History of Python:

- It was developed by Guido Van Rossum in the late 80's and early 90's at National Research Institute for mathematics and computer science in the Netherlands.
- It is a successor to ABC language.
- Its version 0.9.0 was released in 1991 with features functions, data types.
- Python version 1.0 was released in 1994 with functional programming tools such as lambda, map, and reduce.
- 2.0 released in 2000 with features List comprehension, Unicode and garbage collector.
- 3.0 released in 2008 by removing duplicate programming constructs and modules.
- Latest python version is 3.7.4 (as on 15/8/2019)

Note:

- Surprisingly Python is older than Java, java script and R.
- Why is it called **Python**? When he began implementing **Python**, Guido van Rossum was also reading the published scripts from "Monty **Python's** Flying Circus", a BBC comedy series from the 1970s. Van Rossum thought he needed a **name** that was short, unique, and slightly mysterious, so he decided to call the language **Python**.

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.

Literal Constants:

The Literal Constant can be 10,10.5,'A' and "hello". These can be directly used in the program. Constant is a fixed value that does not change. Let us see Number and string constants in C.

Numbers:

It is a numerical value. We can use four types' numbers in python. They are Binary numbers, Integral numbers, floating point numbers and complex numbers.

Binary Numbers made up of with 0's and 1's.In python every Binary number begins with either 0b or 0B (zero b).Eg:0b1110 , 0B1111

Integral numbers can be whole numbers or integer number or octal number or Hexadecimal Number.

a) **Integer numbers** like 10, 34567, 123456778889999907788,-9876.Integer can be small, long, positive or negative. In *python* integer numbers are identified by word *int. Integer number never have a decimal point.*
b) ***Octal numbers*** *start with either 0o or 0O(zero capital O).Octal number is made of digits(0-7).Eg:0o57 is a valid octal number,0o876 is not a valid octal number.* ***Octal numbers identified by oct word****.*
c) ***Hexa decimal numbers*** *start with either 0x or 0X.It is made of digits(0-9) and letters(A-Z or a-z).Eg:oxface oXbeaf,ox975b are valid but 0xbeer,ox5vce are not valid.* ***Hexa decimal numbers identified by hex word.***

Floating point numbers are like 5.678,91.45e-2,91.4E-2(it equals to $91.4*10^{-2}$).**In *python* Floating numbers are identified by word *float*** *which has a decimal point.*

Complex numbers have real ,imaginary part like a+bj
Eg:1+5j ,7-8J
Complex numbers are identified by the word by ***complex.***

Note: **Commas are not allowed in numerical.** Eg: 3,567 -8,90876 are invalid numbers.

- No size limit for integer number that can be represented in python but floating point number have a range of 10^{-323} to 10^{308} with 15 digits precision.
- Issues with floating point numbers:
  Arithmetic Overflow Problem: It occurs when a result is too large in size to be represented. Eg: 2.7e200*4.3e200 you will get infinity.
  Arithmetic Underflow Problem: It occurs when a result is too small in size to be represented. Eg: 2.7e-200/4.3e200 you will get infinity.
  Loss of precision: Any floating number has a fixed range and precision, the result is just approximation of actual or true value. Slight loss in accuracy is not a concern in practically but in scientific computing ,it is an issue.

Strings

A *string* is a group of characters.

• *Using Single Quotes ('):* For example, a string can be written as 'HELLO'.

• *Using Double Quotes ("):* Strings in double quotes are exactly same as those in single quotes. Therefore, 'HELLO' is same as "HELLO".

• *Using Triple Quotes (''' '''):* You can specify multi-line strings using triple quotes. You can use as many single quotes and double quotes as you want in a string within triple quotes.

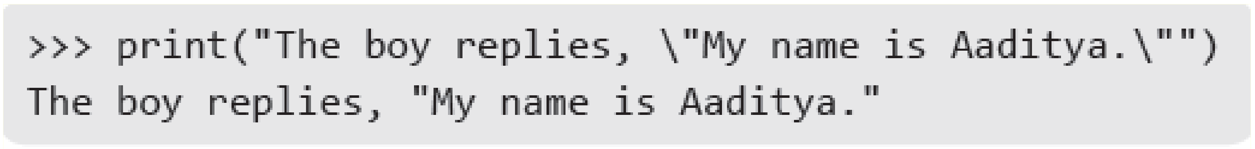
Examples:

`>>> 'Hello'`
`'Hello'`
	`>>> "HELLO"`
	`'HELLO'`
		`>>> '''HELLO'''`
		`'HELLO'`

