

Computational Thinking and Programming - 1

Getting started with Python



Introduction To Python

Python is a general purpose interpreted, interactive, object-oriented programming language. It was created by **Guido van Rossum** in the late eighties and early nineties. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Features of Python

- ❖ It closely resembles the English language. It is a **programmer friendly** language.
- ❖ It is **free and open source**, i.e. it is freely available (can be downloaded from www.python.org). Open source means its source code (complete program instructions) is also available. You can modify, improve/extend an open source software.
- ❖ It is a **high level language**, interpreter based which means that the Python interpreter interprets and executes the code line by line. It also makes debugging easier.
- ❖ It is an **object oriented** programming language.

Features of Python

- ❖ Python supports modules and packages, which encourages program modularity and reuse of code.
- ❖ It is a **platform independent** portable Programming Language i.e It can work on various operating systems like. Windows, Linux/UNIX, Macintosh. Smart Phones.
- ❖ Python can be used in many diverse fields / applications. Some examples of these applications are Web application, GUI programming, app development, Game Development, Database application etc.
- ❖ It is blessed with a large community.
- ❖ Python can be used to create desktop and web based applications.
- ❖ It has a **rich standard library** (<https://docs.python.org/3/library>) containing predefined functions for most of the common operations, and a large number of predefined modules (<https://docs.python.org/3/py-modindex.html>) for specific varieties of functions. This makes programming an easier task.

Minuses of Python Language

- It is not the fastest language. Since Python is an interpreter based language. Development time might be fast but execution time is not as fast compared to other languages.
- Python has lesser libraries compared to languages like C, Java, Pearl. At times they have better and multiple solutions than Python.
- Not strong on type binding. It is not strong on catching 'Type mismatch' issues. If you declare a variable as an integer, and later store a string in it, it will not complain.
- Because of the lack of syntax, Python is an easy language to program in. But difficult to convert a program written in Python to another language, since other languages have a structure defined syntax or a strong syntax.

Application Domain

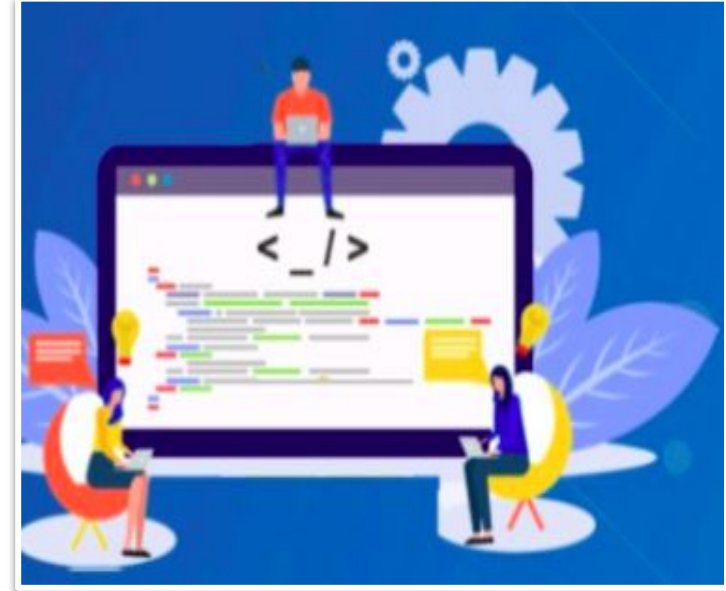
Python is used in many diverse applications. A list of few application areas where Python is used are:

- Web and Internet development
- Database Access
- Desktop GUIs
- Data Sciences
- Network Programming
- Game and 3D Graphics



Python Interface

To develop and execute programs in Python, the Python interpreter is required to be installed. Python provides an **IDLE** (Integrated Development and Learning Environment) for typing, debugging, editing and executing the program.

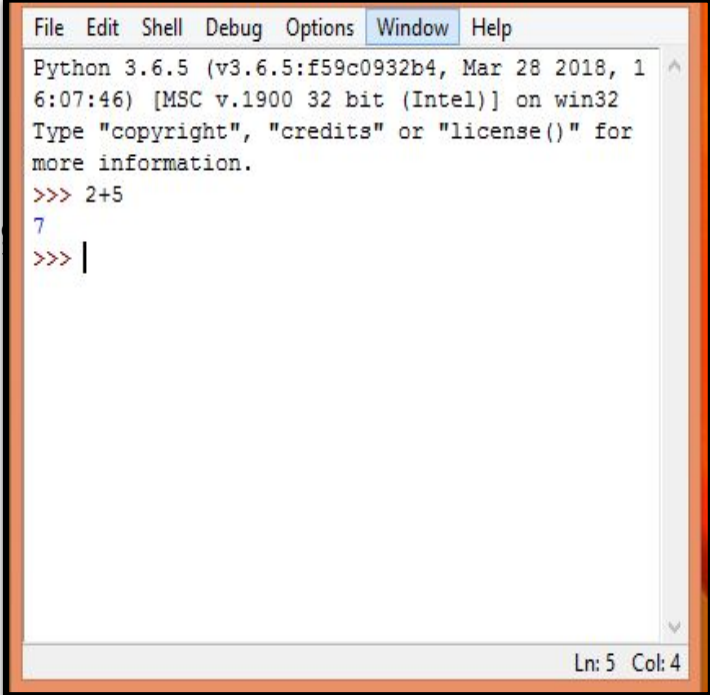


Python's IDLE can work in two basic modes :
Interactive mode and Script mode.

Interactive Mode

When Python IDLE is activated a window entitled as “**Python Shell**” will be displayed on the screen. This screen refers to the **interactive mode** of Python. In this mode, the command, one at a time is entered right of the **Python prompt (>>>)** and Python executes the command there and then and the output.

This mode is useful for testing out individual statements. However, interactive mode does not save the command.

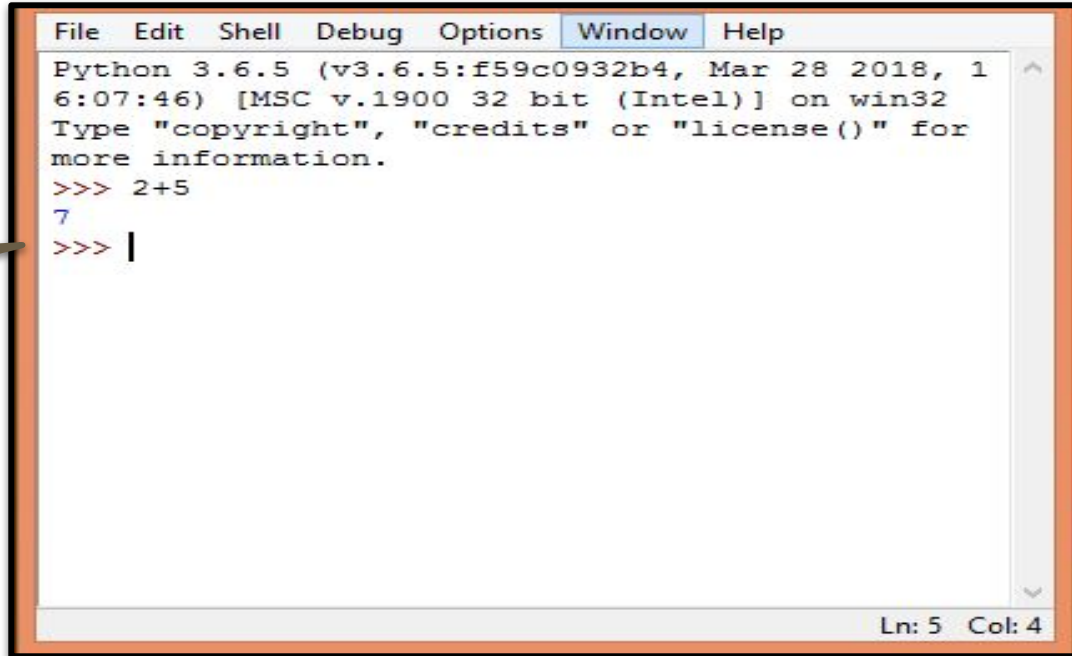


```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
6:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>> 2+5
7
>>> |
```

