Unit-I

Introduction to Python Programming

What is Python:----------

- Python is a high-level programming language created by Guido Van Rossum.
- It was first released in 1991. Today python interpreters are available for many Operating Systems including Windows and Linux.
- Python programmers are often called Pythonists or Pythonistas.
- Python is open source, which means it's available free of cost.
- Python is simple and so easy to learn.
- Python is versatile and can be used to create many different things.
- Python has powerful development libraries, including AI, ML, etc.
- Python is much in demand and ensures a high salary
- Python is Interpreted language.
- Python is Multi paradigm programming language.

Reason for Popularity:----------

- There are several reasons for Python's popularity. These include:
  a. Free
  b. Software Quality
  c. Developer productivity
  d. Program portability
  e. Support libraries
  f. Component integration
  g. Enjoyment

Where is Python:----------

- Python is used for multiple purposes. These includes:-
  a. System Programming
  b. Building GUI applications//tkinter
  c. Internet scripting
  d. Component integration
  e. Database programming
  f. Rapid prototyping
  g. Numeric and Scientific programming
  h. Game programming
  i. Robotics programming

Who uses Python Today:----------

- Many organizations use Python for varied purpose. These include:-
  a. Google – In web search system
  b. Youtube – Video sharing service
  c. Bit-torrent - Peer to peer file sharing system
  d. Intel, HP, IBM, Seagate, Qualcomm- Hardware testing
  e. Industrial Light and Magic- Movie animation
  f. NASA, Fermilab - Scientific programming
  g. And many more……

Why to Learn Python:----------

- There are many reasons to learn Python, which make it popular programming language:-
  a. Simplicity and Readability:- Python has a very simple and readable syntax, making it easy for beginners to learn . We can write and understand code quickly.
  b. Versatility:- Python is used in many fields, such as web development, data analysis, artificial intelligence, machine learning, scientific computing, and automation. This means we can work on various projects using just one language.
  c. Rich Libraries and frameworks:- Python has many libraries and frameworks like- NumPy, Pandas, Django that make development easier and faster. We can use them to handle complex tasks easily.
  d. Cross-Platform Compatibility:- Python is cross-platform meaning we can run it on windows, macOS, and Linux without major changes.
  e. Career Opportunities:- There is a high demand for Python skills, especially in Data Science and Machine Learning fields. Learning this language can give us advantage in job opportunities.

Advantages of Python Programming Language:----------

1. Easy to read, learn and code:- Python is a high-level language and its syntax is very simple. It does not need any semicolons or braces and looks like English. Thus, it is beginner-friendly.

   Due to its simplicity, its maintenance cost is less.

2. Dynamic Typing:- In Python, there is no need for the declaration of variables. The data type of the variable gets assigned automatically during runtime, facilitating dynamic coding.

3. Free, Open Source:- It is free and also has an open-source licence. This means the source code is available to the public for free and one can do modifications to the original code. This modified code can be distributed with no restrictions.

4. Portable:- Python is also platform-independent. That is, if you write the code on one of the Windows, Mac, or Linux operating systems, then you can run the same code on the other OS with no need for any changes.This is called Write Once Run Anywhere (WORA). However, you should be careful while you add system dependent features.

5 . Extensive Third-Party Libraries:- Python comes with a wide range of libraries like NumPy, Pandas, Tkinter, Django, etc.The python package installer (PIP) helps you install these libraries in your interpreter/ IDLE. These libraries have different modules/ packages. These modules contain different inbuilt functions and algorithms. Using these make the coding process easier and makes it look simple.

6. Wide Range of Applications:- Python has many applications like web development, making desktop GUIs, app development, artificial intelligence, data science, etc. It has become a preferred language by the professionals in many areas like engineering, mathematics and science.

7. Extensible and Integrable to Other Programming Languages:-In addition to having libraries like CPython and Jython, it can extend to other languages like C, C++. This feature helps while building projects. It can also integrate with C, C++, and Java, helping in cross-platform development and also in using the strong features of each language. This makes Python a powerful language.