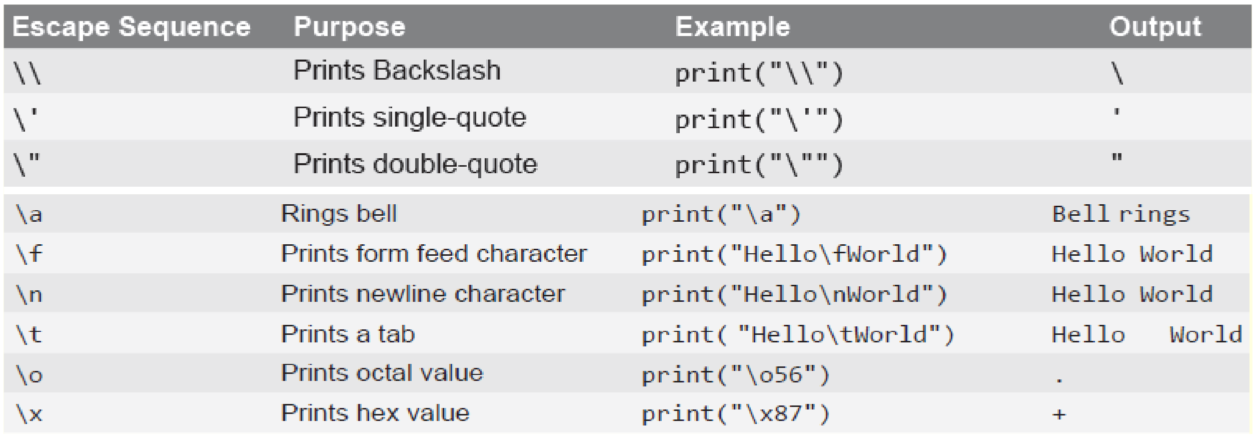
Escape Sequences

Some characters (like ", \) cannot be directly included in a string. Such characters must be escaped by placing a backslash before them.

Example:



```
>>> print("The boy replies, \"My name is Aaditya.\"")
The boy replies, "My name is Aaditya."
```



Escape Sequence	Purpose	Example	Output
`\\`	Prints Backslash	`print("\\")`	`\`
`\'`	Prints single-quote	`print("\'")`	`'`
`\"`	Prints double-quote	`print("\"")`	`"`
`\a`	Rings bell	`print("\a")`	Bell rings
`\f`	Prints form feed character	`print("Hello\fWorld")`	`Hello World`
`\n`	Prints newline character	`print("Hello\nWorld")`	`Hello World`
`\t`	Prints a tab	`print( "Hello\tWorld")`	`Hello    World`
`\o`	Prints octal value	`print("\o56")`	`.`
`\x`	Prints hex value	`print("\x87")`	`+`

Example 2:print(“Welcome to \\Vmeg\\”)

Output:Welocme to \Vmeg\

Example 3: print(“Welcome to \’Vmeg\’ ”)

Output: Welocme to ‘Vmeg’

Example 4: print(“Welcome to\b Vmeg ”)

Output:Welcom to Vmeg

Example 5: print(“Welcome to\r Vmeg ”)

Output:

Use of Triple Quotes:

The triple quotes can be single or double i.e. ‘’’ or “””

1) Triple quotes can be used for multi line string literals
   Example: print(‘’’ Hello
   How are
   You?’’’)
   Output:Hello

```
             How are
              You?
Example: print(“””Hello
                              How are
                               You?”””)
Output:Hello
             How are
              You?
```

2) Triple quotes can be used for printing a single quotes or double quotes

```
Example: print(‘’’ Hello
                              ‘How are’
                               You?’’’)
Output:Hello
             ‘How are’
              You?

Example: print(‘’’Hello
                              “How are”
                               You?’’’)
Output:Hello
             “How are”
              You?
```

String Formatting:

The format function can be used to control the display of strings.

Format(string, format specifier)

The strings can be left,right,central justification in a specified width.