Ln: 5 Col: 4

Interactive Mode

Python
Prompt



```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 1
6:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>> 2+5
7
>>> |
```

Ln: 5 Col: 4

To show / edit / rerun prior command in interactive window, the up arrow and down arrow keys can be used. The up arrow key (\uparrow) can be used to scroll backward through commands history and down arrow key (\downarrow) key to scroll forward.

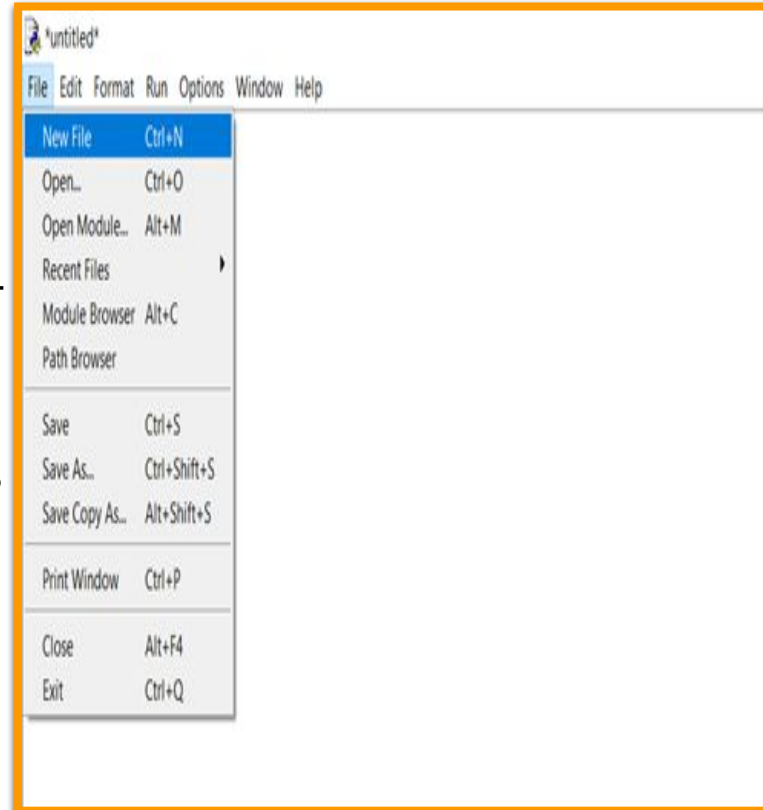
Script Mode

In this mode the python statements are typed in the editor and saved in a .py file.

Then the Python interpreter is used to execute the contents from the file.

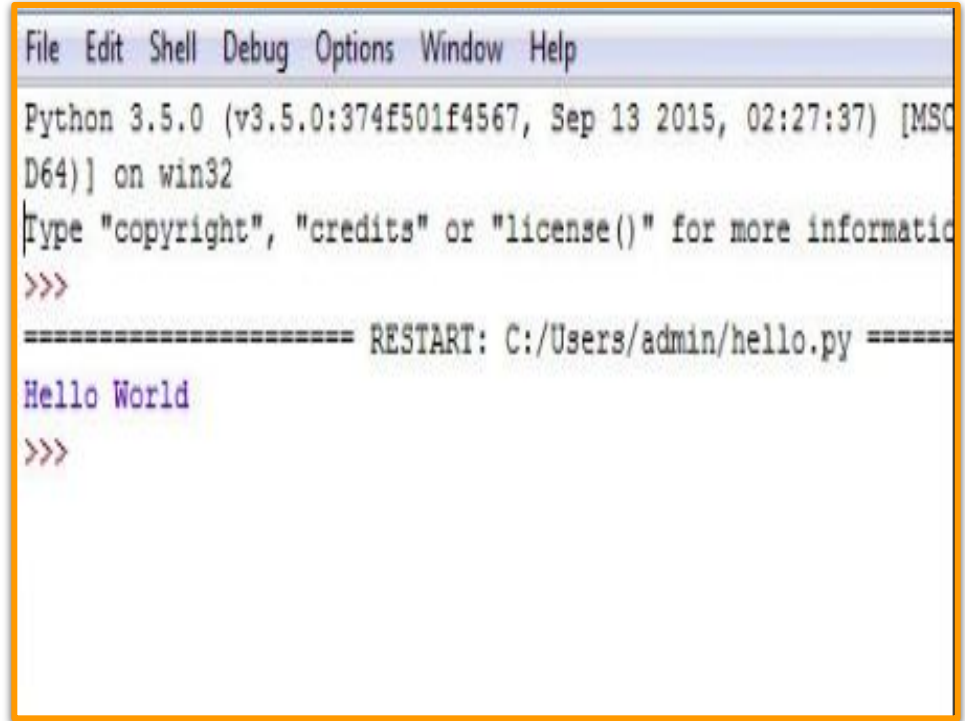
To activate the script mode , follow the given steps:-

- From the python shell window select **File ->New File**. A window entitled as “Untitled” will be opened where the statements can be entered.
- Once the program is ready, first save the file using **File->Save as** .
The file will be saved using **.py extension**.



Script Mode

- To execute the program select **Run ->Run Module (F5 key)** .
The output of the program will be shown in the Python shell window.
The file has to be saved before its execution.



The screenshot shows a Python IDE window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a text area. The text area contains the following text:

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

Basics of the Python Language

Python is a **case sensitive language**.

