main.py

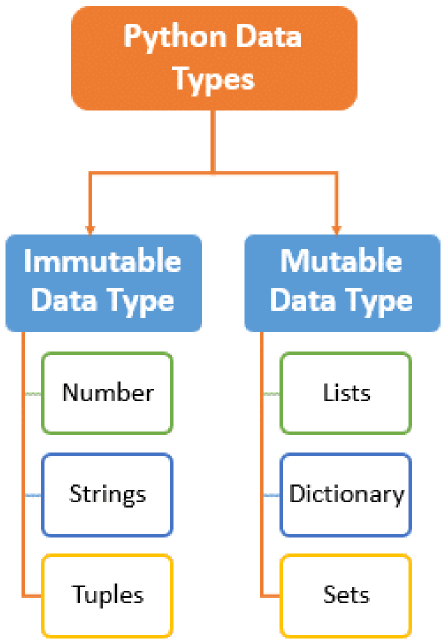
1. import constant

2. print(constant.PI)
3. print(constant.GRAVITY)

When you run the program, the output will be:

```
3.14

9.8
```

Data Type in Python



1. Number Data Type in Python

Python supports integers, floating point numbers and complex numbers. They are defined as int, float and complex class in Python.

Integers and floating points are separated by the presence or absence of a decimal point. 5 is integer whereas 5.0 is a floating point number.

```
a = 5

print(a)

# Output: 5
```

2. Python 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.).

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

output

```
[1,2,3]
```

3.Python 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.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

```
# Tuple having integers

my_tuple = (1, 2, 3,4)

print(my_tuple)

 # Output:

 (1, 2, 3,4)
```

4.Python Strings

A string is a sequence of characters. A character is simply a symbol. 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.

```
# all of the following are equivalent

my_string = 'Hello'

print(my_string)




# triple quotes string can extend multiple lines

my_string = " " "Hello, welcome to
```

```
        the world of Python" " "
print(my_string)
```

5.Python Sets

A set is an **unordered collection of items**. Every element is **unique (no duplicates)** and must be immutable (which cannot be changed).

However, the set itself is mutable. We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
```

6.Python 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.

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**.

```
  my_dict = {1: 'apple', 2: 'ball'}
```

<u>Type Conversion in Python</u>

Python defines type conversion functions to directly **convert one data type to another** which is useful in day to day and competitive programming.

1. int(a) : This function converts **any data type to integer**. 'Base' specifies the **base in which string is** if data type is string.

2. float() : This function is used to convert **any data type to a floating point number**

.

```
# Python code to demonstrate Type conversion

# using int(), float()



# initializing string

s = "10010"



# printing string converting to int

s = "10010"



c = int(s)

print (c)



# printing string converting to float

e = float(s)

print (e)
```

Output:

```
After converting to integer base  10010
```

After converting to float : 10010.0

Python Input and Output Functions

Python provides numerous built-in functions that are readily available to us at the Python prompt.Some of the functions like **input()** and **print()** are widely used for standard input and output operations respectively.

Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

We can also output data to a file, but this will be discussed later. An example use is given below.

```
print('This sentence is output to the screen')

# Output: This sentence is output to the screen


a = 5


print('The value of a is', a)

# Output: The value of a is 5
```

Python Input

Up till now, our programs were static. The value of variables were defined or hard coded into the source code.

To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

```
input([prompt])
```

where prompt is the string we wish to display on the screen. It is optional.

```
#find sum of two number using input function

a=int(input("enter first number"))

b=int(input("enter second number"))

c=a+b

print(c)
```

Python Import

A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.

Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this.

For example, we can import the math module by typing in import math.

```
import  math

r=int(input("enter the radius"))


area=(math.pi)*r*r;

print(area)output:

3.141592653589793
```

Operators in Python

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators

- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	**Name**	**Example**
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
Or	Returns True if one of the statements is true	x < 5 or x < 4
Not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns true if both variables are the same object	x is y (same value)
is not	Returns true if both variables are not the same object	x is not y (not same value)

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

<u>Python Expressions:</u>

Expressions are representations of value. They are different from statement in the fact that statements do something while expressions are representation of value. For example any string is also an expressions since it represents the value of the string as well. X+y,x-y,x*y

- A=c+b
- If(a>b):
- While(a<=10):

Python has some advanced constructs through which you can represent values and hence these constructs are also called expressions.

Following are a few types of python expressions:

1. List comprehension

The syntax for list comprehension is shown below:

```
[ compute(var) for var in iterable ]
```

For example, the following code will get all the number within 10 and put them in a list.

```
>>> [x for x in range(10)]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.Dictionary comprehension

This is the same as list comprehension but will use curly braces:

```
{ k, v for k in iterable }
```

For example, the following code will get all the numbers within 5 as the keys and will keep the corresponding squares of those numbers as the values.

```
>>> {x:x**2 for x in range(5)}

{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

3.Generator expression

The syntax for generator expression is shown below:

```
( compute(var) for var in iterable )
```

For example, the following code will initialize a generator object that returns the values within 10 when the object is called.

```
>>> (x for x in range(10))

<generator object <genexpr> at 0x7fec47aee870>

>>> list(x for x in range(10))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

4.Conditional Expressions

You can use the following construct for one-liner conditions:

true_value if Condition else false_value

Example:

```
>>> x = "1" if True else "2"

>>> x

'1'
```

Python Operator Precedence

Python has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence Order

When two operators share an operand, the operator with the higher *precedence* goes first. For example, since multiplication has a higher precedence than addition, a + b * c is treated as a + (b * c), and a * b + c is treated as (a * b) + c.(BODMAS)

Associativity

When two operators share an operand and the operators have the same precedence, then the expression is evaluated according to the *associativity* of the operators. For example, since the ** operator has right-to-left associativity, a * b * c is treated as a * (b * c). On the other hand, since the / operator has left-to-right associativity, a / b / c is treated as (a / b) / c.

Precedence and Associativity of Python Operators

The Python documentation on operator precedence contains a table that shows all Python operators from lowest to highest precedence, and notes their associativity. Most programmers do not memorize them all, and those that do still use parentheses for clarity.