Ex:format(“HELLO”,”<10”) –left justification

o/p:

H	E	L	L	O					

Rightt justification(>)

Ex:format(“HELLO”,”>10”)

					H	E	L	L	O

Central Justification (^)

Ex:format("HELLO","^10")

		H	E	L	L	O			

By using format function, we can fill the width with other characters also

Ex:print("Hello",format("*","*<10"))

o/p: hello **********

Identifiers:

An identifier is a name used to identify a variable, function, class, module, or object.
In **Python**, an identifier is like a noun in **English**.
Identifier helps in differentiating one entity from the other. For example, name and age which speak of two different aspects are called identifiers.
Python is a case-sensitive programming language. Means, Age and age are two different **identifiers** in **Python**.
Let us consider an example:

```
Name = "VMEG"

name = "VMEG"
```

Here the identifiers Name and name are different, because of the difference in their case.
Below are the rules for writing identifiers in **Python**:
1. Identifiers can be a combination of lowercase letters **(a to z)** or uppercase letters **(A to Z)** or digits **(0 to 9)** or an underscore **(_)**.
Eg: myClass, **var_1**, **print_this_to_screen**, **_number** are valid **Python** identifiers.
2. An identifier can't start with a digit.
Eg: 1_variable is invalid, but **variable_1** is perfectly fine.
3.Keywords cannot be used as identifiers. (**Keywords** are reserved words in **Python** which have a special meaning).
Eg: Some of the **keywords** are **def**, **and**, **not**, **for**, **while**, **if**, **else** and so on.

4.Special symbols like !, @, #, $, % ,space or gap etc. are not allowed in identifiers. Only one special symbol underscore (_) is allowed.
Eg: company#name, **$name**, **email@id** ,simple interst are invalid **Python** identifiers.
5.Identifiers can be of any length.

Variables:

Variable is a name used to store a constant. Like other languages like c, c++, java, no need to declare or write type of a variable.
You can store any piece of information in a variable. Variables are nothing but just parts of your computer's memory where information is stored. *To be identified easily, each variable is given an appropriate name.*
***Examples of valid variable names* are** a, b, c,sum, __my_var, num1, r, var_20, first, etc.
***Examples of invalid variable names* are** 1num, my-var, %check, Basic Sal, H#R&A, etc.

UNDERSTANDING VARIABLES:

In **Python** a variable is a reserved memory location used to **store** values.

For example in the below code snippet, age and city are variables which store their respective values.

```
age = 21
city = "Tokyo"
```

Usually in programming languages like **C**, **C++** and **Java**, we need to declare **variables** along with their **types** before using them. **Python** being a dynamically typed language, there is no need to declare variables or declare their types before using them.

Python has no command for declaring a variable. A variable is created the moment, a value is assigned to it.

The **equal-to (=)** operator is used to assign value to a variable.

Note: **Operators** are special **symbols** used in programming languages that represent particular actions. = is called the **assignment operator**.

For example :

```
marks = 100 # Here marks is the variable and 100 is the value assigned to it.
```

Note: As a good programming practice, we should use meaningful names for variables. For example, if we want to store the **age** of a person, we should not name the variable as x, or y or a, or i. Instead we should name the variable as age.

Python interpreter automatically determines the type of the data based on the data assigned to it.

Note: Henceforth, where ever we say Python does this or that, it should be understood that it is the Python interpreter (the execution engine) which we are referring to.

In Python, we can even change the type of the variable after they have been set once (or initialized with some other type). This can be done by assigning a value of different type to the variable.
A variable in **Python** can hold any type of value
like 12 (integer), 560.09 (float), "Vmeg" (string) etc.
Associating a **value** with a **variable** using the assignment operator (=) is called as Binding.

In **Python**, **assignment** create references, not copies.
We cannot use **Python** variables without assigning any value.
If we try to use a variable without assigning any value then, Python interpreter shows `error` like "`name is not defined`".

Let us assign the variables value1 with **99**, value2 with "**Hello Python**" and value3 with "**Hello World**", and then print the result.

Sample Input and Output:

```
99
Hello Python
Hello World
```

```
#python program
Value1=99
Value2=”Hello Python”
Value3=”Hello World”
print(Value1)
print(Value2)
print(Value3)
```

ASSIGNMENT OF VARIABLES:

Python variables do not need explicit declaration to reserve memory space.

The declaration happens automatically when you assign a value to a variable.

The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand (or expression) to the right of the = operator is the value stored in the variable.

Let us consider the below example:

```
counter = 100 # An integer assignment
print(counter) # Prints 100
miles = 1000.50 # A floating point assignment
print(miles) # Prints 1000.5
name = "John" # A string assignment
print(name) # Prints John
```

In the above example **100**, **1000.50**, and "**John**" are the values assigned to the variables **counter**, **miles**, and **name** respectively.