8. **Interpreted Language:-** Python is an interpreted language. The code gets executed line by line till an error is encountered if any.If an error occurs at a line, it stops execution and gives that error to the console. This leads to an easier and step-by-step debugging process.

9. **Functional, Object-Oriented, and Procedural:-**It is a procedural, functional, and object-oriented language. Procedural means the code gets executed in the top to bottom fashion. A functional language works based on functions, rather than just statements. A function is a collection of codes that takes input and gives output.Information is treated as a real-world object with properties and behaviors in object-orientated language.

10. **Involvement in Large Projects**:- It is used for implementation in big projects and software like YouTube, Google, Yahoo. It is also a preferred language by many companies in various fields like education, medical, research, etc.

11. **Memory Management:-**Python also excels in managing its memory by using a separate library for this purpose. It uses a private heap to hold all the objects and data structures. And a built-in memory manager handles this heap. This property of Python makes it stand out from the other programming languages.

12. **Improved Productivity:-**The fact that the syntax of python is very easy and short, allows more productivity. Developers can focus more on the algorithm, rather than on coding.

13. **Vast Community:-**Python has an active and big community that helps in doing continuous additions to it. It also allows the availability of the information on Python at ease to the developer.

Disadvantages of Python----------

1. It's Simple Nature

Are you wondering how this feature under advantage also is coming as a disadvantage?
Then the answer is Yes! Its simplicity is making it hard for a programmer to adjust to the syntax of the other programming languages.
For example, a coder might forget to declare a variable in C falling into an error.

2. Slow Speed and Memory Inefficient

The interpreter in Python executed the code line by line, which increases the overall time. The dynamic typing feature also decreases the speed. This is because it has to do extra work during runtime. It requires a large amount of memory too!

3. Weak Mobile Computation

Python has many applications on the server-side. But it is hardly seen on the client-side or in mobile applications. The main reasons for this are:

- It occupies a lot of memory.
- This makes the process slow.
- It also does not facilitate security

4. Poor Database Access

Python databases are weak compared to other technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity). This limits its use in high enterprises.

5. Runtime Errors due to dynamic typing

Because it's a dynamically typed language, you do not need to declare any variable and a variable storing an integer can store a string later. This might happen unnoticeably but leads to runtime errors while doing operations.

This is a restriction on the design of the Python programming language. Also, all errors show up only during runtime. So, it is difficult to test.

Formal and Natural Languages----------

Natural languages are the languages that people speak, such as English, Spanish, Korean, and Mandarin Chinese. They were not designed by people (although people try to impose some order on them); they evolved naturally.

Formal languages are languages that are designed by people for specific applications. For example, the notation that mathematicians use is a formal language that is particularly good at denoting relationships among numbers and symbols. Chemists use a formal language to represent the chemical structure of molecules. And most importantly:

Formal languages tend to have strict rules about syntax. For example, 3+3=6 is a syntactically correct mathematical statement, but 3=+6$ is not. H_2O is a syntactically correct chemical name, but $_2Zz$ is not.

Syntax rules come in two flavors, pertaining to **tokens** and structure. Tokens are the basic elements of the language, such as words, numbers, and chemical elements. One of the problems with 3=+6$ is that $ is not a legal token in mathematics (at least as far as we know). Similarly, $_2Zz$ is not legal because there is no element with the abbreviation Zz.

The second type of syntax rule pertains to the **structure** of a statement— that is, the way the tokens are arranged. The statement 3=+6$ is structurally illegal because you can't place a plus sign immediately after an equal sign. Similarly, molecular formulas have to have subscripts after the element name, not before.

When you read a sentence in English or a statement in a formal language, you have to figure out what the structure of the sentence is (although in a natural language you do this subconsciously). This process is called **parsing**.

For example, when you hear the sentence, "The other shoe fell", you understand that the other shoe is the subject and fell is the verb. Once you have parsed a sentence, you can figure out what it means, or the **semantics** of the sentence. Assuming that you know what a shoe is and what it means to fall, you will understand the general implication of this sentence.

Although formal and natural languages have many features in common — tokens, structure, syntax, and semantics — there are many differences:

Ambiguity

Natural languages are full of ambiguity, which people deal with by using contextual clues and other information. Formal languages are designed to be nearly or completely unambiguous, which means that any statement has exactly one meaning, regardless of context.

Redundancy

In order to make up for ambiguity and reduce misunderstandings, natural languages employ lots of redundancy. As a result, they are often verbose. Formal languages are less redundant and more concise.

Literalness

Formal languages mean exactly what they say. On the other hand, natural languages are full of idiom and metaphor. If someone says, "The other shoe fell", there is probably no shoe and nothing falling.

People who grow up speaking a natural language—that is, everyone—often have a hard time adjusting to formal languages. In some ways, the difference between natural and formal language is like the difference between poetry and prose, but more so.

Poetry

Words are used for their sounds as well as for their meaning, and the whole poem together creates an effect or emotional response. Ambiguity is not only common but often deliberate.

Prose

The literal meaning of words is more important, and the structure contributes more meaning. Prose is more amenable to analysis than poetry but still often ambiguous.

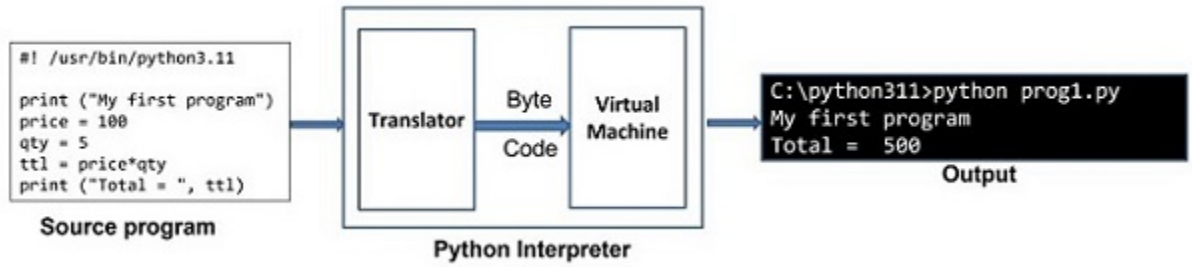
Program

The meaning of a computer program is unambiguous and literal, and can be understood entirely by analysis of the tokens and structure

Python Interpreter----------

Python is an interpreted language developed by Guido van Rossum in the year of 1991. As we all know Python is one of the most high-level languages used today because of its massive versatility and portable library & framework features. It is an interpreted language because it executes line-by-line instructions. There are actually two way to execute python code one is in Interactive mode and another thing is having Python prompts which is also called script mode. Python does not convert high level code into low level code as many other programming languages do rather it will scan the entire code into something called bytecode. every time when Python developer runs the code and start to execute the compilation part execute first and then it generate an byte code which get converted by PVM Python Virtual machine that understand the analogy and give the desired output.
Interpreted Languages: Perl, BASIC, Python, JavaScript, Ruby, PHP.
Compiled Languages: C, C++, C#, COBOL and CLEO.

Python Interpreter :- works in interactive and scripted mode. Python code is executed by one statement at a time method. Python interpreter has two components. The translator checks the statement for syntax. If found correct, it generates an intermediate byte code. There is a Python virtual machine which then converts the byte code in native binary and executes it. The following diagram illustrates the mechanism-



Python Interpreter - Interactive Mode

When launched from a command line terminal without any additional options, a Python prompt >>> appears and the Python interpreter works on the principle of REPL (Read, Evaluate, Print, Loop). Each command entered in front of the Python prompt is read, translated and executed. A typical interactive session is as follows.

```
>>> a=10
>>>print(a)
>>>10
>>>b=20
>>>print(b)
>>>20
>>>c=a+b
>>>print(c)
>>>30
```

To close interactive session type “quit()” or ctr+d on promt

Python Interpreter - Scripting Mode

Instead of entering and obtaining the result of one instruction at a time as in the interactive environment, it is possible to save a set of instructions in a text file, make sure that it has .py extension, and use the name as the command line parameter for Python command.

Save the following lines as prog.py, with the use of any text editor such as vim on Linux or Notepad on Windows

```
 a =10
 b=20
c=30
d=a+b+c
print(d)
```

Note that even though Python executes the entire script in one go, but internally it is still executed in line by line fashion.

Variables in Python----------

- A **variable** is a name that refers to a value stored in memory.

- Python variables **do not require explicit declaration**; they are created when a value is assigned to them.
- Python is **dynamically typed**, meaning you don't need to declare the variable type.

Syntax-
```
x = 5
y = "John"
print(x)
print(y)
```

- You can get the data type of a variable with the type( ) function.

Syntax-
```
x = 5
y = "John"
print(type(x))
print(type(y))
```

- String variables can be declared either by using single or double quotes:

Syntax-
```
x = "John"
# is the same as
x = 'John'
```

- Variable names are case-sensitive.

Syntax-
```
a = 4
A = "Sally"
#A will not overwrite a
```

- Python allows us to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- And you can assign the same value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

- The Python print() function is often used to output variables.

```
x = "Python is awesome"
print(x)
```

- In the print() function, you output multiple variables, separated by a comma:

```
x = "Python"
y = "is"
```

```
z = "awesome"
print(x, y, z)
```

- You can also use the + operator to output multiple variables:
```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

- The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

```
x = 5
y = "John"
print(x, y)
```

Variable Names----------

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)
- A variable name cannot be any of the Python Keywords.

Some legals variable-
```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

Comments in Python----------

Python comments start with the hash symbol # and continue to the end of the line.Comments in python are useful information that the developers provide to make the reader understand the source code. It explains the logic or a part of it used in the code. Comments in Python are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it. These are often cited as useful programming convention that does not take part in the output of the program but improves the readability of the whole program.
Comments in Python are identified with a hash symbol, #, and extend to the end of the line.

Types of comments in Python

A comment can be written on a single line, next to the corresponding line of code, or in a block of multiple lines. Here, we will try to understand examples of comment in Python one by one:

Single-line comment in Python

Python single-line comment starts with a hash symbol (#) with no white spaces and lasts till the end of the line. If the comment exceeds one line then put a hashtag on the next line and continue the comment. Python's single-line comments are proved useful for supplying short explanations for variables, function declarations, and expressions. See the following code snippet demonstrating single line comment:

Example 1:

Python allows comments at the start of lines, and Python will ignore the whole line.

```
# This is a comment

# Print "Hello Learners" to console

print("Hello Learners ")
```

Output

```
Hello Learners
```

Example 2:

Python also allows comments at the end of lines, ignoring the previous text.

```
a, b = 1, 3   # Declaring two integers

sum = a + b  # adding two integers

print(sum)  # displaying the output
```

Output

```
4
```

Multiline comment in Python

Use a hash (**#**) for each extra line to create a multiline comment. In fact, Python multiline comments are not supported by Python's syntax. Additionally, we can use Python multi-line comments by using multiline strings. It is a piece of text enclosed in a delimiter ("""") on each end of the comment. Again there should be no white space between delimiter (""""). They are useful when the comment text does not fit into one line; therefore need to span across lines. Python Multi-line comments or paragraphs serve as documentation for others reading your code. See the following code snippet demonstrating a multi-line comment:

Example 1:

In this example, we are using an extra # for each extra line to create a Python multiline comment.

```
# This is a comment
# This is second comment
# Print "Hello Learners" to console
print("Hello Learners ")
```

Output
Hello Learners

Example 2:
In this example, we are using three double quotes (“) at the start and end of the string without any space to create a Python multiline comment.

```
"""
This would be a multiline comment in Python that
spans several lines and describes Hello Learners.
A Computer Science portal for Learners. It contains
well written, well thought
and well-explained computer science
and programming articles,
quizzes and more.
…
"""
print("Hello Learners ")
```

Output
Hello Learners

Example 3:
In this example, we are using three single quotes (‘) at the start and end of the string without any space to create a Python multiline comment.

```
'''This is a
perfect example of
multi-line comments'''

print("Hello Learners ")
```

Output
Hello Learners

Keywords in Python----------

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging

break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.

is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents 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, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To return a list of values from a generator

python 3.10 introduced two new keywords: match and case

Operators in Python----------

Python divides the operators in the following groups:

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

Python Arithmetic Operators

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

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

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3

<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

Python Comparison Operators

Comparison operators are used to compare two values:

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

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example

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

Python Identity Operators

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

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence(list,tuple,string,dictionay) is presented in an object:

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

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

Operator Precedence

Operator	Description
()	Parentheses
**	Exponentiation
+x -x ~x	Unary plus, unary minus, and bitwise NOT

`* / // %`	Multiplication, division, floor division, and modulus
`+ -`	Addition and subtraction
`<< >>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`== != > >= < <= is is not in not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

If two operators have the same precedence, the expression is evaluated from left to right.

Precedence	Operators	Description	Associativity
1	**()**	Parentheses	Left to right
2	**	Exponentiation	Right to left

Precedence	Operators	Description	Associativity
3	**+x**, **-x**, **~x**	Positive, negative, bitwise NOT	Right to left
4	*****, **@**, **/**, **//**, **%**	Multiplication, matrix, division, floor division, remainder	Left to right
5	**+**, **–**	Addition and subtraction	Left to right
6	**<<**, **>>**	Shifts	Left to right
7	**&**	Bitwise AND	Left to right
8	**^**	Bitwise XOR	Left to right
9	**|**	Bitwise OR	Left to right
10	**in**, **not in**, **is**, **is not**, **<**, **<=**, **>**, **>=**, **!=**, **==**	Comparisons, membership tests, identity tests	Left to Right
11	**not x**	Boolean NOT	Right to left
12	**and**	Boolean AND	Left to right
13	**or**	Boolean OR	Left to right
14	**if-else**	Conditional expression	Right to left
15	**lambda**	Lambda expression	N/A
16	**:=**	Assignment expression (walrus operator)	Right to left

Expressions in Python----------

An expression is a combination of operators and operands that is interpreted to produce some other value. In any programming language, an expression is evaluated as per the precedence of its operators. So that if there is more than one operator in an expression, their precedence decides which operation will be performed first. We have many different types of expressions in Python. Let's discuss all types along with some exemplar codes :

1. Constant Expressions: These are the expressions that have constant values only.
Example: # Constant Expressions

```
x = 15 + 1.3

print(x)
```

Output:----16.3

2. Arithmetic Expressions: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value. The operators used in these expressions are arithmetic operators like addition, subtraction, etc. Here are some arithmetic operators in Python:

Operators	Syntax	Functioning
+	x + y	Addition
–	x – y	Subtraction
*	x * y	Multiplication
/	x / y	Division
//	x // y	Quotient
%	x % y	Remainder

**	x ** y	Exponentiation

Example:
Let's see an exemplar code of arithmetic expressions in Python :

```
# Arithmetic Expressions
x = 40
y = 12
  add = x + y
sub = x - y
pro = x * y
div = x / y

print(add)
print(sub)
print(pro)
print(div)
```

Output-----

52

28

480

3.3333333333333335

3. Integral Expressions: These are the kind of expressions that produce only integer results after all computations and type conversions.

Example:

```
# Integral Expressions
a = 13
b = 12
c = a + b
print(c)
```

Output

25

4. Floating Expressions: These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.

Example:

```
# Floating Expressions
a = 13
b = 5
c = a / b
print(c)
```

Output

```
2.6
```

5. Relational Expressions: In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end. These expressions are also called Boolean expressions.

Example:

```
# Relational Expressions
a = 21
b = 13
c = 40
d = 37
  p = (a + b) >= (c - d)
print(p)
```

Output

```
True
```

6. Logical Expressions: These are kinds of expressions that result in either *True* or *False.* It basically specifies one or more conditions. For example, (10 == 9) is a condition if 10 is equal to 9. As we know it is not correct, so it will return False. Studying logical expressions, we also come across some logical operators which can be seen in logical expressions most often. Here are some logical operators in Python:

Operator	Syntax	Functioning
and	P and Q	It returns true if both P and Q are true otherwise returns false
or	P or Q	It returns true if at least one of P and Q is true
not	not P	It returns true if condition P is false

Example:

Let's have a look at an exemplar code :

```
P = (10 == 9)
Q = (7 > 5)
# Logical Expressions
R = P and Q
S = P or Q
T = not P
print(R)
print(S)
```

```
print(T)
```

Output

```
False
True
True
```

7. Bitwise Expressions: These are the kind of expressions in which computations are performed at bit level.

Example:

```
# Bitwise Expressions
a = 12
x = a >> 2
y = a << 1
print(x, y)
```

Output

```
3 24
```

8. Combinational Expressions: We can also use different types of expressions in a single expression, and that will be termed as combinational expressions.

Example:

```
# Combinational Expressions
a = 16
b = 12
c = a + (b >> 1)
print(c)
```

Output

```
22
```

But when we combine different types of expressions or use multiple operators in a single expression, operator precedence comes into play.

Python String Methods----------

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

Note: All string methods returns new values. They do not change the original string.

Method	Description
capitalize()	Converts the first character to upper case // syntax- print(ob.capitalize())

casefold()	Converts string into lower case //syntax-print(ob.casefold())
center()	Returns a centered string //syntax-print(ob.center(20,'*'))
count()	Returns the number of times a specified value occurs in a string//syntax-print(ob.count('anystring'))
encode()	Returns an encoded version of the string //syntax-print(ob.encode('utf-8'))
endswith()	Returns true if the string ends with the specified value //syntax-print(ob.endswith('anystring'))
expandtabs()	Sets the tab size of the string //syntax-print(ob.expandtabs(tabsize))
find()	Searches the string for a specified value and returns the position of where it was found // syntax- ob.find('anystring') or ob.find('anystring',start,end)
format()	Formats specified values in a string
format_map()	Formats specified values in a string //syntax-print(string.format_map(dictionary))
index()	Searches the string for a specified value and returns the position of where it was found //ob.index('anystring')
isalnum()	Returns True if all characters in the string are alphanumeric //ob.isnum()
isalpha()	Returns True if all characters in the string are in the alphabet // ob.isalpha()
isascii()	Returns True if all characters in the string are ascii characters //ob.isascii()

isdecimal()	Returns True if all characters in the string are decimals //ob.isdecimal()
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier //syntax-print(ob.isidentifier())
islower()	Returns True if all characters in the string are lower case //syntax-print(ob.islower())
isnumeric()	Returns True if all characters in the string are numeric //syntax-print(ob.isnumeric()) //syntax-print(ob.isnumeric)
isprintable()	Returns True if all characters in the string are printable // syntax-print(ob.isprintable)
isspace()	Returns True if all characters in the string are whitespaces // syntax-print(ob.isspace)
istitle()	Returns True if the string follows the rules of a title //syntax-print(ob.istitle)
isupper()	Returns True if all characters in the string are upper case //syntax-print(ob.isupper())
join()	Converts the elements of an iterable into a string //ob1.join(ob2)
ljust()	Returns a left justified version of the string //syntax-print(ob.ljust(width, fillchar))
lower()	Converts a string into lower case //syntax-print(ob.lower())
lstrip()	Returns a left trim version of the string // syntax-print(ob.lstrip()))….print(ob.lstrip('a'))
maketrans()	Returns a translation table to be used in translations

partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value //string.replace(old, new)
rfind()	Searches the string for a specified value and returns the last position of where it was found //string.rfind('anystring')
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list // syntax-string.rsplit()
rstrip()	Returns a right trim version of the string //syntax-string.rstrip
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case

translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Note: All string methods returns new values. They do not change the original string.