Non associative operators(<,>,==,!=)

Some operators like assignment operators and comparison operators do not have associativity in Python. There are separate rules for sequences of this kind of operator and cannot be expressed as associativity.

For example, x < y < z neither means (x < y) < z nor x < (y < z). x < y < z is equivalent to x < y and y < z, and is evaluates from left-to-right.

Unit-2

Control Structures

Types

1.Decision Making Statements

- If statements
- If-else statements
- elif statements
- Nested if and if ladder statements
- elif ladder

2.Iteration Statements

- While loop
- For loop

3.break,Continue Statements

1.Decision Making Statements

Conditional statements are also known as decision-making statements. We use these statements when we want to execute a block of code when the given condition is true or false.

#1) If statements

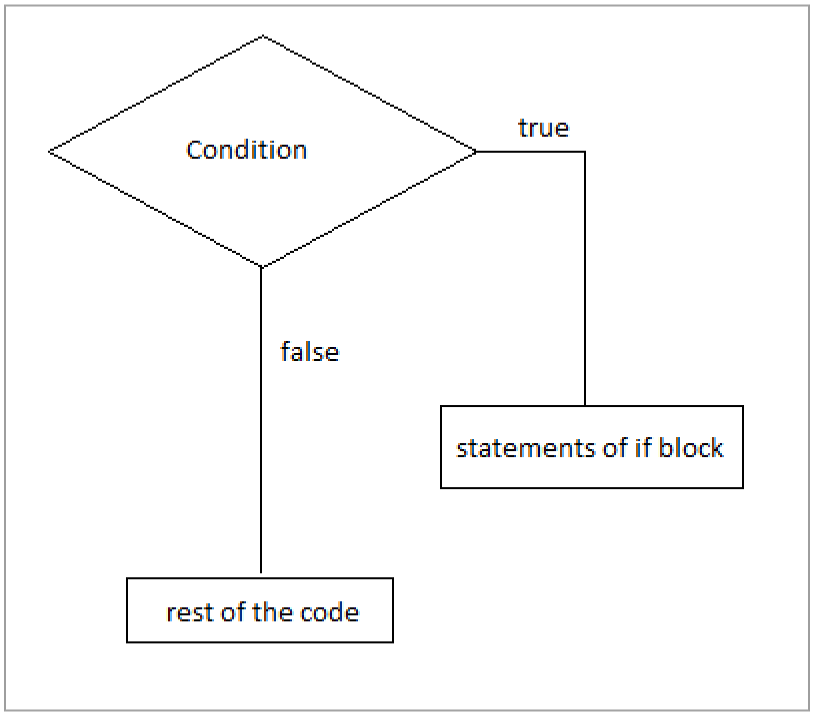
If statement is one of the most commonly used conditional statement in most of the programming languages. It decides whether certain statements need to be executed or not. If statement checks for a given condition, if the condition is true, then the set of code present inside the if block will be executed.

The If condition evaluates a Boolean expression and executes the block of code only when the Boolean expression becomes TRUE.

Syntax:

If (Boolean expression): Block of code

flow chart



If you observe the above flow-chart, first the controller will come to an if condition and evaluate the condition if it is true, then the statements will be executed, otherwise the code present outside the block will be executed.

Let’s see some examples on if statements.

<u>Example: 1</u>

```
Num = 5
If(Num < 10):
    print(“Num is smaller than 10”)

print(“This statements will always be executed”)
```

Output: Num is smaller than 10.

2.if else

The statement itself tells that if a given condition is true then execute the statements present inside if block and if the condition is false then execute the else block.

Else block will execute only when the condition becomes false, this is the block where you will perform some actions when the condition is not true.

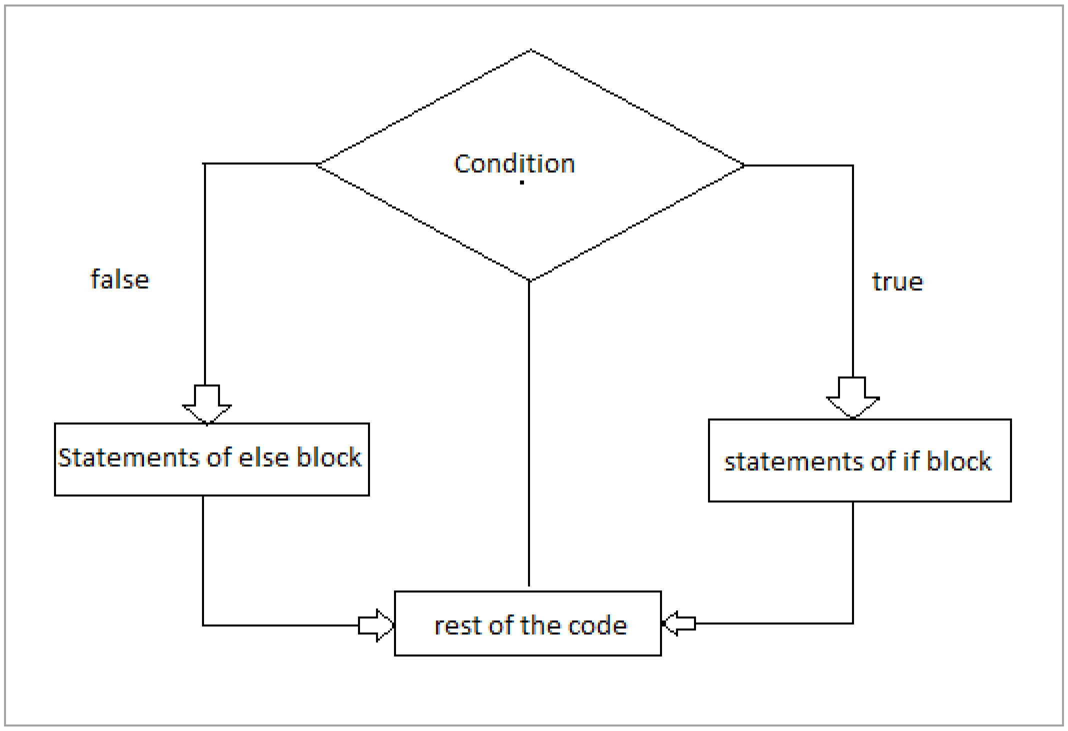
If-else statement evaluates the Boolean expression and executes the block of code present inside the if block if the condition becomes TRUE and executes a block of code present in the else block if the condition becomes FALSE.