The above example program produces the result as:

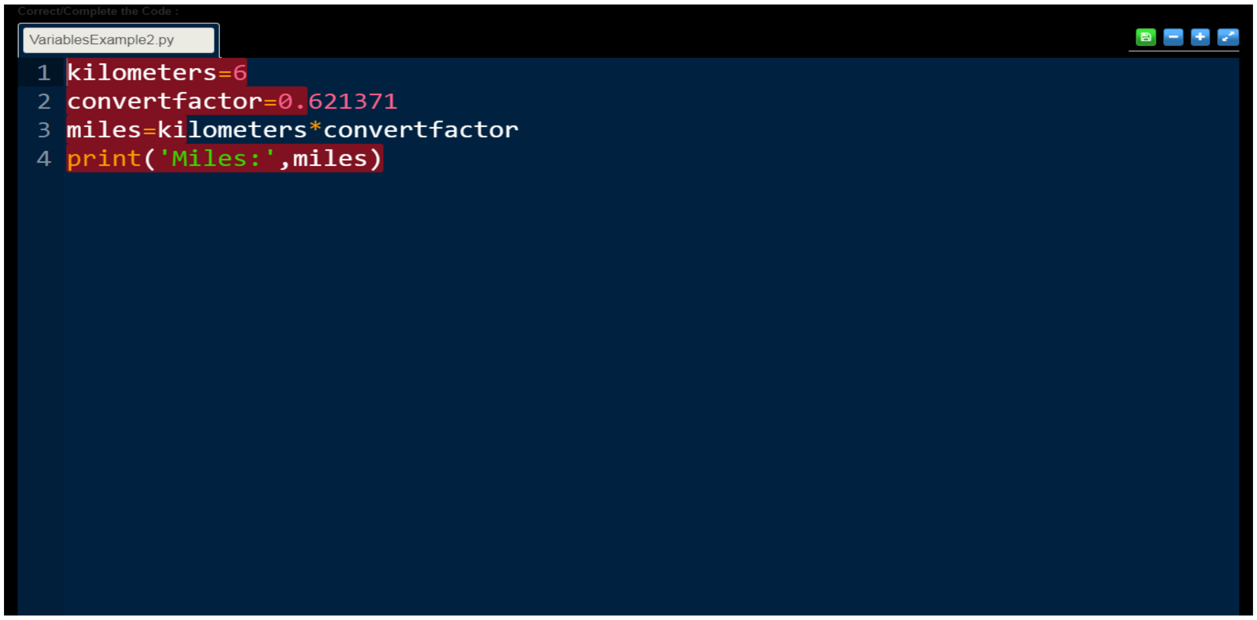
```
100

1000.5

John
```

Write a program which does the following:

1. Create a variable kilometers and assign 6 to it.
2. Create a variable convertfactor and assign 0.621371 to it.
3. Calculate the product of kilometers and convertfactor and assign it to a variable miles
4. Print the values of miles as "Miles:" miles

Expected Output:

```
Miles:·3.7282260000000003
```



```python
kilometers=6
convertfactor=0.621371
miles=kilometers*convertfactor
print('Miles:',miles)
```

UNDERSTANDING MULTIPLE ASSIGNMENTS

Python allows you to assign a single value to several variables **simultaneously**.

Let us consider an example:

```
number1 = number2 = number3 = 100
```

Here an **integer object** is created with the value 100, and all the three variables are references to the same memory location. This is called chained assignment.

We can also assign multiple objects to multiple variables, this is called multiple assignment.

Let us consider the below example:

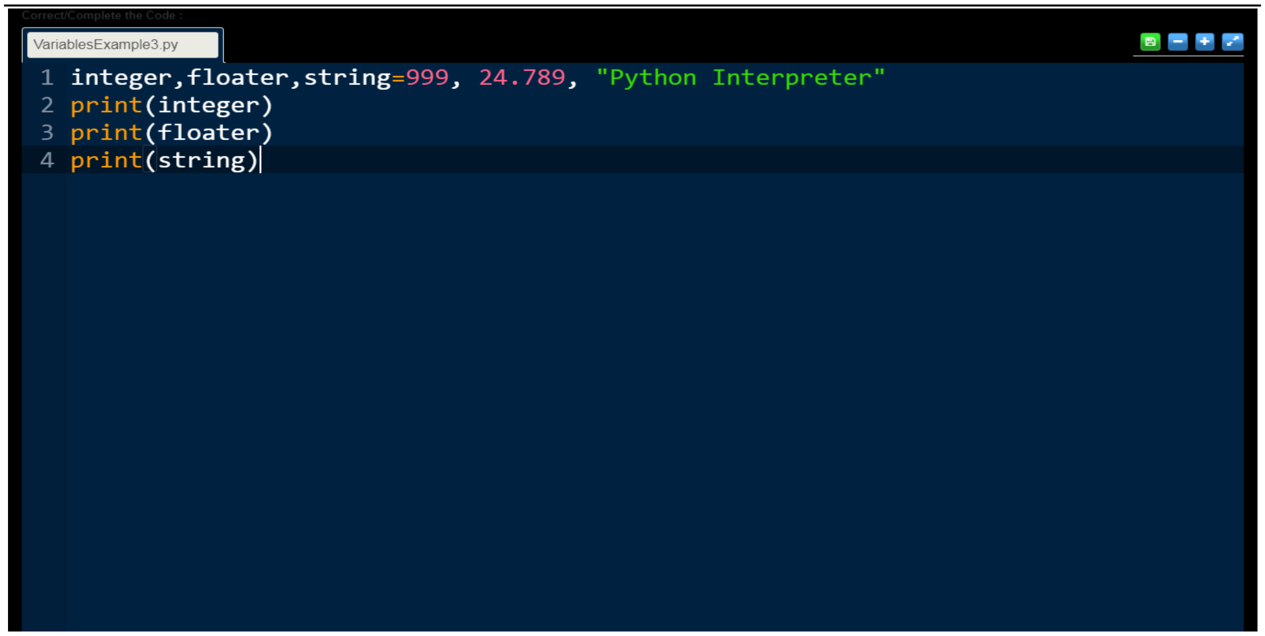
```
value1, value2, value3 = 1, 2.5, "Ram"
```

Here the **integer object** with value 1 is assigned to variable **value1**, the **float object** with value 2.5 is assigned to variable **value2** and the **string object** with the value "Ram" is assigned to the variable **value3**.

Write a program to assign the integer **999**, float **24.789** and string "**Python Interpreter**" to three variables using multiple assignment and print them individually.

Sample Input and Output:

```
999

24.789

Python Interpreter
```



```python
integer,floater,string=999, 24.789, "Python Interpreter"
print(integer)
print(floater)
print(string)
```

CHAINED ASSIGNMENT:

n **Python**, assignment statements do not return a value. Chained assignment is recognised and supported as a special case of the assignment statement.

A statement like a = b = c = x is called a chained assignment in which the value of **x** is assigned to multiple variables **a**, **b**, and **c**.

Let us consider a simple example:

```
a = b = c = d = 10
print(a) # will print result as 10
print(b) # will print result as 10
print(c) # will print result as 10
print(d) # will print result as 10
```

Here, we initialising the **a**, **b**, **c**, **d** variables with value 10.

Write a program to assign a user given value to a, b, c variables.

At the time of execution, the program should print message on the console as:

```
Enter a value:
```

For example, if the user gives the input as:

```
Enter a value: 100
```

then the program's result is as follows :

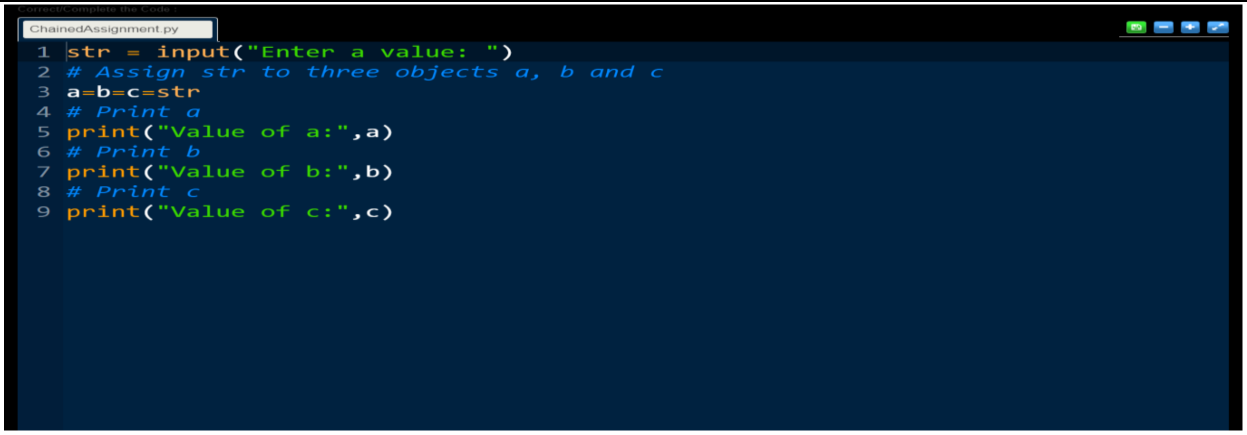
```
Value of a: 100
Value of b: 100
Value of c: 100
```