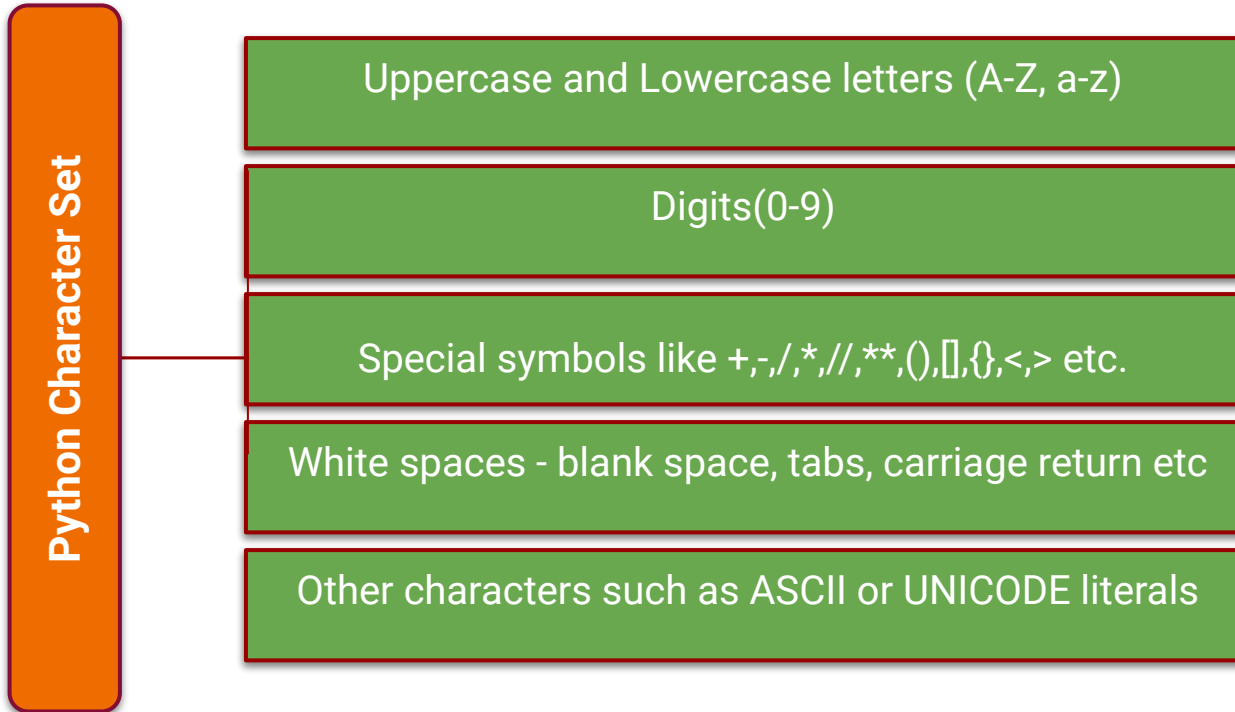
It means Python differentiates between upper and lower case alphabets.

For example, **Print** and **print** are two different words for Python.

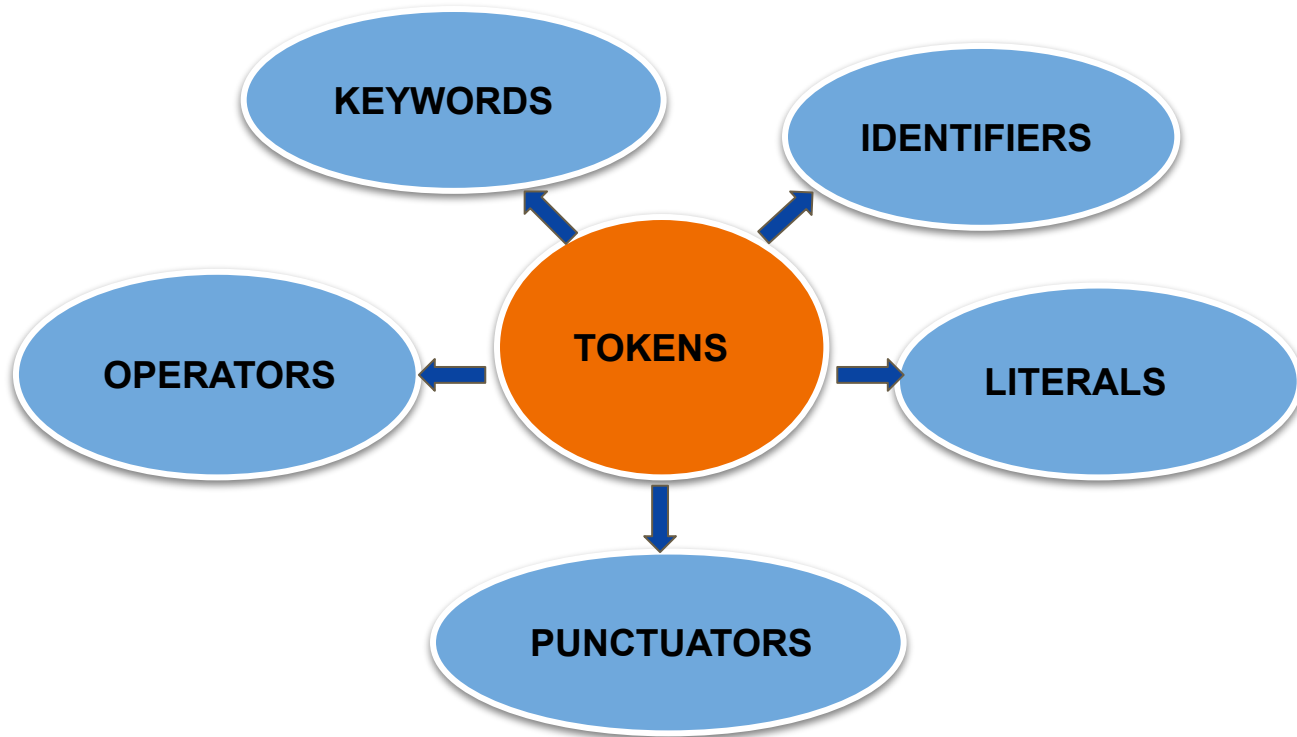
Print is just a word and not a command, whereas print is a valid command in Python.

Python Character Set

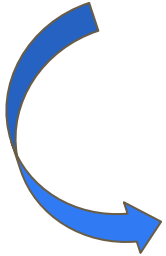
Character set is a set of valid characters that a language can recognize.



TOKENS



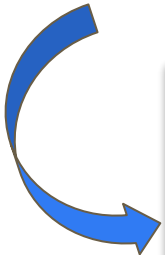
Keywords



Keywords are the words that convey **special meaning** to the Python interpreter. These are reserved for special purpose and must not be used as normal identifier names. In python, keywords contain lower case letters only.

TOKENS

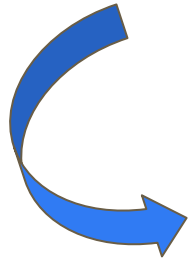
Keywords



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

TOKENS

Identifiers



Identifiers are fundamental building blocks of a program and are used to name different parts of a program, i.e. variables, objects, functions, lists, tuples etc.

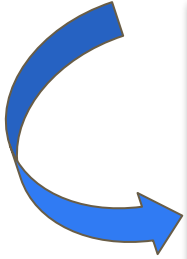
Variables

Objects

Functions

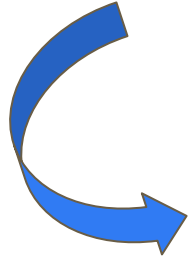
Lists or tuples

Rules for naming Python identifiers:



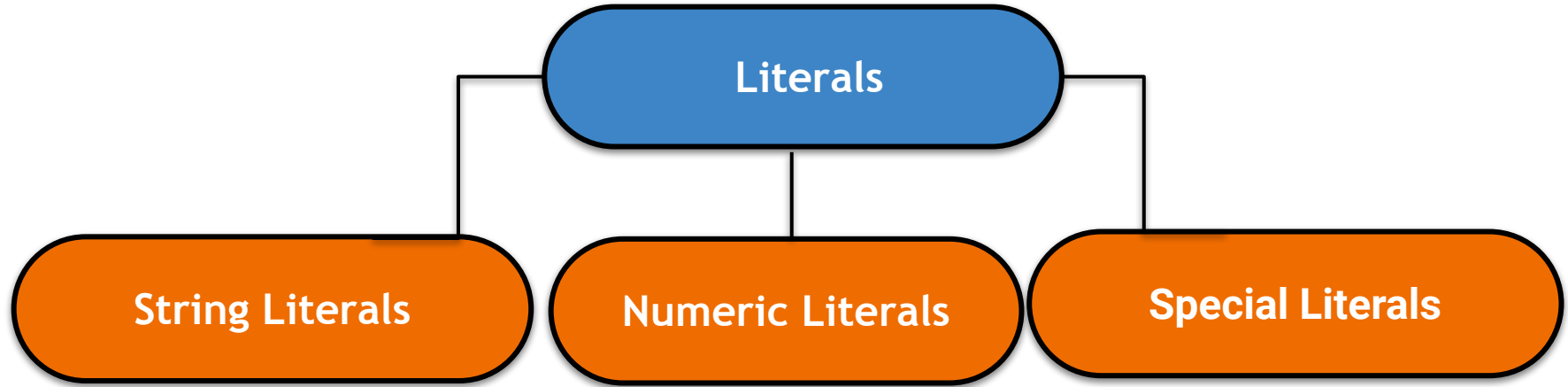
- An identifier is an arbitrarily long sequence of letters and digits
- The first character must be a alphabet letter or an underscore (_)
- Identifier names can contain uppercase alphabets (A - Z), lowercase letters (a-z), digits (0-9) and underscore.
- No special characters other than an underscore can be used.
- Identifier names are case sensitive and can be unlimited in length
- A keyword or reserved word cannot be used as an Identifier
- Names used for object, function, List, String, Dictionary etc. are examples of Identifiers.

Literals



Literals refer to the data items, which do not change their values during a program execution (often referred to as constant values). Depending on the data types of the data items, Literals can further be categorized into different categories :

String literals, Numeric literals and special literals



String Literals

The text enclosed within quotes forms a string literal in Python. For example, 'a', 'aa', "a" are all string literals in Python. A string literal is a sequence of characters within quotes (single, double or triple) .

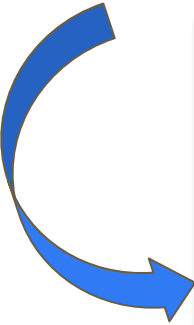
One can form a string literal by enclosing text in both forms of quotes - single or double.

Some valid string literals are:

'Astha', "Mehra", 'Hello World', "Raj's", "12345", "1A2B",
"a-b-0-d", "return"

String Literals

Types of Strings



Python allows you to have two string types, **Single line** strings and **Multi line** strings.

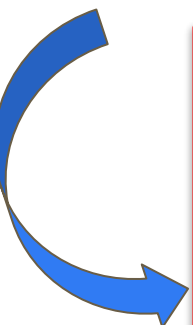
Single line strings - The strings you create by enclosing text in single quotes or double quotes are single line strings i.e., they must terminate in one line.

Example: Valid Single line string

```
Str1 = 'hello'
```

```
Str2="how are you"
```

String Literals



Multiline strings - Sometimes you need to store some text spread across multiple lines as one string. For that Python offers multiline strings which are created in two ways :

- By adding a backslash at the end of the normal single-quoted/ double quoted strings.

Example: Valid multiline string

```
Str1 = 'hello\  
        Friends'
```

String Literals

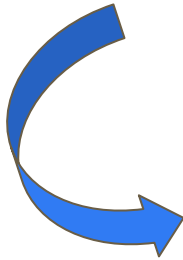
- 
- By typing the text in triple quotation marks (No backslash needed)

Example: Valid multiline string

```
Str2 = "Hello World  
      Welcome"
```


String Literals

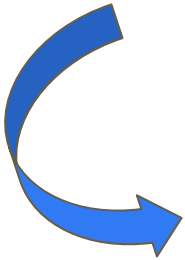
Escape Sequences



Python also allows non graphic characters in string values. Non graphic characters are those which cannot be typed from the keyboard directly, example backspace, tab, carriage return etc. These non graphic characters can be represented by using escape sequences. An escape sequence is represented by a \ (backslash) followed by one or more characters.

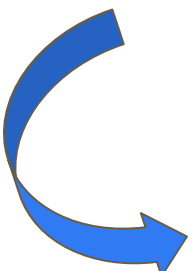
String Literals

Escape Sequences



Code	Result/Output	Description
\'	Single Quote	Add single quote with in a String
\\	Backslash	Insert single Back Slash
\n	New Line	\n takes the cursor to first position of a new line
\r	Carriage Return	\r takes the cursor to the first position of the same line
\t	Tab	\t add spaces of 8 characters
\b	Backspace	\b takes the cursor one position backward

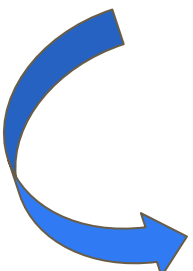
String Literals



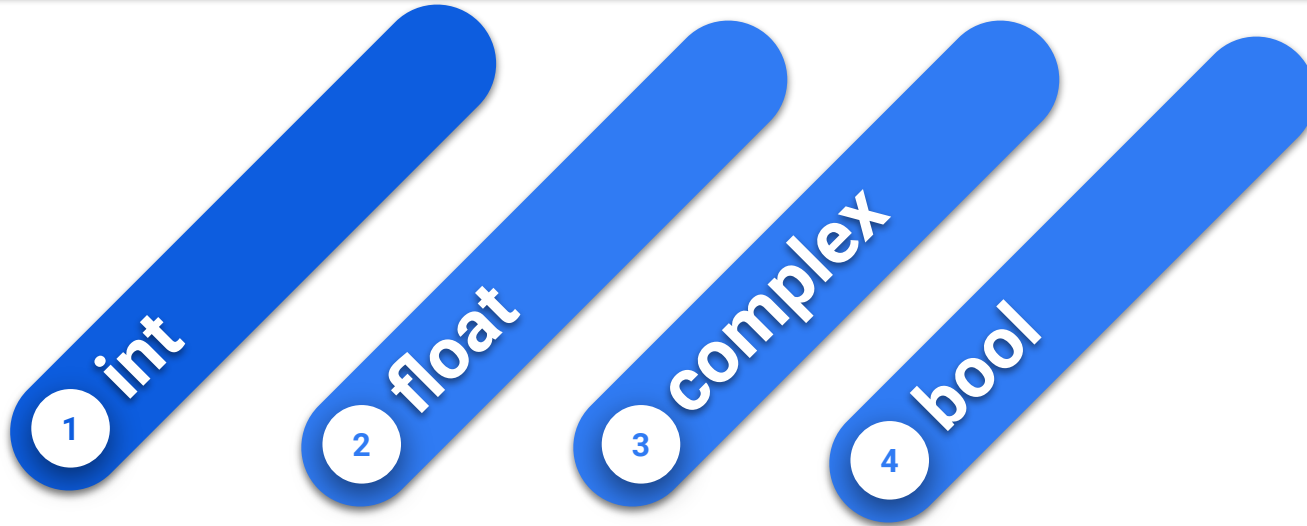
Size of String: Each character of a Python string takes 1 byte of memory. Python determines the size of a string by counting the total character present in the string. Any escape sequence present in a string will take one byte of memory.