Syntax:

```
if(Boolean expression):

Block of code #Set of statements to execute if condition is true


else:

Block of code #Set of statements to execute if condition is false
```

Here, the condition will be evaluated to a Boolean expression (true or false). If the condition is true then the statements or program present inside the if block will be executed and if the condition is false then the statements or program present inside else block will be executed.

flowchart of if-else



If you observe the above flow chart, first the controller will come to if condition and evaluate the condition if it is true and then the statements of if block will be executed otherwise else block will be executed and later the rest of the code present outside if-else block will be executed.

Example: 1

```
num = 5
if(num > 10):
    print("number is greater than 10")
else:
    print("number is less than 10")

```

7 print(“This statement will always be executed”)

Output:

number is less than 10.

#3) elif statements

In python, we have one more conditional statement called elif statements. Elif statement is used to check multiple conditions only if the given if condition false. It's similar to an if-else statement and the only difference is that in else we will not check the condition but in elif we will do check the condition.

Elif statements are similar to if-else statements but elif statements evaluate multiple conditions.

Syntax:

```
if (condition):
            #Set of statement to execute if condition is true
elif (condition):
            #Set of statements to be executed when if condition is false and elif
condition is true
else:
      #Set of statement to be executed when both if and elif conditions are false
```

<u>Example: 1</u>

```
a=int(input("enter the number to find +ve or -ve or whole number"))
if(a>0):
    print("number is +ve")
```

```
elif(a==0):
__print("number is zero/whole number")
else:
    print("number is -ve")
```

#4) Nested if/ladder statements

Nested if-else statements mean that an if statement or if-else statement is present inside another if or if-else block. Python provides this feature as well, this in turn will help us to check multiple conditions in a given program.

An if statement present inside another if statement which is present inside another if statements and so on.

Nested if Syntax:

```
if(condition):
          #Statements to execute if condition is true
          if(condition):
                  #Statements to execute if condition is true
          #end of nested if
#end of if
```

The above syntax clearly says that the if block will contain another if block in it and so on. If block can contain ‘n' number of if block inside it.

example

```
# print days of week by choice from 1 to 7
a=(int (input("enter keys from 1 to 7")))
```

```
if(a==1):
 print("today is sunday")
if(a==2):
 print("today is monday")
if(a==3):
 print("today is tuesday")
if(a==4):
 print("today is wednesday")
if(a==5):
 print("today is thursday")
if(a==6):
 print("today is friday")
if(a==7):
 print("today is saturday")
```

#5) elif Ladder

We have seen about the elif statements but what is this elif ladder. As the name itself suggests a program which contains ladder of elif statements or elif statements which are structured in the form of a ladder.

This statement is used to test multiple expressions.

Syntax:

```
if (condition):
        #Set of statement to execute if condition is true
```

```
elif (condition):

            #Set of statements to be executed when if condition is false and elif condition is true

elif (condition):

            #Set of statements to be executed when both if and first elif condition is false and second elif condition is true

elif (condition):

             #Set of statements to be executed when if, first elif and second elif conditions are false and third elif statement is true

else:

      #Set of statement to be executed when all if and elif conditions are false
```

<u>Example: 1</u>

example

```
# print days of week by choice from 1 to 7

a=(int (input("enter keys from 1 to 7")))

if(a==1):

 print("today is sunday")

elif(a==2):

 print("today is monday")

elif(a==3):

 print("today is tuesday")

elif(a==4):

 print("today is wednesday")

elif(a==5):

 print("today is thursday")
```

```
elif(a==6):
 print("today is friday")
elif(a==7):
 print("today is saturday")
```

Looping Statements in Python

Looping statements in python are used to execute a block of statements or code repeatedly for several times as specified by the user.

Python provides us with 2 types of loops as stated below:

- While loop
- For loop

#1) While loop:

While loop in python is used to execute multiple statement or codes repeatedly until the given condition is true.

We use while loop when we don’t know the number of times to iterate.

3 parts of loop

1.intialization (Starting point)

2.condition (ending point)

3.increment /decrement

Syntax:

while (expression): block of statements Increment or decrement operator

In while loop, we check the expression, if the expression becomes true, only then the block of statements present inside the while loop will be executed. For every iteration, it will check the condition and execute the block of statements until the condition becomes false.

```
i = 0



while (i<=10):

      print(i)

      i = i+1



print(“end loop)
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

#2) For loop:

For loop in python is used to execute a block of statements or code several times until the given condition becomes false.

We use for loop when we know the number of times to iterate.

Syntax:

for var in sequence: Block of code

Here var will take the value from the sequence and execute it until all the values in the sequence are done.

```
language = [‘Python’, ‘Java’, ‘Ruby’]
```

```
for lang in language:
    print(“Current language is: “, lang)
```

Output:

```
Current language is: Python
Current language is: Java
Current language is: Ruby
```

Using range function

Example

```
for i in range(1,11):
Print(i)
```

output

```
1 2 3 4 5 6 7 8 9 10
```

Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to **abort the current execution of the program** and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

```
#loop statements
break;
```

```
   example
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
    if i==5:
        break;
    print(i);
```

output

```
1 2 3 4
```

Python continue Statement

The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

The syntax of Python continue statement is given below.

```
#loop statements
continue;
#the code to be skipped
```

Example

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
        if i==5:
            continue;
        print(i);
```

Output:

```
1
2
3
4
6
7
8
9
10
```

Python Native Data Types

1.Python 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.).

```
1. # empty list
2. my_list = []


3. # list of integers
```

4. my_list = [1, 2, 3]

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

Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

access elements from a list

```
 my_list = ['p','r','o','b','e']
 # Output: p
 print(my_list[0])
# Output: o
print(my_list[2])
# Output: e
 print(my_list[4])
```

Python List Built-in functions

Python provides the following built-in functions which can be used with the lists.