Note: Let us assume that input() is used to read values given by the user. We will learn about input() later sections.
Here,

```
a = b = c = str
```

Sample Input and Output:

```
Enter a value: Hello Python
Value of a: Hello Python
Value of b: Hello Python
Value of c: Hello Python
```



```
str = input("Enter a value: ")
# Assign str to three objects a, b and c
a=b=c=str
# Print a
print("Value of a:",a)
# Print b
print("Value of b:",b)
# Print c
print("Value of c:",c)
```

Data Types:

Variables can hold values of different data types. Python is a dynamically typed language hence we need not define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

Python enables us to check the type of the variable used in the program. Python provides us the **type()** function which returns the type of the variable passed.

Consider the following example to define the values of different data types and checking its type.

```
A=10
b="Hi Python"
c = 10.5
print(type(a));
print(type(b));
print(type(c));
```

Output:

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.

Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. Numbers
2. String
3. List
4. Tuple
5. Dictionary

Python Keywords

Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter. Each keyword have a special meaning and a specific operation. These keywords can't be used as variable. Following is the List of Python Keywords.

True	False	None	and	as
Asset	def	Class	continue	break
Else	finally	Elif	del	except
Global	for	If	from	import
Raise	try	Or	return	pass
Nonlocal	in	Not	is	lambda

Comments:

A computer program is a collection of instructions or statements.

A **Python** program is usually composed of multiple statements.
Each statement is composed of one or a combination of the following:

1. Comments

2. Whitespace characters
3. Tokens

In a computer program, a comment is used to mark a section of code as non-executable.

Comments are mainly used for two purposes:

1. To mark a section of source code as non-executable, so that the Python interpreter ignores it.
2. To provide remarks or an explanation on the working of the given section of code in plain English, so that a fellow programmer can read the comments and understand the code.

In **Python**, there are two types of comments:

1. single-line comment : It starts with # (also known as the **hash** or **pound** character) and the content following # till the end of that line is a comment.
2. Docstring comment : Content enclosed between tripple quotes, either ''' or """. (We will learn about it later).

INPUT AND OUTPUT STATEMENTS

In Python, to read the input from the user, we have an in-built function called input().

The syntax for input() function is :

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

here, the **optional** prompt string will be printed to the user output and then will wait for the user input. The prompt string is to used to tell the user what to input.

Reading string inputs

Let us consider the below example:

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

print("User Name:", name)
```

Sample input output of this program is as follows

```
Enter your Name: Anand Vihan

User Name: Anand Vihan
```

If the input is given as "**Jacob**", the result will be

User Name: Jacob
Eg:
Write a program to print your favorite place. At the time of execution, the program should print the message on the console as:

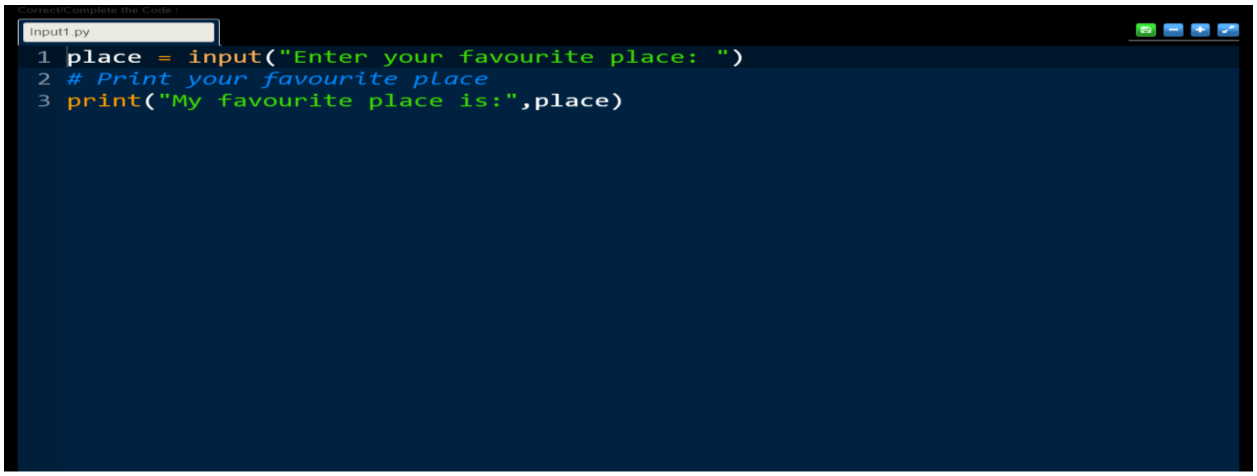
```
Enter your favourite place:
```

For example, if the user gives the input as:

```
Enter your favourite place: Hyderabad
```

then the program should print the result as:

```
My favourite place is: Hyderabad
```



```python
place = input("Enter your favourite place: ")
# Print your favourite place
print("My favourite place is:",place)
```

UNDERSTANDING OUTPUT IN PYTHON:

We already discussed this function to print output on Screen.

It is **print()** function, which is a built-in function in Python. We can pass zero or more number of expressions separated by **commas(,)** to **print()** function.

The **print()** function converts those expressions into Strings and write the result to standard output which then displays the result on Screen.

Let us consider an example of **print()** with multiple values with comma separator.

```
print("Hi", "Hello", "Python") # will print output as follows
Hi Hello Python
```

Let us discuss another example:

```
a = 10
b = 50
print("A value is", a, "and", "B value is", b) # will print output as follows
A value is 10 and B value is 50
```

Here we have assigned values 10 and 50 to a and b respectively.

The above **print()** statement consists of both strings and integer values.

Write a program to print your favourite programming language.

At the time of execution, the program should print the message on the console as:

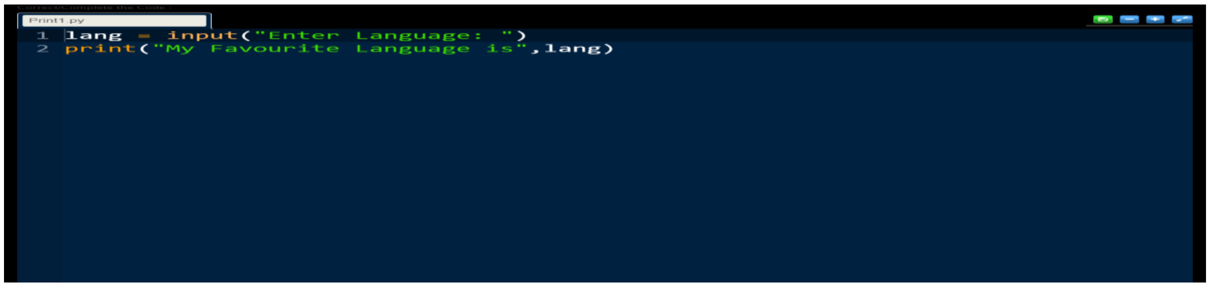
```
Enter Language:
```

For example, if the user gives the input as:

```
Enter Language: Python
```

then the program should print the result as:

```
My Favourite Language is Python
```



```python
lang = input("Enter Language: ")
print("My Favourite Language is",lang)
```

Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Operators are the pillars of a program on which the logic is built in a particular programming language. Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Unary operators
- Bitwise Operators
- Membership Operators
- Identity Operators

1.Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations between two operands. It includes +(addition), - (subtraction), *(multiplication), /(divide), %(reminder), //(floor division), and exponent (**).In the table a=100,b=200

Operator	Description	Example	Output
+	Addition: Adds the operands	`>>> print(a + b)`	300
-	Subtraction: Subtracts operand on the right from the operand on the left of the operator	`>>> print(a – b)`	-100
*	Multiplication: Multiplies the operands	`>>> print(a * b)`	20000
/	Division: Divides operand on the left side of the operator with the operand on its right. The division operator returns the quotient.	`>>> print(b / a)`	2.0
%	Modulus: Divides operand on the left side of the operator with the operand on its right. The modulus operator returns the remainder.	`>>> print(b % a)`	0

//	Floor Division: Divides the operands and returns the quotient. It also removes the digits after the decimal point. If one of the operands is negative, the result is floored (i.e.,rounded away from zero towards negative infinity).	`>>> print(12//5)`
		`>>> print( 12.0//5.0)`
		`>>> print(-19//5)`
		`>>> print(-20.0//3)`
			2
			2.0
			
			-4
			-7.0

**	Exponent: Performs exponential calculation, that is, raises operand on the right side to the operand on the left of the operator.	`>>> print(a**b)`	100^{200}

2.Comparison Operators:

Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly. The comparison operators are described in the following table a=5,b=6

Operator	Description	Example	Output
==	Returns True if the two values are exactly equal.	`>>> print(a == b)`	False
!=	Returns True if the two values are not equal.	`>>> print(a != b)`	True
>	Returns True if the value at the operand on the left side of the operator is greater than the value on its right side.	`>>> print(a > b)`	False
<	Returns True if the value at the operand on the right side of the operator is greater than the value on its left side.	`>>> print(a < b)`	True
>=	Returns True if the value at the operand on the left side of the operator is either greater than or equal to the value on its right side.	`>>> print(a >= b)`	False
<=	Returns True if the value at the operand on the right side of the operator is either greater than or equal to the value on its left side.	`>>> print(a <= b)`	True

3. Assignment operators

The assignment operators are used to assign the value of the right expression to the left operand. The assignment operators are described in the following table.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b

	will assign 4//3 = 1 to a.

4. Logical Operators:

The logical operators are used primarily in the expression evaluation to make a decision. Python supports the following logical operators.

Operator	Description
And	If both the expression are true, then the condition will be true. If a and b are the two expressions, a → true, b → true => a and b → true.
Or	If one of the expressions is true, then the condition will be true. If a and b are the two expressions, a → true, b → false => a or b → true.
Not	If an expression **a** is true then not (a) will be false and vice versa.

5. Unary Operators:

Unary operators act on single operands. Python supports unary minus operator. Unary minus operator is strikingly different from the arithmetic operator that operates on two operands and subtracts the second operand from the first operand. When an operand is preceded by a minus sign, the unary operator negates its value.

For example, if a number is positive, it becomes negative when preceded with a unary minus operator. Similarly, if the number is negative, it becomes positive after applying the unary minus operator. Consider the given example.

b = 10 a = -(b)

The result of this expression, is a = -10, because variable b has a positive value. After applying unary minus operator (-) on the operand b, the value becomes -10, which indicates it as a negative value.

6. Bitwise Operators:

The bitwise operators perform bit by bit operation on the values of the two operands.

Operator	Description
& (binary and)	If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied.
| (binary or)	The resulting bit will be 0 if both the bits are zero otherwise the resulting bit will be 1.
^ (binary xor)	The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0.
~ (negation)	It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa.
<< (left shift)	The left operand value is moved left by the number of bits present in the right operand.
>> (right shift)	The left operand is moved right by the number of bits present in the right operand.

For example,

```
if a = 7;
   b = 6;
then, binary (a) = 0111
    binary (b) = 0011
  hence, a & b = 0011 (3 in decimal number)
      a | b = 0111 (7 in decimal number)
             a ^ b = 0100  (4 in decimal number)
       ~ a = 1000
a<<2=11100(28 in decimal number)
```

10. a>>2=0001(1 in decimal number)
Similarly 5 binary equivalent is 0101 and 3 is 0011

7. **Membership Operators:**

Python supports two types of membership operators–in and not in. These operators, test for membership in a sequence such as strings, lists, or tuples.
in Operator: The operator returns true if a variable is found in the specified sequence and false otherwise.
For example,a=6
nums=[1,4,3,7,6]
a in nums returns True, if a is a member of nums.
not in Operator: The operator returns true if a variable is not found in the specified sequence and false otherwise.
For example, a=99
Nums=[1,3,5,7,2,0]
a not in nums returns True, if a is not a member of nums otherwise returns False.

8. **Identity Operators**

is Operator: Returns true if operands or values on both sides of the operator point to the same object (*Memory location*)and False otherwise. For example, if a is b returns True , if id(a) is same as id(b) otherwise returns False. id(a) gives memory location of a
is not Operator: Returns true if operands or values on both sides of the operator does not point to the same object(*Memory location*) and False otherwise. For example, if a is not b returns 1, if id(a) is not same as id(b).

Operator Precedence

The precedence of the operators is important to find out since it enables us to know which operator should be evaluated first. The precedence table of the operators in python is given below.

Operator	**Description**
**	The exponent operator is given priority over all the others used in the expression.

~ + -	The negation, unary plus and minus.
* / % //	The multiplication, divide, modules, reminder, and floor division.
+ -	Binary plus and minus
>> <<	Left shift and right shift
&	Binary and.
^ |	Binary xor and or
<= < > >=	Comparison operators (less then, less then equal to, greater then, greater then equal to).
<> == !=	Equality operators.
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators