1.1 Python Basics

1.1.1 Introduction: Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Python is named after a TV Show called 'Monty Python's Flying Circus' and not after Python-the snake. Some of the Features that Makes Python more popular are:

- Python is Simple and Easy to learn and code.
- Python is Free and Open Source. It is freely available at the https://www.python.org/. Python source code is also available to the public, one can download it, use it, and share it.
- Python is High Level Language and supports both Procedure oriented and Object-Oriented Language concepts along with dynamic memory management.
- Python is portable. Python code can be run on any platforms like Linux, Unix, Mac and Windows.
- Python is extensible and integrated. Python code can be extended and integrated with among other languages like C, C++, Java, etc.
- Python is an interpreted language. Python code is executed line by line at a time and there is no need to compile, which makes debugging easier. The
- Python has rich set of libraries for data analytics, machine learning, artificial intelligence, deep learning, mathematical computation, web app development, mobile app development, testing, etc.
- Python is a dynamically typed language. Here the data type for variable is decided at run time. As a result, there is no need to specify the type of variable.

1.1.2 Why One Should Learn Python Program?

Python Programming is a fun, creative and rewarding activity. Python is one of the most widely used programming language across the world for developing software applications. It is named as one of top picked programming languages of most of the universities and industries. Python developer is one of the "10 Most in Demand Tech Jobs of 2019"[Source : https://www.techrepublic.com/] As of February 23,

2019, the average salary for a Python developer is $123,201 per year in the United States, making it one of the most popular and lucrative careers today [source: https://www.codingdojo.com/].
Python can be used on the following:

1. Multiple Programming Paradigms
2. Web Testing
3. Data Extraction
4. Artificial Intelligence
5. Machine Learning
6. Data Science
7. Web Application and Internet Development
8. Cybersecurity

Entering Expressions into the interactive Shell

In Python, expressions are combinations of values, variables, operators, and function calls that can be evaluated to produce a result. They represent computations and return a value when executed. Here are some examples of expressions in Python:

Examples: `17, x, x+17 , 1+2*2 , X**2, x**2 + y**2`

Entering expressions into the Python interactive shell allows you to evaluate and execute code in real-time. You can use the shell as a convenient way to test and experiment with Python code. Here are some examples of entering expressions into the Python interactive shell:

Arithmetic Operations:
You can perform basic arithmetic operations, such as addition, subtraction, multiplication, and division, directly in the shell. For example:

```
>>> 2 + 3
5
>>> 4 * 5
20
>>> 10 / 3
3.3333333333333335
```

Variable Assignment:
You can assign values to variables and use those variables in expressions. For example:

```
>>> x = 5
>>> y = 2
>>> x + y
7
>>> x * y
10
```

Function Calls:
You can call built-in functions or user-defined functions within the shell. For example:

```
>>> abs(-10)
10
>>> len("Hello, World!")
13
>>> def add(a, b):
...         return a + b
...
>>> add(3, 4)
7
```

Boolean Expressions:
You can use boolean operators like and, or, and not to evaluate logical expressions. For example:

```
>>> True and False
False
>>> True or False
True
>>> not True
False
```

Conditional Statements:
You can use conditional statements like if, elif, and else to perform different actions based on conditions. For example:

```
>>> x = 10
```

```
>>> if x > 0:
...       print("Positive")
...    elif x < 0:
...       print("Negative")
...    else:
...     print("Zero")
...
Positive
```

Value: A value is a letter or a number. In Python, a value is a fundamental piece of data that can be assigned to variables, used in expressions, and manipulated by operations. Values can be of different types, such as numbers, strings, booleans, lists, tuples, dictionaries, and more. Each type of value has its own characteristics and behaviors.

Examples :

```
x = 10  # integer
y = 3.14  # floating-point number
z = 2 + 3j  # complex number
name = "John"  # string
message = 'Hello, World!'   # string
is_true = True  # Boolean Value
is_false = False
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "orange"]
coordinates = (10, 20)
person = ("John", 30, "USA")
student = {"name": "John", "age": 20, "grade": "A"}
```

type() function :

In Python, the type() function is used to determine the type of a given object or value. It returns the data type of the object as a result. The type() function is particularly useful when you need to programmatically determine the type of a variable or check if a value belongs to a specific data type. Here's an example:

Example: 1,2 and “Hello, World!”. Types are the data types to which the Values belong.
type(arg) function returns the data type of the argument as illustrated below :

```
x = 5
y = 3.14
name = "John"
is_true = True
fruits = ["apple", "banana", "orange"]
student = {"name": "John", "age": 20}

print(type(x))  # <class 'int'>
print(type(y))  # <class 'float'>
print(type(name))  # <class 'str'>
print(type(is_true))  # <class 'bool'>
print(type(fruits))  # <class 'list'>
print(type(student))  # <class 'dict'>
```

Data types

In Python, data types represent the classification or categorization of values. Each data type defines the operations that can be performed on the values, the storage format, and the behavior of the values. Python supports several built-in data types. Here are the commonly used data types supported in Python, along with examples:

Numeric Data Types:

int: Integers represent whole numbers, positive or negative, without decimal points.
Example: x = 5
float: Floating-point numbers represent decimal or floating-point values.
Example: y = 3.14
complex: Complex numbers consist of a real and an imaginary part.
Example: z = 2 + 3j

String Data Type:

str: Strings represent sequences of characters enclosed in single quotes (' ') or double quotes (" ").
Example: name = "John"

Boolean Data Type:

bool: Booleans represent either True or False, denoting logical values.
Example: is_true = True

Sequence Data Types:

list: Lists are ordered collections of items enclosed in square brackets ([]). They can contain values of any type and are mutable.
Example: fruits = ["apple", "banana", "orange"]
tuple: Tuples are similar to lists but are enclosed in parentheses (()). They are immutable, meaning their values cannot be changed once defined.
Example: coordinates = (10, 20)

Mapping Data Type:

dict: Dictionaries are unordered collections of key-value pairs enclosed in curly braces ({}). They provide a way to associate values with unique keys.
Example: student = {"name": "John", "age": 20}

Set Data Type:

set: Sets represent unordered collections of unique elements enclosed in curly braces ({}). They do not allow duplicate values.
Example: numbers = {1, 2, 3, 4, 5}

None Type:

None: The None type represents the absence of a value or a null value. It is often used to indicate the absence of a meaningful result.
Example: result = None

String Concatenation and Replication

String concatenation and replication are operations that allow you to combine and repeat strings in Python. Here's an explanation of each operation with examples:

String Concatenation:

String concatenation is the process of combining two or more strings together to create a single string. In Python, you can concatenate strings using the + operator. Here's an example:

```
greeting = "Hello"
name = "John"
message = greeting + " " + name
print(message)  # Output: Hello John
```

In this example, the + operator is used to concatenate the strings greeting, a space (" "), and name, resulting in the string "Hello John". The concatenated string is then stored in the message variable and printed.

String Replication:

String replication allows you to repeat a string multiple times. In Python, you can replicate a string by using the * operator. Here's an example:

```
fruit = "apple"
repeated_fruit = fruit * 3
print(repeated_fruit)  # Output: appleappleapple
```

In this example, the * operator is used to replicate the string "apple" three times, resulting in the string "appleappleapple". The replicated string is stored in the repeated fruit variable and printed.

String concatenation and replication can be combined to achieve more complex string operations. Here's an example that demonstrates both concatenation and replication:

```
word = "Hi"
punctuation = "!"
greeting = word * 3 + punctuation
print(greeting)  # Output: HiHiHi!
```

Variables:

A variable is a name that refers to a value. In Python, a variable is a named storage location that holds a value. It allows you to assign values to identifiers and refer to those values by using the identifiers later in the program. An assignment statement creates new variables as illustrated in the example below:

x = 10

In this example, the value 10 is assigned to the variable x. Now, x holds the value 10, and you can refer to it later in the program.

Examples:

Message = '*Python Programming* ',
p =1000, t= 2, r=3.142,
Si = p*t*r/100,
pi = 3.1415926535897931,
area_of _circle = *pi*r*r.*

To know the type of the variable one can use type () function. **Ex:** type(p)
To display the value of a variable, you can use a print statement:
Ex: print (Si) ; print(pi)

Rules for writing Variable names

1. **Variable names** can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
2. **Variable names** cannot start with a number/digit.
3. **Keywords** cannot be used as Variable names.
4. Special symbols like **!, @, #, $, %** etc. cannot be used in Variable names.
5. Variable names can be of any length.
6. Variable name must be of single word.

Table: Valid Variable Names and Invalid Variable Names

Valid Variable Names	Invalid Variable Names
python12	current- account(hyphens are not allowed)
Simple	savings account (spaces are not allowed)
interest_year	4freinds (can't begin with a number)
_rate_of_interest	1975 (can't begin with a number)
_spam	10April_$ (cannot begin with a number and special characters like $ are not allowed)
HAM	Principle#@(special characters like # and @ are not allowed)
account1234	'bear' (special characters like ' is not allowed)

Note:

Variable names are case-sensitive, meaning that velocity, VELOCITY, Velocity, and velocity are four different variables. It is a Python convention to start your variables with a lowercase letter.

Storing Values in a Variables:

Values can be stored in a variable using an Assignment statement. An assignment statement consists of a variable name, an equal (=) sign and the value to be stored.

Example 1:

x = 40

In this example, the value 10 is assigned to the variable x. The variable x now holds the value 10, and you can use x to refer to that value throughout the program.

Example 2:

a, b, c = 1, 2, 3

In this example, the values 1, 2, and 3 are assigned to variables a, b, and c respectively. Each value is assigned to its corresponding variable based on the order of appearance.

Example 3:

```
x = 5
y = 3
result = x + y
```

In this example, the variables x and y hold the values 5 and 3 respectively. The expression x + y is evaluated, and the result 8 is assigned to the variable result.

Example 4:

```
x = 10
x = x + 5  # x is updated to 15
```

In this example, the variable x initially holds the value 10. The expression x + 5 evaluates to 15, and that value is assigned back to the variable x.

My First Program :

```
# This program demonstrates the usage of common built-in functions in Python

# Using the print() function to display a message
print("Welcome to the Python function demo!")

# Using the len() function to determine the length of a string
text = input("Enter a word or phrase: ")
length = len(text)
print("The length of the entered text is:", length)

# Using the input() function to get user input
name = input("Enter your name: ")
age = input("Enter your age: ")

# Using the int() function to convert a string to an integer
age = int(age)
age_in_future = age + 10
print("In 10 years, you will be", age_in_future, "years old.")

# Using the str() function to convert an integer to a string
message = "Hello, " + name + "! You are " + str(age) + " years old."
print(message)
```

Dissecting the Sample Program

Sample program comprises executable statements containing comments and built in functions like print () , input () , len() , int() and str().

Explanation of the program:

The program starts with a **comment** explaining the purpose of the program.
The **print()** function is used to display the welcome message.
The **input()** function is used to prompt the user to enter a word or phrase. The value entered by the user is stored in the text variable.

The **len()** function is used to determine the length of the text string, and the result is stored in the length variable.
The program displays a message with the length of the entered text using the **print()** function.
The **input()** function is used again to get the user's name and age. The values entered by the user are stored in the name and age variables, respectively.
The **int()** function is used to convert the age value from a string to an integer.
The program calculates the user's age in 10 years by adding 10 to the age variable, and the result is stored in the age_in_future variable.
The program displays a message with the user's name, age, and the calculated age in 10 years using the **print()** function.
The **str()** function is used to convert the age integer back to a string so that it can be concatenated with other strings in the message variable.
The final message is displayed using the **print()** function.

The program demonstrates the usage of these functions: print() for displaying messages, len() for determining the length of a string, input() for obtaining user input, int() for converting a string to an integer, and str() for converting an integer to a string.

Comments:

Comments are readable explanation or descriptions that help programmers better understand the intent and functionality of the source code. Comments are completely ignored by interpreter.

Advantages of Using Comments:

1. Makes code more readable and understandable.
2. Helps to remember why certain blocks of code were written.
3. Can also be used to ignore some code while testing other blocks of code**.**

Single Line Comments in Python:

The hash symbol #is used to write a single line comment.

Example:

 # Printing a message
print(“ Enter your Name “)
myName = input (“ Enter Your Name”) **# Read your name to myName**

Multiline Comments in Python:

1. Using **#** at the beginning of each line of comment on multiple lines

Example:
It is a
multiline
comment

2. Using String Literals **'''** at the beginning and end of multiple lines

Example:
'''
I am a
Multiline comment!
'''

The print() Function :

The print function is used to display the string value written within pair of double quotes inside the parentheses on the screen .

print('It is Good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName))
print('You Will be ' + str(int(myAge)+1) + ' in a year.')

The line *print('The length of your name is:')* means “Print out the text in the string ‘'The length of your name is:’ . When Python executes the print statement, python interpreter calls the *print()*function and the string value is being *passed* to the function. The value within print() is called argument . Quotes within parentheses marks where the string begins and ends ; they are not part of the string value.

The input() function

This function is used to take the input from the user . Whatever the user enter as input, input() function convert it into a string . if you enter an integer value still input() function convert it into a string . The programmer is needed to convert it into an integer in your code using *typecasting*.
Myname = input("Enter your name")

Reading multiple values using input()

Programmer often want as user to enter multiple values in one line . In Python user can take multiple values or inputs in one line by using split() method . It breaks the given input by the specified separator. If separator is not provided then any white space is a separator. Generally, user use a split() method to split a Python string but one can used it in taking multiple input.

Example:

```
>>> x, y,z = input ("Enter three values").split()
Enter three values 2 3 4

>>> x
'2'
>>> y
'3'
>>> z
'4'
```

The len() Function

In Python, the len() function is used to determine the length of an object, such as a string, list, tuple, or any other iterable. It returns the number of elements or characters present in the object. Here's an explanation of the len() function with an example:

```
text = "Hello, World!"
length = len(text)
print(length)  # Output: 13
```

The str() , int() and float Functions :

The str(), int(), and float() functions in Python are used to convert values between different data types. Here's an explanation of each function with examples:

str() Function:
The str() function is used to convert a value to a string data type. It takes any valid Python object as an argument and returns a string representation of that object. Here's an example:

```
number = 10
number_str = str(number)
print(number_str)  # Output: "10"
```

str() function can be used convert integer or floating numbers into string data type.

int() Function:
The int() function is used to convert a value to an integer data type. It can convert a string or a float to an integer by truncating any decimal places. Here are some examples:

```
number_str = "20"
number_int = int(number_str)
print(number_int)  # Output: 20

float_num = 3.14
float_int = int(float_num)
print(float_int)  # Output: 3
```

In the first example, the int() function converts the string "20" to an integer value 20. In the second example, the int() function converts the float value 3.14 to an integer by truncating the decimal places, resulting in the value 3.

float() Function:
The float() function is used to convert a value to a floating-point data type. It can convert a string or an integer to a float. Here's an example:

```
number_str = "3.14"
number_float = float(number_str)
print(number_float)  # Output: 3.14
```

```
integer_num = 10
integer_float = float(integer_num)
print(integer_float)  # Output: 10.0
```

In the first example, the float() function converts the string "3.14" to a floating-point value 3.14. In the second example, the float() function converts the integer value 10 to a float value 10.0.

These functions are useful when you need to convert values between different data types in Python. They allow you to perform operations and manipulate values in the desired format.

Operators and operands

- Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called operands.
- The operators **+, -, *, /, and **** perform *addition, subtraction, multiplication, division, and exponentiation*, as in the following examples:

Table: Operators and Examples

Operator	Operation	Example	Evaluates to
**	Exponent	5**3	125
%	Modulus/Remainder	33%7	5
//	Integer Division/Floored quotient	33//5	6
/	Division	23/7	3.2857142857142856
*	Multiplication	7*8	56
-	Subtraction	8 – 5	3
+	Addition	7+ 3	10

Order of operations

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.

PEMDAS order of operation is followed in Python:

- **Parentheses** have the highest precedence and can be used to force an expression to evaluate in the order you want.
- **Exponentiation** has the next highest precedence,
- **Multiplication and Division** have the same precedence, which is higher than
- **Addition and Subtraction**, which also have the same precedence.
- **Operators with the same precedence** are evaluated from left to right.

Following examples illustrates the evaluation of expressions by Python interpreter. In each case the programmer must enter the expression, python interpreter evaluates the expression to a single value .

```
>>> 5+4*3
17
>>> (4+5)*3
27
>>> 12345678*45678
563925879684
>>> 3**5
243
>>> 22//7
3
>>> 22/7
3.142857142857143
>>> 27%5
2
>>>3 +  3
6
>>> (5-2)*((8+4)/(5-2))
12.0
```

Python interpreter evaluates parts of the expression as per the PEMDAS rule until it becomes a single value as illustrated below:

(5-2)*((8+4)/(5-2))

3 * ((8+4)/(5-2))

3*(12/(5-2))

3*(12/3)

3*4.0

12.0

Note: If you type invalid expressions, python interpreter will not be able to understand it and will display a SyntaxError message as illustrated below:

```
>>> 45+*
SyntaxError: invalid syntax
>>> 45++75)
SyntaxError: unmatched ')'
>>> 43-
SyntaxError: invalid syntax
>>> 43+5+*7
SyntaxError: invalid syntax
```

Python Character Set :

The set of valid characters recognized by Python like letter, digit or any other symbol. The latest version of Python recognizes Unicode character set. Python supports the following character set:

- **Letters** : A-Z ,a-z
- **Digits** :0-9
- **Special Symbols** : space +-/***()[]{}//=!= == <> ,”””,;:%!#?$&^⇦=@_
- **White Spaces** : Blank Space, tabs(->), Carriage return , new line , form feed
- **Other Characters** : All other 256 ACII and Unicode characters

Python Tokens:

A token (lexical unit) is the smallest element of Python script that is meaningful to the interpreter. Python has following categories of tokens: Identifiers, Keywords, Literals , operators and delimiters.

Identifiers: Identifiers are names that you give to a variable, class or Function. There are certain rules for naming identifiers similar to the variable declaration rules, such as : No Special character except_ , Keywords are not used as identifiers , the first character of an identifier should be _ underscore or a character , but a number is not valid for identifiers and identifiers are case sensitive .

```
# Variables
my_variable = 10
counter = 0
# Function
def calculate_area(length, width):
    return length * width
# Class
class MyClass:
    def __init__(self, name):
        self.name = name
# Module
import math
# Usage
rectangle_area = calculate_area(5, 3)
print(rectangle_area)  # Output: 15

my_object = MyClass("John")
print(my_object.name)  # Output: John

print(math.pi)  # Output: 3.141592653589793
```

In the above example, we have used identifiers like **my_variable, counter, calculate_area, MyClass, and math.**

Literals: A fixed numeric or non-numeric value is called a **literal**. Literals may be string, numbers (int, long, float and complex), Boolean (True or False), NONE and Operators.

1. Numeric literals: Numeric literals represent numeric values such as integers, floating-point numbers, and complex numbers.
Examples: x = 10, y = 3.14, z = 2 + 3j
2. **String literals:** String literals represent sequences of characters enclosed in either single quotes (') or double quotes (").
Examples: name = 'John', sage = "Hello, world!“
3. **Boolean literals**: Boolean literals represent the truth values True and False.
Examples: is_valid = True
4. **None literal:** The None literal represents the absence of a value or a null value. It is often used to indicate the absence of a meaningful result or as an initial value for variables.
Example: result = None
4. **Operator Literals** : Operator literals include arithmetic operators, comparison operators, assignment operators, logical operators, and more.
Examples: +,-,/,//,%,*,**, <,>,!=,==,and,or,not,etc.

Operators : A Symbol or a word that performs some kind of operation on given values and returns the result. There are 7 types of operators available for Python: Arithmetic Operator ,Assignment Operator, Comparison Operator, Logical Operator , Bitwise Operator , Identity Operator and Membership Operator .

1. Arithmetic operators: +, -, *, /, %, **, //
2. Assignment operators: =, +=, -=, *=, /=, %=, **=, //=
3. Comparison operators: ==, !=, >, <, >=, <=
4. Logical operators: and, or, not
5. Bitwise operators: &, |, ^, ~, <<, >>
6. Membership operators: in, not in
7. Identity operators: is, is not

Delimiters: Delimiters are the symbols which can be used as separators of values or to enclose some values.

> **Examples** : Comma (,),Colon (:),Parentheses ((and)),Square brackets ([and]),Curly braces ({ and }),Quotation marks (' and ") and Backslash (\)

Note : Comments and # symbol used to insert a comment is not a token.

Keywords: The reserved words of Python which have a special fixed meaning for the interpreter are called keywords. No keyword can be used as an identifier or variable names. There are 35 keywords in python as listed below:

Keyword	Description
and	Logical and operator
as	Alias
assert	Used for debugging
async	Used to make a function asynchronous by adding the async keyword before the function's regular definition
await	Used in asynchronous functions to specify a point in the function where control is given back to the event loop for other functions to run. You can use it by placing the await keyword in front of a call to any async function
break	To break out of a loop
class	To define a class
continue	For skipping the statements and conitinuing the next iteration
def	For defining user defined functions
del	To delete an object
elif	Conditional statement, same as else if
else	Conditional statement
except	Used in exception handling
False	Boolean Value
finally	Used in exception handling , to execute a block of code no matter whether exception is there or not
for	Used to create for loop – iterative statement
from	Used to import specific parts of a module
global	Used to declare global variable
if	Conditional /decision making statement
import	Used to import a module or library
in	Used to check if a value if present in list, tuple, dictionaries , sets ,etc.
is	To check if two variables are equal
lambda	Used for defining an anonymous function
None	Used to represent a null value
nonlocal	To declare a non-local variable

not	A logical operator
or	A logical operator
pass	A null statement , a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean Value
try	Used in exception handling
while	For creating a while iterative loop
with	Used to simplify exception handling
yield	To end a function , returns a generator

Following code segment can be used to obtain the list of python keywords :

```
import keyword
# Get the list of keywords
print(keyword.kwlist)
print("\n Total Number of Keywords: ",len(keyword.kwlist))

# Output :

['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
 Total Number of Keywords:  36
```

Simple Python Programs

1. **To perform basic calculations**

```
# Sample numbers
num1 = int(input("Enter first number"))
num2 = int(input("Enter second number"))

# Calculate the sum of two numbers
sum_result = num1 + num2
print("Sum:", sum_result)

# Calculate the product of two numbers
product_result = num1 * num2
print("Product:", product_result)

# Calculate the difference of two numbers
difference_result = num1 - num2
print("Difference:", difference_result)

# Calculate the division of two numbers
division_result = num1 / num2
print("Division:", division_result)

# Calculate the integer division of two numbers
integer_division_result = num1 // num2
print("Integer Division:", integer_division_result)

# Calculate the modulo division of two numbers
modulo_division_result = num1 % num2
print("Modulo Division:", modulo_division_result)
```

 # output

```
Enter first number 3
Enter second number 2
Sum: 5
Product: 6
```

Difference: 1
Division: 1.5
Integer Division: 1
Modulo Division: 1

2. To find the area of a circle

```
import math
# Take input for the radius of the circle
radius = float(input("Enter the radius of the circle: "))
# Calculate the area of the circle
area = math.pi * radius**2
# Display the result
print("The area of the circle is:", area)
```

 # output

Enter the radius of the circle: 2.5
The area of the circle is: 19.634954084936208

3. To find the simple interest

```
# Take input for principal amount, rate, and time
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the interest rate: "))
time = float(input("Enter the time period (in years): "))

# Calculate the simple interest
simple_interest = (principal * rate * time) / 100

# Display the result
print("The simple interest is:", simple_interest)
```

 # Output
Enter the principal amount: 10000
Enter the interest rate: 2.5
Enter the time period (in years): 3
The simple interest is: 750.0