Python List built-in methods/functions

SN	Function	Description

1	list.append(obj)	The element represented by the object obj is added to the list.
		
		a=[1,2,3]
		
		a.append(4)
		
		print(a)

2	list.clear()	It removes all the elements from the list.
		
		a=[1,2,3]
		
		a.clear()
		
		print(a)

3	List.copy()	It returns a shallow copy of the list.
		
		a=[1,2,3]
		
		b=a.copy()
		
		print(b)

4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
		
		a=[1,2,3,4,5,2,5,6]
		
		Print(a.count(5))

5	list.extend(seq)	The sequence represented by the object seq is extended to the list.

		List1=[1,2,3]
		
		List2=[4,5,6]
		
		List1.extend(List2)
		
		Print(List1)

6	list.index(obj)	It returns the index value in the list that object appears.
		
		l=[1,2,3,4,5]
		
		print(l.index(5))

7	list.insert(index, obj)	The object is inserted into the list at the specified index.
		
		L=[1,2,4,5]
		
		L.insert(2,3)
		
		Print(L)

8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
		
		S=[1,2,3,4,5]
		
		int(S.pop())
		
		print(S)

9	list.remove(obj)	It removes the specified object from the list.
		
		L=[1,2,1,1,3]
		
		L.remove(1)
		
		Print(L)

10	list.reverse()	It reverses the list.

		List=[1,2,3,4,5]
		
		List.reverse()
		
		Print(List)

3.Python 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.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

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)
```

A tuple can also be created without using parentheses. This is known as tuple packing.for example

```
my_tuple = 3, 4.6, "dog"

print(my_tuple)   # Output: 3, 4.6, "dog"
```

Python Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false. tuple1, tuple2 = (123, 'xyz'), (456, 'abc')

		print cmp(tuple1, tuple2)
2	len(tuple)	It calculates the length of the tuple. tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc') print ("First tuple length : ", len(tuple1)) print ("Second tuple length : ", len(tuple2))
3	max(tuple)	It returns the maximum element of the tuple. tuple1, tuple2 = ('maths', 'che', 'phy', 'bio'), (456, 700, 200) print ("Max value element : ", max(tuple1)) print ("Max value element : ", max(tuple2))
4	min(tuple)	It returns the minimum element of the tuple. tuple1, tuple2 = ('maths', 'che', 'phy', 'bio'), (456, 700, 200) print ("Max value element : ", min(tuple1)) print ("Max value element : ", min(tuple2))

5	tuple(seq)	It converts the specified sequence to the tuple.
		list1= [1, 2, 3, 4]
		tuple2 = tuple(list1)

		print(tuple2)

Basic Tuple operations

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example

Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1 = (1, 2, 3, 4, 5,)
		T1=T1*2
		Print(T1)

Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1= (1, 2, 3, 4, 5, 6, 7, 8, 9)
		T1=T1+(10,)
		Print(T1)

Membership	It returns true if a particular item exists in the tuple otherwise false.	T1=(1,2,3,4,5) print (2 in T1)

Iteration	The for loop is used to iterate over the tuple elements.	T1=(1,2,3)
		for i in T1:
		    print(i)

		Output 1 2 3 4 5
Length	It is used to get the length of the tuple.	T1=(1,2,3,4,5) len(T1) = 5

List VS Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the variable length.	The tuple has the fixed length.
4	The list provides more functionality than tuple.	The tuple provides less functionality than the list.
5	The list Is used in the scenario in which we need to store the simple	The tuple is used in the cases where we need to store the read-only collections

	collections with no constraints where the value of the items can be changed.	i.e., the value of the items can not be changed. It can be used as the key inside the dictionary.
6	Syntax	
7.	Example	

Python Sets

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

However, the set itself is mutable. We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

Example 1: using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output:

```
looping through the set elements ...
Friday
Tuesday
Monday
Saturday
Thursday
Sunday
Wednesday
```

Example 2: using set() method

```python
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output:

```
looping through the set elements ...
Friday
```

```
Tuesday

Monday

Saturday

Thursday

Sunday

Wednesday
```

Python Set operations

In the previous example, we have discussed about how the set is created in python. However, we can perform various mathematical operations on python sets like union, intersection, difference, etc.

Union of two Sets

The union of two sets are calculated by using the or (|) operator. The union of the two sets contains the all the items that are present in both the sets.

Consider the following example to calculate the union of two sets.

Example 1 : using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Python also provides the **union()** method which can also be used to calculate the union of two sets. Consider the following example.

Example 2: using union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
```

2. Days2 = {"Friday","Saturday","Sunday"}
3. **print**(Days1.union(Days2)) #printing the union of the sets

Output:

```
{'Friday', 'Monday', 'Tuesday', 'Thursday', 'Wednesday', 'Sunday', 'Saturday'}
```

Intersection of two sets

The & (intersection) operator is used to calculate the intersection of the two sets in python. The intersection of the two sets are given as the set of the elements that common in both sets.

Consider the following example.

Example 1: using & operator

1. set1 = {"Ayush","John", "David", "Martin"}
2. set2 = {"Steve","Milan","David", "Martin"}
3. **print**(set1&set2) #prints the intersection of the two sets

Output:

```
{'Martin', 'David'}
```

Example 2: using intersection() method

1. set1 = {"Ayush","John", "David", "Martin"}
2. set2 = {"Steave","Milan","David", "Martin"}
3. **print**(set1.intersection(set2)) #prints the intersection of the two sets

Output:

```
{'Martin', 'David'}
```

The intersection_update() method

The intersection_update() method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).

The Intersection_update() method is different from intersection() method since it modifies the original set by removing the unwanted items, on the other hand, intersection() method returns a new set.

Consider the following example.

```
1. a = {"ayush", "bob", "castle"}
2. b = {"castle", "dude", "emyway"}
3. c = {"fuson", "gaurav", "castle"}
4.
5. a.intersection_update(b, c)
6.
7. print(a)
```

Output:

```
{'castle'}
```

Python Built-in set methods

Python contains the following methods to be used with the sets.

SN	Method	Description

1	add(item)	It adds an item to the set. It has no effect if the item is already present in the set.
		
		GEEK = {'g', 'e', 'k'}
		
		# adding 's'
		
		GEEK.add('s')

		print('Letters are:', GEEK)

2	clear()	It deletes all the items from the set.
		set1 = {1,2,3,4,5,6}
		set1.clear()
		print("\nSet after using clear() function")
		print(set1)

3	copy()	It returns a shallow copy of the set.
		set1 = {1, 2, 3, 4}
		set2 = set1.copy()
		print(set2)

4	difference_update(....)	It modifies this set by removing all the items that are also present in the specified sets.
		A = {'s', 'u', 'n', 'n', 'y'}
		B = {'b', 'u', 'n', 'n', 'y'}
		result = A.symmetric_difference_update(B)
		print('A = ', A)
		print('B = ', B)
		print('result = ', result)

5	discard(item)	It removes the specified item from the set.
		
		fruits = {"apple", "banana", "cherry"}
		
		fruits.discard("banana")
		
		print(fruits)

6	intersection()	It returns a new set that contains only the **common elements** of both the sets. (all the sets if more than two are specified).
		
		x = {"apple", "banana", "cherry"}
		y = {"google", "microsoft", "apple"}
		
		z = x.intersection(y)
		
		print(z)

Python String

Till now, we have discussed numbers as the standard data types in python. In this section of the tutorial, we will discuss the most popular data type in python i.e., string.

In python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to **use single quotes, double quotes, or triple quotes to create the string.**

Consider the following example in python to create a string.

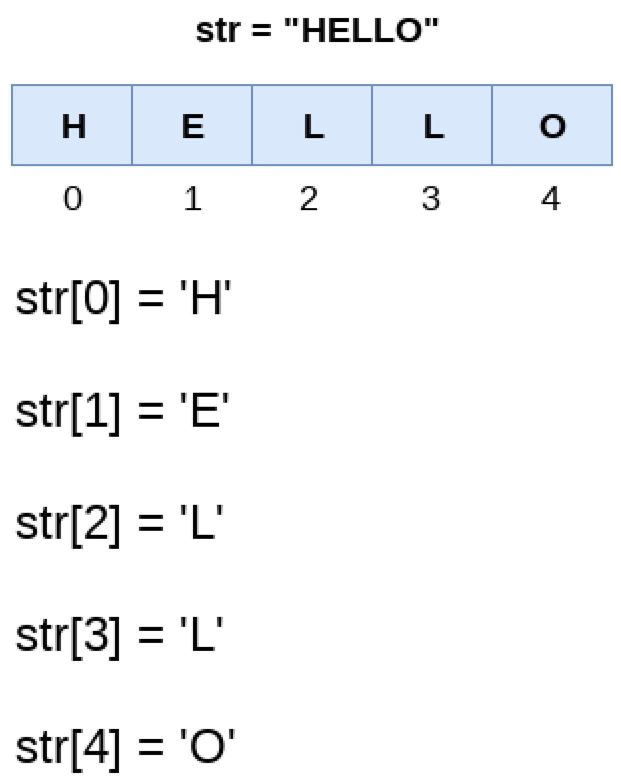
str = "Hi Python !"

print(type(str)), then it will **print** string (str).

In python, strings are treated as the sequence of strings which means that python doesn't support the character data type instead a single character written as 'p' is treated as the string of length 1.

Strings indexing and splitting

Like other languages, the indexing of the python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.



```
str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'
```

Built-in String functions

Python provides various in-built functions that are used for string handling. Many String fun

Method	**Description**
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3

	string = "python is AWesome."
	
	b = string.capitalize()
	
	print('New String: ', b)
	print('Capitalized String:', b)

casefold()	It is change string in lower case.
	
	string = "PYTHON IS AWESOME"
	
	# print lowercase string
	print("Lowercase string:", string.casefold())

center(width ,fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
	string = "Python is awesome"
	
	new_string = string.center(24)
	
	print("Centered String: ", new_string)

upper()	The string upper() method converts all lowercase characters in a string into uppercase characters and returns it. string = "this should be uppercase!" print(string.upper())

split()	The split() method breaks up a string at the specified separator and returns a list of strings.
	
	text= 'Love thy neighbor'
	
	print(text.split( ))

replace()	The replace() method returns a copy of the string where all occurrences of a substring is replaced with another substring.
	
	song = 'cold, cold heart'
	
	# replacing 'cold' with 'hurt'
	
	print(song.replace('cold', 'hurt'))

Index()	The index() method **returns the index number** of given string (if found).

	sentence = 'Python'
	
	result = sentence.index('n')
	print("Substring 'is fun':", result)

endswith()	The endswith() method returns True if a string ends with the specified suffix. If not, it returns False.
	text = "Python is easy to learn."
	
	result = text.endswith('to learn.')
	# returns False
	print(result)

String Operators

Operator	**Description**
+	It is known as concatenation operator used to join the strings given either side of the operator.

	1. str = "Hello" 2. str1 = " world" 3. **print**(str+str1) # prints Hello world
*	It is known as repetition operator. It concatenates the multiple copies of the same string. 1. **print**(str*3) # prints HelloHelloHello
[ ]	It is known as slice operator. It is used to access the sub-strings of a particular string. 1. **print**(str[4]) # prints o
[:]	It is known as range slice operator. It is used to access the characters from the specified range. 1. **print**(str[2:4]); # prints ll
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string. 1. **print**('w' **in** str) # prints false as w is not present in str

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.

An item has a key and the corresponding value expressed as a **pair, key: value**.

```
# dictionary with integer keys

my_dict = {1: 'apple', 2: 'ball'}
```

Python has a set of built-in methods that you can use on dictionaries.

Method	Description

clear()	Removes all the elements from the dictionary
	
	car = {
	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	
	car.clear()
	
	print(car)

copy()	Returns a copy of the dictionary car = {

	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	
	x = car.copy()
	
	print(x)

fromkeys()	Returns a dictionary with the specified keys and value,**it Create a dictionary with 3 keys, all with the value 0:**
	x = ('key1', 'key2', 'key3')
	y = 0
	thisdict = dict.fromkeys(x, y)
	print(thisdict)

get()	Returns the value of the specified key
	car = {
	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	
	x = car.get("model")
	
	print(x)

items()	Returns a list containing a tuple for each key value pair
	
	car = {
	
	"brand": "Ford",
	
	"model": "Mustang",
	
	"year": 1964
	
	}
	
	x = car.items()
	
	print(x)

keys()	Returns a list containing the dictionary's keys
	
	car = {
	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	
	x = car.keys()
	
	print(x)

pop()	Removes the element with the specified key car = { "brand": "Ford", "model": "Mustang",

	"year": 1964
	}
	
	car.pop("model")
	
	print(car)

popitem()	Removes the last inserted key-value pair,it delete last value.
	
	car = {
	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	
	car.popitem()
	
	print(car)

setdefault()	Returns the value of the specified key value. If the key does not exist: insert the key, with the specified value car = { "brand": "Ford", "model": "Mustang", "year": 1964

	}
	
	x = car.setdefault("model", "Bronco")
	
	print(x)

update()	Updates the dictionary with the specified key-value pairs
	car = {
	"brand": "Ford",
	"model": "Mustang",
	"year": 1964
	}
	car.update({"color": "White"})
	car.update({“age”:34})
	print(car)

values()	Returns a list of all the values in the dictionary car = { "brand": "Ford", "model": "Mustang", "year": 1964

```
}

x = car.values()

print(x)
```

Unit-3

Python Functions

Functions are the most important aspect of an application. **A function can be defined as the organized block of reusable code which can be called whenever required.**

Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by **{}**. A function can be called multiple times to **provide reusability** and **modularity** to the python program.

Types of functions in python

1.Inbuilt functions

Python provide us various inbuilt functions like range() or print(),input().

2.User defined functions

The user can create its functions which can be called user-defined functions.

Types of user defined functions in python.

1. A function without parameter
2. A function with parameter
3. A function with return type

Advantage of functions in python

There are the following advantages of C functions.

- By using functions, **we can avoid rewriting** same logic/code again and again in a program.
- We can call python functions any number of times in a program and from any place in a program.
- We can track a large python program easily when it is divided into multiple functions.

- Reusability is the main achievement of python functions.
- Improving clarity of the code
- Information hiding
- Reducing duplication of code

1.A function without parameter

Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

1. **def** my_function():
2.   function-suite
3.   <expression>

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.

A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

A simple function that prints the message "Hello Word" is given below.

1. **def** hello_world():

```
    print("hello world")

hello_world()
```

Output:

```
hello world
```

2.A function with Parameter

The information into the functions can be passed as the parameters. The parameters are specified in the parentheses. We can give any number of parameters, but we have to separate them with a comma.

Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

```
def my_function(parameterlist):
    function-suite
    <expression>
```

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.

A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

Consider the following example which contains a function that accepts a string as the parameter and prints it.

Example

```
#python function to calculate the sum of two variables
#defining the function
def sum (a,b):
    c=a+b;
  Print(“sum is”,c)
#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))
sum(a,b)
```

Output:

```
Enter a: 10

Enter b: 20

Sum =  30
```

3.A function with return type

Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

```
def my_function():
```

2. function-suite
3. Return <expression>

The function block is started with the colon (:) and all the same level block statements remain at the same indentation.

A function can accept any number of parameters that must be the same in the definition and function calling.

Function calling

In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.

Note: Return statement can not be used outside the function.

#program add two number

```
def f(x, y):

    z = (x + y)

    return z



a = 4

b = 7
```

```
res2 = f(a, b)
print("Result of function call:", res2)
```

Call by value in Python

In the event that you pass arguments like whole numbers, strings or tuples to a function, the passing is like call-by-value because you can not change the value of the immutable objects being passed to the function.

```
# Python code to demonstrate
# call by value



string = "hello"



def test(string):

    string = "world"
    print("Inside Function:", string)



test(string)
print("Outside Function:", string)
```

Output

```
Inside Function: world
```

```
Outside Function: hello
```

Call by reference in Python

In python, all the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.

However, there is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.

Example 1 Passing Immutable Object (List)

```
list1=[1,2,3,4,5]


def fun(list1):

    list1.append(20)

    print("inside the list",list1)




fun(list1)

print("outside",list1)
```

Output:

```
('inside the list', [1, 2, 3, 4, 5, 20])

('outside', [1, 2, 3, 4, 5, 20])
```

Scope of variables

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables: these are declare outside of the block
2. Local variables: these are declare inside of the block

Parameter	Local	Global
Scope	It is declared inside a function.	It is declared outside the function.
Value	If it is not initialized, a garbage value is stored	If it is not initialized zero is stored as default.
Lifetime	It is created when the function starts execution and lost when the functions terminate.	It is created before the program's global execution starts and lost when the program terminates.
Data sharing	Data sharing is not possible as data of the local variable can be accessed by only one function.	Data sharing is possible as multiple functions can access the same global variable.
Parameters	Parameters passing is required for local variables to access the value in other function	Parameters passing is not necessary for a global variable as it is visible throughout the program
Modification of variable value	When the value of the local variable is modified in one function, the changes are not visible in another function.	When the value of the global variable is modified in one function changes are visible in the rest of the program.
Accessed by	Local variables can be accessed with the help of statements,	You can access global variables by any statement in the program.

Parameter	Local	Global
	inside a function in which they are declared.	
Memory storage	It is stored on the stack unless specified.	It is stored on a fixed location decided by the compiler.

Python Recursive Function

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

Following is an example of recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720.