Example	Size
'\\'	1
'abc'	3
'\ab'	2
'Seema\'s pen'	11

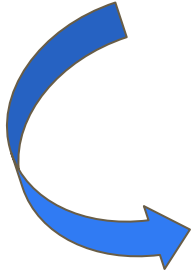
Numeric Literal



There are four types of numeric literals: **int** (Integer), **float** (floating-point numbers), **bool** (Boolean values) and **complex**. They have the features of their given data type.



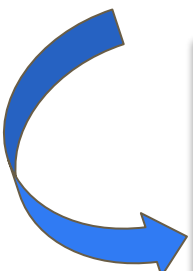
Numeric Literal



Examples of Numeric Literal

int	float	complex	bool
123	45.68	2+3j	True
-912	2.3E+3	9-2j	False
433	6.6E-2	-12+7j	

Special Literal



Python has one special literal called None. The None literal is used to indicate the absence of a value. It is used to indicate the end of a list in Python. The None value means “There is no useful information” or “There is nothing here”.

Example :

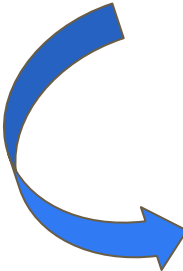
```
value1=1
```

```
value2=None
```

```
print(value2)
```

```
#displays None
```

Operators



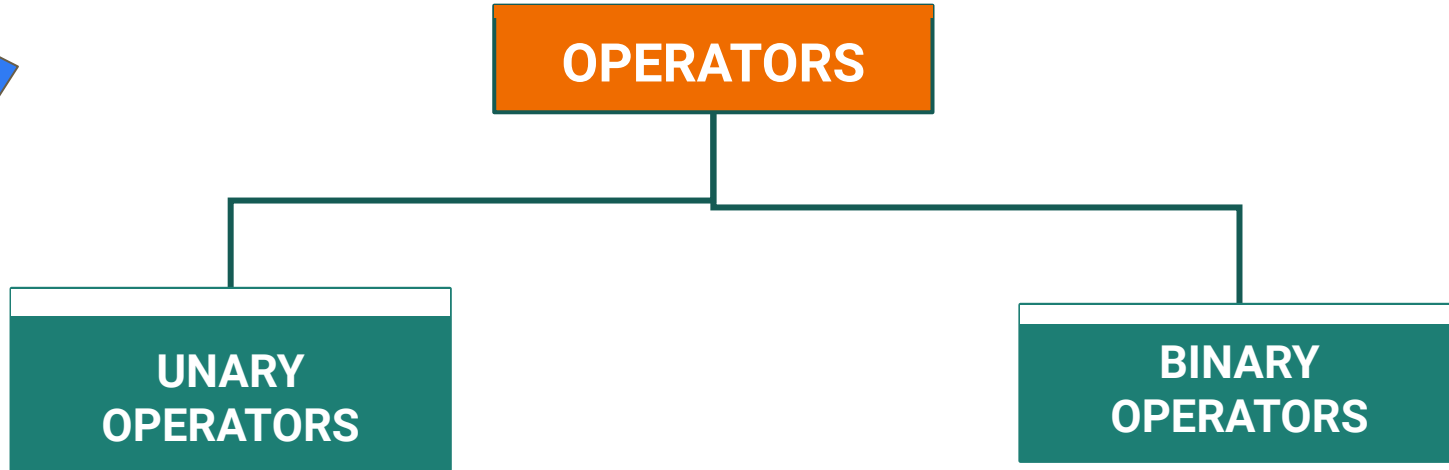
They are tokens which trigger some computation when applied to variables or other objects in an expression.

Variables and objects to which the computation applies are called operands. So an operator requires operands to work on.

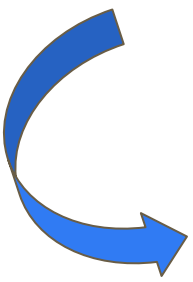


Operators

Depending on the number of operands , operators are classified as:



Operators



UNARY OPERATORS

Unary operator
+

Unary operator
-

BINARY OPERATORS

Arithmetic Operators

Relational Operators

Logical Operators

Membership Operators

Assignment Operators

PUNCTUATORS

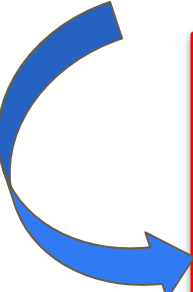


They are symbols used in programming to organise sentence structures.

' (single quote)	\ (backslash)	{ } (Curly braces)
" (double quote)	() (Parenthesis)	@ (at the rate)
# (hash)	[] (Square bracket)	, (comma)
: (colon)	. (dot)	

BAREBONES OF A PROGRAM

Expression

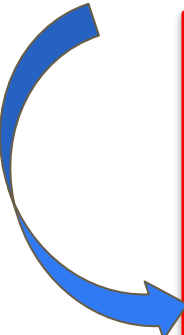


An expression is defined as a combination of constants, variables and operators. An expression always evaluates to a value. A value or a standalone variable is also considered as an expression but a standalone operator is not an expression. Some examples of expressions are given below:

- 100
- 70-20
- 2.7 + 3.4
- " Hello " + " World "

BAREBONES OF A PROGRAM

Statement



Expression represents a value, statement is a programming instruction that does something, that is, some action takes place.

```
print("hello")
```

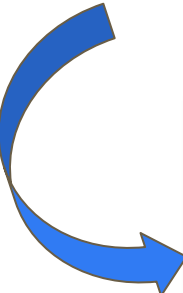
```
if b>5:
```

Statement is something that gets executed. Not necessary that statement produces a value, it may or may not yield a value.

```
a=10, b=a-10, print(a+3), if b>15:
```

BAREBONES OF A PROGRAM

Comments




Comments are used in Python program to allow the programmer to insert small explanatory notes or description to enhance readability or understandability of the program.

Python Interpreter does not execute the code given as comment.

Comments can be given as a Single Line Comment or a MultiLine Comment

BAREBONES OF A PROGRAM

Types of Comments

- 
- **Single Line Comment** - Single Line comment in Python start with the # symbol. Anything written after # in a line will not be executed by Interpreter.


Example :

```
side=float(input("Enter Side-"))  
#calculating area  
area=side**2  
print("Calculated area=",area,end=" ")
```

Single line comment

BAREBONES OF A PROGRAM


Types of Comments

- 
- Multi-Line Comment - Multiple line comment or Block comment can be entered in Python in two ways
 - By adding # symbol in the beginning of every physical line of Multi-line comment.

```
side=float(input("Enter Side-"))  
#Calculating  
#area of Square  
area=side**2
```

BAREBONES OF A PROGRAM

Types of Comments


- 
- Multi Line comment can also be entered as triple-quoted Multi-line string. Comments enclosed in triple-quotes are known as Doc String. Triple single quotes(' ' ') or triple double quotes (" " ") can be used to write doc string.

Example:

```
'''The following statement is used to  
display the output My First Python'''  
print('My First Python')
```


BAREBONES OF A PROGRAM

Blocks and Indentation



Sometimes a group of statement(s) is part of another statement or a function. Such a group of statements is called a block or a code block or suite.

Example

```
if b<5:
```

```
    print("Value of b is less than 5")
```

```
    print("Thank you")
```



Block

Four spaces (**can be changed**) together mark the next indent level. This is a block with all its statements at the same indentation level.

Variables

A variable comes into existence when we assign a value to it. Python is not "statically typed". We do not need to declare the type of a variable prior to its use. The moment you assign a value to it, a variable in Python is automatically created.

It is a "dynamically typed" language, the data type of a variable is dependent on the type of data which it holds.

The creation of a variable is simple, just use the name of the variable, followed by the = (equals to) symbol and then the value.

```
My_Name = "DPS Mathura Road"
```

The above creates a variable called `My_Name`, and then assigns the address of the string literal `"DPS Mathura Road"` to the variable `My_Name`

Variables and Assignments

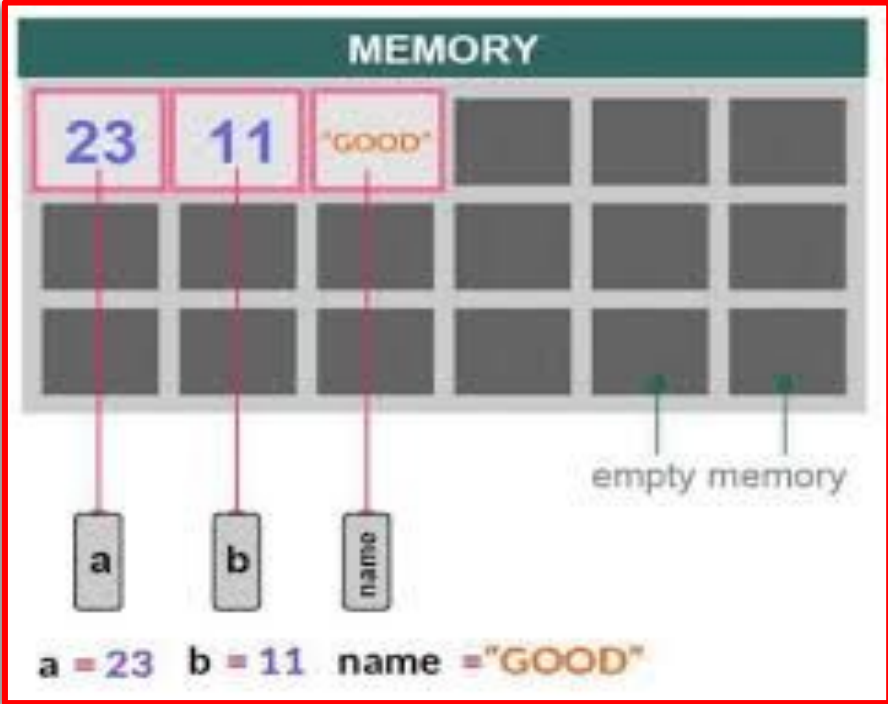
If we declare variables like :

a = 23

b = 11

name = "GOOD"

a, b, and name become pointers that store the address of the locations holding the values 23, 11 and "GOOD" respectively.



Variables and Assignments

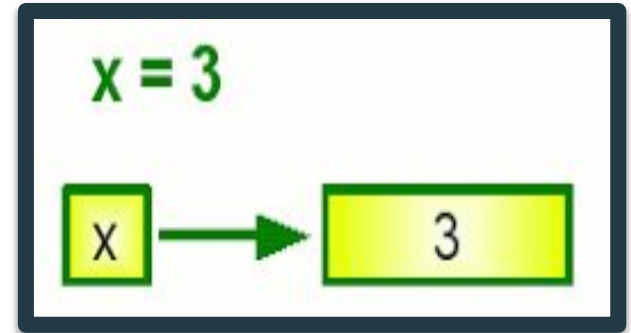
In Python a variable is a pointer/indicator/locator which stores the address of a memory location.

For ex., the statement `x=3` will store a value 3 in some memory location and variable `x` will point to that memory location.

`id()` function is used to find address of memory location to which variable is referring

```
>>>x=3
```

```
>>>id(x)           #displays memory address
```

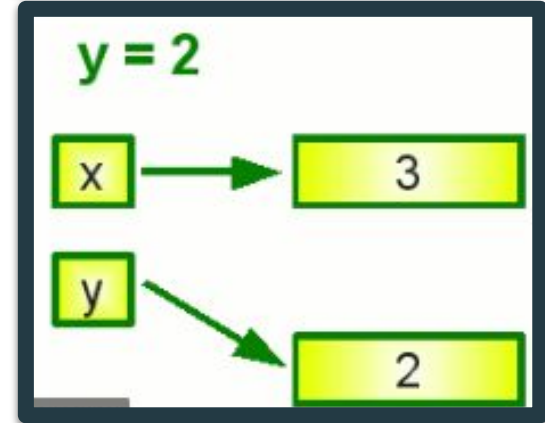
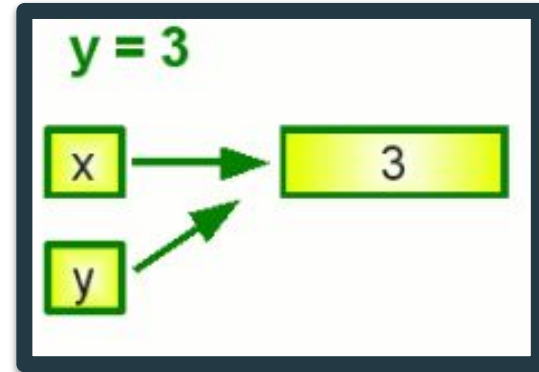


Variables and Assignments

Another statement `y=3` will also point to same memory location.

```
>>>x=3
>>>id(x)    #displays memory address
140004345567654
>>>y=3
>>>id(y)    #displays memory address
140004345567654

>>>y=2
>>>id(y)    #displays memory address
140004345522222
```



Data types - Integer, Float, String or Boolean

The type of a literal/variable/identifier can be determined by the command `type()` in Python as shown below:

```
>>> type(111)
<class 'int'>
```

Integer type

```
>>> type(12.45)
<class 'float'>
```

Float type

```
>>> type(True)
<class 'bool'>
```

Boolean type

```
>>> type("Hello World")
<class 'str'>
```

String type

Declaring Multiple Variables

In Python, we can declare (or assign values to) multiple variables in a single statement. There are two ways of doing this:

(i) `var1, var2, var3, . . . , varn = exp1, exp2, exp3, . . . , expn`

This statement assign values of expressions `exp1, exp2, ..., expn` to the variables `var1, var2, . . . , varn` respectively.

Some examples are:

```
>>>a,b,c=65,"Hello",23.5
```

Value 65 will be assigned to variable a

#Value "Hello" will be assigned to variable b

#Value 23.5 will be assigned to variable c

Declaring Multiple Variables

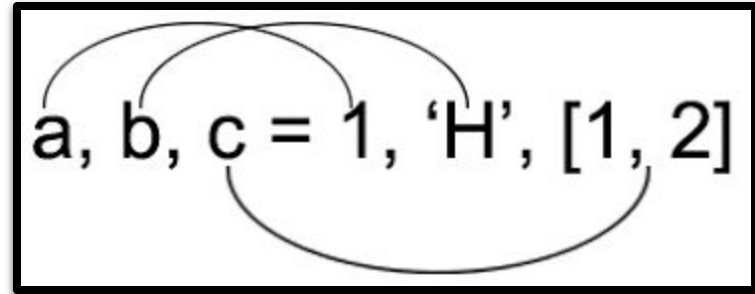
```
Num, Str, Val=10, "Computer", 24.6  
print("Number = ", Num)  
print("Value = ", Val)  
print("Text = ", Str)
```

OUTPUT:

Number = 10

Value = 24.6

Text = Computer



Declaring Multiple Variables

(ii) $\text{var1}=\text{var2}=\text{var3}=\dots=\text{var}_n = \text{expression}$

This statement will assign value of expression to all the variables var1 , var2 , \dots , var_n .

For example, the statement

```
>>>a=b=c=d=0
```

#assigns value 0 to the variables a, b, c, and d.

Note : A variable is not created until some value is assigned to it.

Naming a variable

There are certain rules in Python which have to be followed to form valid variable names.

Any variable which violates any of these rules will be an invalid variable name.

The rules are:

(i) A variable name must start with an alphabet (capital or small) or an underscore (_).

(ii) A variable name can consist of UNICODE alphabets, digits, and underscore(_). No

other character is allowed.

(iii) A Python keyword cannot be used as a variable name.

Examples of some valid variable names are:

st_name, father_name, TotalValue, Age, Bal_Qty, A123, BillNo9

Some invalid variable names :

Invalid Variable Name	Reason
roll number	Space is not allowed
D.P.S	No other character (.) is allowed
9years	Variable name should not start with digit
while	while is a keyword, so it cannot be taken as variable name
No#	No special character is allowed

Simple Input Statement

In Python, data is input from the user during script execution using `input()` function.

The `input()` function takes one argument (called prompt). This argument is generally a string prompting the user to enter a value. During execution, `input()` shows the prompt to the user and waits for the user to input a value from the keyboard. When the user enters value from the keyboard, `input()` returns this value to the script. In almost all cases, we store this value in a variable.



```
>>>name=input("Enter your name :")
```

```
Enter your name : Anu
```

Prompts the user
to enter value

The name entered by the user is stored in variable “name”

Simple Python Script/Program

```
name1=input("Enter a name: ")  
name2=input("Enter another name: ")  
print(name1,"and",name2,"are friends")
```

OUTPUT :

Enter a name: Mohit

Enter another name: Alok

Mohit and Alok are friends

Simple Python Script/Program

Program to input two numbers and display their sum and product.

```
n1 = input("Enter first number: ")
n2 = input("Enter second number: ")
sum = n1+n2
print("Sum of the numbers=",sum)
pro = n1*n2
print("Product of the numbers=",pro)
```

Simple Python Script/Program

OUTPUT :

```
Enter first number: 10
Enter second number: 20
Sum of the numbers= 1020
Traceback (most recent call last):
  File "/Users/monicasahni/Documents/multiplevar.py", line 5, in
<module>
    pro = n1*n2
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

Simple Python Script/Program

We observe the following problems with the above code:

- The sum is not correct. Instead of being added, the numbers are being joined.
- An error occurred in line 5 of the code (`pro = n1*n2`). The Python interpreter is showing it as `TypeError` and flashes the message “can’t multiply sequence by non-int of type ‘str’”

The solution is to use function `int()`, `float()`, and `eval()`.

int() function

This function takes a number, expression, or a string as an argument and returns the corresponding integer value. Different types of arguments are treated as follows:

1. if argument is an integer value, `int()` returns the same integer.
For example: `int(12)` returns 12.
2. if the argument is an integer expression, the expression is evaluated, and `int()` returns this value.
For example: `int(12+34)` returns 46.
3. if the argument is a float number, `int()` returns the integer part (before the decimal point) of the number.
For example: `int(12.56)` returns 12.

int() function

4. If the argument is a float expression, the expression is evaluated, and int() returns the integer part of this value.

For example: `int(12+13/5)` returns 14.

5. If the argument is a string containing leading +/- sign and digits only, int() returns the integer represented by this string.

For example: `int('12')` returns 12.

6. If the string argument contains any character other than leading +/- sign and digits, then int() results in an error.

For example:

The following statements will result in errors:

`int('12/3'), int('12+3'), int('23a')`

int() function

```
>>> int(38)
38
>>> int(9+8)
17
>>> int(5+8/9)
5
>>> int(67.8)
67
>>> int('21')
21
>>> int('3.09')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.09'
```

float() function

This function takes a number, expression, or a string as a parameter and returns the corresponding float value. Different types of parameters are treated as follows:

1. if the argument is a number, float() returns the same number as a float.
For example: float(12) returns 12.0, and float(12.46) returns 12.46.
2. if the argument is a numeric expression (arithmetic or algebraic), the expression is evaluated, and float() returns its value.
For example: float(12+34) returns 46.0, and float(12+3/4) returns 12.75.
3. If the argument is a string containing leading +/- sign and a number in correct format, float() returns the float value represented by this string.
For example: float ('12.45') returns 12.45.

float() function

4. If the string argument contains any character other than leading +/- sign and floating point number in the correct format, then float() results in an error.

For example: the following statements will result in errors:

`float('12.2.3'), float('67.6-'), float('23+3/5')`

5. If no argument is passed, float() returns 0.0

float() function

```
>>> float(311)
311.0
>>> float(34+20)
54.0
>>> float(2+5.6)
7.6
>>> float(10+5/6)
10.833333333333334
>>> float(3.4)
3.4
```

```
>>> float('-3.45')
-3.45
>>> float('98.7degrees')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '98.7degrees'
>>> float('4+6/2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '4+6/2'
```

eval() function

It takes a string (not a number) as an argument, evaluates this string as a number, and returns the numeric result (int or float as the case may be). If the given argument is not a string, or if it cannot be evaluated as a number, then eval() results in an error.

```
>>> eval('23')
23
>>> eval('-31.9')
-31.9
>>> eval('3+45')
48
>>> eval('2+4/5')
2.8
>>> eval('2+3pens')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<string>", line 1
      2+3pens
      ^
SyntaxError: unexpected EOF while parsing
```

eval() function

```
#Demonstration of eval() function
x=1
print(eval('x'))
print(eval('x+7'))
print(eval("4==5"))
print(eval("3<4"))
print(eval('2*9+3/4'))
print(eval("'good'*5"))
```

OUTPUT :

```
1
8
False
True
18.75
goodgoodgoodgoodgood
```


print() function

It is a function which is used to display the specified content on the screen.

Syntax:

```
print (objects, [sep=' ' or <sep-string> end ='\\n' or <end-string>])
```

The content, called argument(s) is/are specified within the parentheses.

print() function

Python is a case-sensitive language. It means that Python differentiates between capital and small letters. For example, Print (P capital) and print (p small) are two different words for Python. Where print is a valid command in Python, Print is just a word and not a command.

```
>>> print("hello students")
hello students
>>> Print("hello students")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
```

In the print command, if the argument is put within quotation marks (single " or double ""), then the argument is displayed as it is.

print() function

If the argument is not placed within quotation marks, then the interpreter tries to evaluate the argument and displays the result. This feature is used to perform calculations.

```
>>> print(2+10)
12
>>> print(3+4/3)
4.333333333333333
>>> print(22/7)
3.142857142857143
```

print() function

If the argument is not placed within quotation marks, then the interpreter tries to evaluate the argument and displays the result. This feature is used to perform calculations.

```
>>> print(2+10)
12
>>> print(3+4/3)
4.333333333333333
>>> print(22/7)
3.142857142857143
```

print() function

We can also pass more than one argument to the print() function. In such a case the arguments are separated by commas.

```
>>> print("hello","how are you")
hello how are you
>>> print("10+30",10+30)
10+30 40
>>> print("4+3/2*7=",4+3/2*7)
4+3/2*7= 14.5
>>> print("2 plus 2 is ",2+2," ","2 minus 2 is ",2-2)
2 plus 2 is 4 , 2 minus 2 is 0
```

print() function

We can also pass more than one argument to the print() function. In such a case the arguments are separated by commas.

```
>>> print("hello","how are you")
hello how are you
>>> print("10+30",10+30)
10+30 40
>>> print("4+3/2*7=",4+3/2*7)
4+3/2*7= 14.5
>>> print("2 plus 2 is ",2+2,"","2 minus 2 is ",2-2)
2 plus 2 is 4 , 2 minus 2 is 0
```

Consecutive arguments are separated by commas(,) in print() function.

print() function

We see that space is the default separator of values in the output. If we wish, we can also specify some other string as the separator using the sep argument of print() function.

```
>>> print(10,20,30)
10 20 30
>>> print(10,20,30,sep='@')
10@20@30
>>> print(10,20,30,sep=',')
10,20,30
>>> print(10,20,30,sep='\n')
10
20
30
>>> print(10,20,30,sep='\t')
10      20      30
```

Default separator: space

Separator: @

Separator: ,

Separator: \n

Separator: \t

print() function

If we use sep argument to specify the separator, then it must be specified after all the values to be printed. Otherwise, the interpreter shows a syntax error. An example is given:

```
>>>print(10,20,30, sep="*",50)
```

Syntax Error : positional argument follows keyword argument

print() function

print() appends a new line character at the end of the line unless you give your own end argument.

```
>>>print("Good Morning")  
>>>print("How are you")
```

OUTPUT :

```
=== RESTART: C:/Users/Admin DPS/AppData/Local/Programs/Python/Python37/p3.py =  
Good Morning  
how are you?  
>>> |
```

Programs

1. Write a program to input the radius of a circle and then display the area and the circumference of the circle.
2. Write a program to input the marks (out of 100) of a student in 5 subjects and then display the total marks along with the percentage.
3. Write a program to input a number n and then display n^2 , n^3 and n^4
4. Write a program to input the values of principal, rate and time and then print the value of simple interest. $SI = (P \cdot R \cdot T) / 100$
5. Write a program to input the temperature in Celsius and convert to Fahrenheit using the formula:

$$F = C \cdot 9/5 + 32$$

Programs

6. Write a program to calculate the BMI (Body Mass Index) of a person. Input the weight of the person in kg and height in meters. BMI is calculated as kg/m^2

7. Find outputs of the following commands in Python:

(i) `print("Hello")` (ii) `print(2+3,34-67)` (iii) `print("2+3",2+3)`

(iv) `print ('2+3',2+3,sep="=")` (v) `print(2**3,3**2,sep='*')`

(vi) `print(2**3, 2**-3, -2**3, (-2)**3, sep=" and ")`

8. Find error(s), if any, in the following statements:

(i) `print("three')`

(ii) `Print(3)`

(iii) `print(44'+56)`

(iv) `print(3,2 sep=': ')`

(v) `print "wisdom"`

(vi) `print['33+44']`

(vii) `PRINT("15 August 1947")`

Simplicity of Instructions:

Sample Program to calculate the area of a circle:

```
a = float(input("Enter radius"))  
b = 3.14 * a * a  
print(" Area : ", b)
```

Program with more meaningful variable names:


```
r = float(input("Enter radius"))  
area = 3.14 * r * r  
print(" Area : ", area)
```

Tips to be followed while writing a program

Tips to be used are as follows:

- Avoid clever instructions.
- One instruction per task
- Use standards – Every language has its standards, follow them
- Use appropriate comments

```
bill = float(input("Enter the bill amount"))  
total = bill + (10/100) * bill - (5/100) * bill  
print("Total = ", total)
```



```
bill = float(input("Enter the bill amount"))  
tax = (10/100) * bill  
discount = (5/100) * bill  
total = bill + tax - discount  
print("Total = ", total)
```

Clarity and Simplicity of Expressions

An expression in a Python program is a sequence of operators and operands to perform an arithmetic or logical computation.

Some examples of valid expressions:

- Assigning a value to a variable
- Arithmetic calculations using one or more variables
- Comparing two values

Be careful while writing such expressions

- Avoid expressions which may give ambiguous results
- Avoid complex expressions

Simplicity of Instructions:

- Programs are not written purely for the purpose of executing it once by the computer, one must write clear instructions so that whosoever reads the program later (even you yourself!!) should be able to understand. It is common for the programmers not to understand the logic of their own programs after some time.
- Maintenance and Modification of such programs would be very difficult.
- Names of variable(s) used in a program must represent or identify the purpose of their use, they must be simple, short and meaningful.

Error in a Program

- **Error/Bug:** Any malfunction, which either “stops the normal execution of the program” OR “program execution with wrong results” is known as an error/bug in the program. Removal of Bug from a program is known as **debugging**.

There are different types of errors in a Python program:

Syntax error: Like other programming languages, Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there. For example, parentheses must be in pairs, so the expression $(10 + 12)$ is syntactically correct, whereas $(7 + 11$ is not due to absence of right parenthesis. Such errors need to be removed before the execution of the program

Errors in a Program

A **logical error** is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program. Since the program interprets successfully even when logical errors are present in it, it is sometimes difficult to identify these errors. The only evidence to the existence of logical errors is the wrong output.

For example, if we wish to find the average of two numbers 10 and 12 and we write the code as $10 + 12/2$, it would run successfully and produce the result 16. Surely, 16 is not the average of 10 and 12. The correct code to find the average should have been $(10 + 12)/2$ to give the correct output as 11.

Errors in a Program

A **runtime error** causes abnormal termination of program while it is executing. Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing. For example, we have a statement having division operation in the program. By mistake, if the denominator entered is zero then it will give a runtime error like “**division by zero**”.

```
A=int(input("Numerator"))  
B=int(input("Denominator"))  
C=A/B;  
print(C)
```

The program shown above will throw an Exception (**ZeroDivisionError**), if user enters **0** for the variable **B**.

The program shown above will throw an Exception (**ValueError**), if user enters a non numeric value for example **Hello** for any of the variables A or B.

THANK YOU!

DEPARTMENT OF COMPUTER SCIENCE