```
# find the factorial of a number


def calc_factorial(x):


    if x == 1:
        return 1
    else:
        return (x * calc_factorial(x-1))


num = 4
print("The factorial of", num, "is", calc_factorial(num))
```

Output

The factorial of 4 is 24

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Python Modules

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

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

To use the functionality of one module into another, we must have to import the specific module.

Example

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

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

```
#displayMsg prints a message to the name being passed.
def displayMsg(name)
    print("Hi "+name);
```

Here, we need to include this module into our main module to call the method displayMsg() defined in the module named file.

Loading the module in our python code

We need to load the module in our python code to use its functionality. Python provides two types of statements as defined below.

1. The import statement
2. The from-import statement

The import statement

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

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

The syntax to use the import statement is given below.

1. **import** module1,module2,........ module n

Hence, if we need to call the function displayMsg() defined in the file file.py, we have to import that file as a module into our module as shown in the example below.

Example:

```
import file;

name = input("Enter the name?")

file.displayMsg(name)
```

Output:

```
Enter the name?John

Hi John
```

The from-import statement

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

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

Consider the following module named as calculation which contains three functions as summation, multiplication, and divide.

calculation.py:

1. #place the code in the calculation.py
2. **def** summation(a,b):

```
    return a+b
def multiplication(a,b):
    return a*b;
def divide(a,b):
    return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))
```

Output:

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

The from...import statement is always better to use if we know the attributes to be imported from the module in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using *.

Consider the following syntax.

```
from <module> import *
```

Renaming a module

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.

The syntax to rename a module is given below.

import <module-name> as <specific-name>

Example

```
#the module calculation of previous example is imported in this example as cal.
 import calculation as cal;

a = int(input("Enter a?"));

b = int(input("Enter b?"));

print("Sum = ",cal.summation(a,b))
```

Output:

```
Enter a?10

Enter b?20

Sum =  30
```

Using dir() function

The dir() function returns a sorted list of names defined in the passed module. This list contains all the sub-modules, variables and functions defined in this module.

Consider the following example.

Example

```
1. import json
2.
3. List = dir(json)
4.
5. print(List)
```

Output:

```
['JSONDecoder', 'JSONEncoder', '__all__', '__author__', '__builtins__',
'__cached__', '__doc__',
```

'__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__',

'_default_decoder', '_default_encoder', 'decoder', 'dump', 'dumps', 'encoder', 'load', 'loads', 'scanner']

The reload() function

As we have already stated that, a module is loaded once regardless of the number of times it is imported into the python source file. However, if you want to reload the already imported module to re-execute the top-level code, python provides us the reload() function. The syntax to use the reload() function is given below.

1. reload(<module-name>)

for example, to reload the module calculation defined in the previous example, we must use the following line of code.

1. reload(calculation)

Standard Modules in Python

Statistics Module

This module, as mentioned in the Python 3 documentation, provides functions for calculating mathematical statistics of numeric (Real-valued) data.

Math Module

This module, as mentioned in the Python 3's documentation, provides access to the mathematical functions defined by the C standard.

Random module

This module, as mentioned in the Python 3's documentation, implements pseudo-random number generators for various distributions.

Create and Access a Python Package

Packages are a way of structuring many packages and modules which helps in a well-organized hierarchy of data set, making the directories and modules easy to access.

To create a package in Python, we need to follow these three simple steps:

1. First, we create a directory and give it a package name, preferably related to its operation.
2. Then we put the classes and the required functions in it.
3. Finally we create an __init__.py file inside the directory, to let Python know that the directory is a package.

Example of Creating Package

Let's look at this example and see how a package is created. Let's create a package named Cars and build three modules in it namely, Bmw, Audi and Nissan.

1. **First we create a directory and name it Cars.**
2. **Then we need to create modules**. To do this we need to create a file with the name Bmw.py and create its content by putting this code into it.

```
# Python code to illustrate the Modules

class Bmw:
        # First we create a constructor for this class
        # and add members to it, here models
        def __init__(self):
            self.models = ['i8', 'x1', 'x5', 'x6']

        # A normal print function
        def outModels(self):
            print('These are the available models for BMW')
            for model in self.models:
                print('\t%s ' % model)
```

Then we create another file with the name Audi.py and add the similar type of code to it with different members.

```
def add(x,y):
     z=x*y
  return(z)
```

3. **Finally we create the __init__.py file.** This file will be placed inside Cars directory and can be left blank or we can put this initialisation code into it.

```
from cars import b
x=int(input("enter first number"))
y=int(input("enter second number"))
print(b.add(x,y))
```

Now, let’s use the package that we created. To do this make a sample.py file in the same directory where Cars package is located and add the following code to it:

```
# Import classes from your brand new package
from Cars import Bmw
from Cars import Audi

# Create an object of Bmw class & call its method
ModBMW = Bmw()
ModBMW.outModels()

# Create an object of Audi class & call its method
ModAudi = Audi()
ModAudi.outModels()
```

Unit-4

Exception Handling

An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

Why use Exceptions?

Exceptions are convenient in many ways for handling errors and special conditions in a program. When you think that you have a code which can produce an error then you can use exception handling.

Types of Exception

1)Build in

2) User Define

1)Build in Exception

Below is some common exceptions errors in Python:

IOError

If the file cannot be opened.

ImportError

If python cannot find the module

ValueError

Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value

KeyboardInterrupt

Raised when the user hits the interrupt key (normally Control-C or Delete)

EOFError

Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

Syntax

```
try:
   some statements here
except:
   exception handling
```

Example user define

```
try:
    print (1/0)
except ZeroDivisionError:
    print "You can't divide by zero."
```

Output

```
  You can’t divide by zero
```

Build in

user-generated interruption is signaled by raising **the Keyboard Interrupt exception.**

```
>>> while True:
...     try:
...         x = int(input("Please enter a number: "))
...         break
...     except ValueError:
...         print("Oops!  That was no valid number.  Try again...")
```

<u>File Handling</u>

File handling in Python requires no importing of modules.

File Object

Instead we can use the built-in object "file". That object provides basic functions and methods necessary to manipulate files by default. Before you can read, append or write to a file, you will first have to it using

Use the different methods of the file object

1.Open()

The open() function is used to open files in our system, the filename is the

name of the file to be opened.

The mode indicates, how the file is going to be opened "r" for reading,"w" for writing and "a" for a appending. The open function takes two arguments, the name of the file and and the mode or which we would like to open the file. By default, when only the filename is passed, the open function opens the file in read mode.

Example

This small script, will open the (hello.txt) and print the content.

This will store the file information in the file object "filename".

```
filename = "hello.txt"

file = open(filename, "r")

for line in file:

   print line,
```

2.Read ()

The read functions contains different methods, read(),readline() and readlines()

read() #return one big string

readline #return one line at a time

readlines #returns a list of lines

3.Write ()

This method writes a sequence of strings to the file.

write () #Used to write a fixed sequence of characters to a file

writelines() #writelines can write a list of strings.

4.Append ()

The append function is used to append to the file instead of overwriting it.

To append to an existing file, simply open the file in append mode ("a"):

5.Close() When you're done with a file, use close() to close it and free up any system

resources taken up by the open file.

6.seek() sets the file's current position at the offset. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

7.tell() Python file method **tell()** returns the current position of the file read/write pointer within the file.

File Handling Examples

To open a text file, use:

fh = open("hello.txt", "r")

To read a text file, use:

```
fh = open("hello.txt","r")
print fh.read()
```

To read one line at a time, use:

```
fh = open("hello".txt", "r")
print fh.readline()
```

To read a list of lines use:

```
fh = open("hello.txt.", "r")
print fh.readlines()
```

To write to a file, use:

```
fh = open("hello.txt","w")
write("Hello World")
fh.close()
```

To write to a file, use:

```
fh = open("hello.txt", "w")
lines_of_text = ["a line of text", "another line of text", "a third line"]
fh.writelines(lines_of_text)
fh.close()
```

To append to file, use:

```
fh = open("Hello.txt", "a")

write("Hello World again")

fh.close()
```

To close a file, use

```
fh = open("hello.txt", "r")

print fh.read()

fh.close()
```

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

8.The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file test1.txt −

```
#!/usr/bin/python

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )
```

9.The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

os.remove(file_name)

Example

Following is the example to delete an existing file test2.txt −

```
#!/usr/bin/python

import os

# Delete file test2.txt

os.remove("text2.txt")
```

Listing out directories and files in Python

The following is a list of some of the important methods/functions in Python with descriptions that you should know to understand this article.

1. **len()** – It is used to count number of elements(items/characters) of iterables like list, tuple, string, dictionary etc.
2. **str()** – It is used to transform data value(integers, floats, list) into string.
3. **abspath()** – It returns the absolute path of the file/directory name passed as an argument.
4. **enumerate()** – Returns an enumerate object for the passed iterable that can be used to iterate over the items of iterable with an access to their indexes.

5. **list()** – It is used to create a list by using an existing iterable(list, tuple, dictionary, set).
6. **listdir()** – It is used to list the directory contents. The path of directory is passed as an argument.
7. **isfile()** – It checks whether the passed parameter denotes the path to a file. If yes then returns **True** otherwise **False**
8. **isdir()** – It checks whether the passed parameter denotes the path to a directory. If yes then returns **True** otherwise **False.**

Object Oriented Programming Concept in Python

Python is a multi-paradigm programming language. It supports different programming approaches.

One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

1.Class

A class is a blueprint for the object.

We can think of class as a sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, a parrot is an object.

The example for class of parrot can be :

```
class Parrot:
   pass
```

2.Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

obj = Parrot()

3.Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

4.Inheritance

Inheritance is a way of creating a new class for using details of an existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

5.Encapsulation

Using OOP in Python, we can restrict access to **methods and variables**. This prevents data from direct modification which is called encapsulation. In Python, we denote private attributes using underscore as the prefix i.e single _ or double __.

6.Polymorphism

Polymorphism is an ability (in OOP) to use a common interface for multiple forms (data types).

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

7.Data Abstraction

Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonyms because data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does.

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Example

Create a class named MyClass, with a property named x:

```
class  MyClass:
  x = 5
```

Create Object/Accessing members

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

Editing class attributes

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Example

Insert a function that prints a greeting, and execute it on the p1 object:

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

  def myfunc(self):
    print("Hello my name is " + self.name)
    print(“my age is “+self.age)



p1 = Person("John", 36)
p1.myfunc()
```

Output:

```
Hello my name is John
my age is 36
```

Built-in class attributes

Following are the built-in class attributes.

Attribute	**Description**

__dict__	This is a dictionary holding the class namespace.
__doc__	This gives us the class documentation if documentation is present. None otherwise.
__name__	This gives us the class name.
__module__	This gives us the name of the module in which the class is defined. In an interactive mode it will give us __main__.
__bases__	A possibly empty tuple containing the base classes in the order of their occurrence.

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
      print "Total Employee %d" % Employee.empCount
   def displayEmployee(self):
    print "Name : ", self.name, ", Salary: ", self.salary
print "Employee.__doc__:", Employee.__doc__
print "Employee.__name__:", Employee.__name__
print "Employee.__module__:", Employee.__module__
print "Employee.__bases__:", Employee.__bases__
print "Employee.__dict__:", Employee.__dict__
```

Output

```
Employee.__doc__: Common base class for all employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: ()
Employee.__dict__: {'__module__': '__main__', 'displayCount':
<function displayCount at 0xb7c84994>, 'empCount': 2,
'displayEmployee': <function displayEmployee at 0xb7c8441c>,
'__doc__': 'Common base class for all employees',
'__init__': <function __init__ at 0xb7c846bc>}
```

Garbage collection/dynamic memory allocation

Python's memory allocation and deallocation method is automatic. The user does not have to preallocate or deallocate memory similar to using dynamic memory allocation in languages such as C or C++.
Python uses two strategies for memory allocation:

- Reference counting
- Garbage collection

Destroying objects.

A class implements the special method **__del__(),** called a destructor, that is invoked when the instance is about to be destroyed. This method might be used to clean up any non memory resources used by an instance.

Example

This __del__() destructor prints the class name of an instance that is about to be destroyed −

https://www.javatpoint.com/python-modules

https://www.tutorialspoint.com/execute_python_online.php

https://www.onlinegdb.com/online_python_compiler