Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

**Features of Python:**Easy to use and Read - Python's syntax is clear and easy to read, making it an ideal language for both beginners and experienced programmers. This simplicity can lead to faster development and reduce the chances of errors.Dynamically Typed - The data types of variables are determined during run-time. We do not need to specify the data type of a variable during writing codes.High-level - High-level language means human readable code.Compiled and Interpreted - Python code first gets compiled into bytecode, and then interpreted line by line. When we download the Python in our system form [org](https://www.python.org/) we download the default implement of Python known as CPython. CPython is considered to be Complied and Interpreted both.Garbage Collected - Memory allocation and de-allocation are automatically managed. Programmers do not specifically need to manage the memory.Purely Object-Oriented - It refers to everything as an object, including numbers and strings.Cross-platform Compatibility - Python can be easily installed on Windows, macOS, and various Linux distributions, allowing developers to create software that runs across different operating systems.Rich Standard Library - Python comes with several standard libraries that provide ready-to-use modules and functions for various tasks, ranging from web development and data manipulation to machine learning and networking.Open Source - Python is an open-source, cost-free programming language. It is utilized in several sectors and disciplines as a result.

**Python Basic Syntax**There is no use of curly braces or semicolons in Python programming language. It is an English-like language. But Python uses indentation to define a block of code. Indentation is nothing but adding whitespace before the statement when it is needed.

For example -

1. def func():
2. statement 1
3. statement 2
4. …………………
5. …………………
6. statement N

In the above example, the statements that are the same level to the right belong to the function.

* Generally, we can use four whitespaces to define indentation.
* Instead of Semicolon as used in other languages, Python ends its statements with a NewLine character.
* Python is a case-sensitive language, which means that uppercase and lowercase letters are treated differently. For example, 'name' and 'Name' are two different variables in Python.
* In Python, comments can be added using the '#' symbol. (Even for multi and single lines)
* ['If'](https://www.javatpoint.com/python-if-else), 'otherwise', ['for'](https://www.javatpoint.com/python-for-loop), ['while'](https://www.javatpoint.com/python-while-loop), 'try', 'except', and 'finally' are a few reserved keywords in Python that cannot be used as variable names.

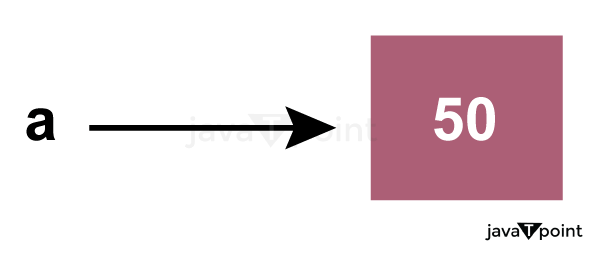
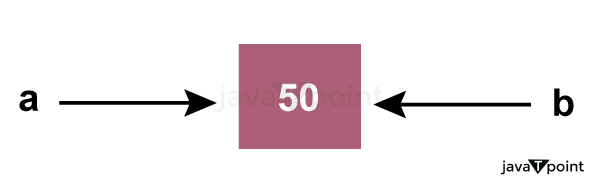
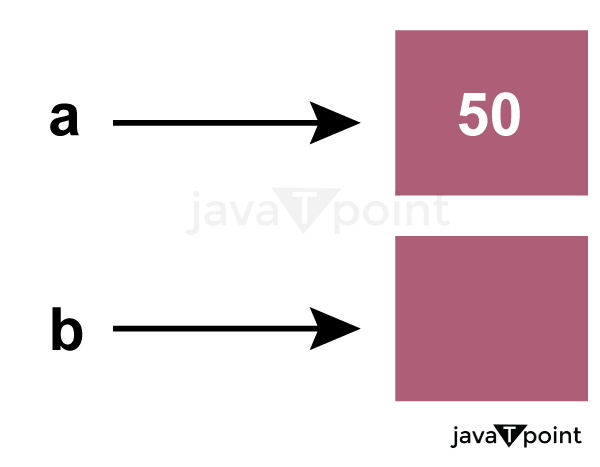
**Python Popular Frameworks and Libraries**

* Web development (Server-side) - [Django](https://www.javatpoint.com/django-tutorial) [Flask](https://www.javatpoint.com/flask-tutorial), [Pyramid](https://www.javatpoint.com/pyramid-framework-in-python), CherryPy
* GUIs based applications - [Tkinter](https://www.javatpoint.com/python-tkinter), [PyGTK](https://www.javatpoint.com/pygtk-for-gui-programming), [PyQt](https://www.javatpoint.com/pyqt-library-in-python), PyJs, etc.
* Machine Learning - [TensorFlow](https://www.javatpoint.com/tensorflow), [PyTorch](https://www.javatpoint.com/pytorch), [Scikit-learn](https://www.javatpoint.com/what-is-sklearn-in-python), [Matplotlib](https://www.javatpoint.com/matplotlib), [Scipy](https://www.javatpoint.com/python-scipy), etc.
* Mathematics - [NumPy](https://www.javatpoint.com/numpy-tutorial), [Pandas](https://www.javatpoint.com/python-pandas), etc.
* BeautifulSoup: a library for web scraping and parsing HTML and XML
* [Requests](https://www.javatpoint.com/python-requests-module-http-request): a library for making HTTP requests
* [SQLAlchemy](https://www.javatpoint.com/flask-sqlalchemy): a library for working with SQL databases
* [Kivy](https://www.javatpoint.com/kivy): a framework for building multi-touch applications
* [Pygame](https://www.javatpoint.com/pygame): a library for game development
* Pytest: a testing framework for Python Django
* [REST framework](https://www.javatpoint.com/create-rest-api-using-django-rest-framework): a toolkit for building RESTful APIs
* [FastAPI](https://www.javatpoint.com/fast-api-a-framework-to-create-apis): a modern, fast web framework for building APIs
* [Streamlit](https://www.javatpoint.com/deploy-a-machine-learning-model-using-streamlit-library): a library for building interactive web apps for machine learning and data science
* [NLTK](https://www.javatpoint.com/natural-language-toolkit): a library for natural language processing

**Python provides us the two ways to run a program:**  
Using Interactive interpreter prompt  
Using a script file (.py extension)

**Python Variables:**  
A variable is the name given to a memory location. A value-holding Python variable is also known as an identifier.

* Since Python is an infer language that is smart enough to determine the type of a variable, we do not need to specify its type in Python.
* Variable names must begin with a letter or an underscore, but they can be a group of both letters and digits.
* White space and special characters (!, @, #, %, etc.) are not allowed in the identifier name. ^, &, \*).
* The equal (=) operator is utilized to assign worth to a variable.

**Object References**  

In the above image, the variable a refers to an integer object. Suppose we assign the integer value 50 to a new variable b. The variable b refers to the same object that a point to because Python does not create another object.

* Make use of the built-in type() function in Python to determine object’s type(string, integer, float)
* Every object created in Python has a unique identifier. Python gives the dependability that no two items will have a similar identifier. The object identifier is identified using the built-in id() function
* a,b,c=50,'Aadhya',20.5  
  d=a  
  print(type(a),type(b),type(c))  
  #<class 'int'> <class 'str'> <class 'float'>  
  print(id(a),id(b),id(c),id(d))  
  #140729559109592 2397997306992 2397993938480 140729559109592  
  a=90  
  print(id(a),id(d))  
  #140729559110872 140729559109592

The multi-word keywords can be created by the following method.

* Camel Case - In the camel case, each word or abbreviation in the middle of begins with a capital letter. There is no intervention of whitespace. For example - nameOfStudent, valueOfVaraible, etc.
* Pascal Case - It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
* Snake Case - In the snake case, Words are separated by the underscore. For example - name\_of\_student, etc.

Multiple assignments, also known as assigning values to multiple variables in a single statement, is a feature of Python.

a=b=c=20 #Assigning same value to multiple variables  
a,b,c,d=20,30,61.69,'Aadhya' #Assigning different values to multiple variables

There are two types of variables in Python - Local variable and Global variable.

* The variables that are declared within the function and have scope within the function are known as local variables.
* Global variables can be utilized all through the program, and their extension is in the whole program. Global variables can be used inside or outside the function. By default, a variable declared outside of the function serves as the global variable. The function treats it as a local variable if we don't use the global keyword. Let's examine the following illustration.

# Declare a variable and initialize it  
x = 101  
  
  
# Global variable in function  
def mainFunction():  
 # printing a global variable  
 global x  
 print(x)  
 # modifying a global variable  
 x = 'Welcome To Javatpoint'  
 print(x)  
  
  
mainFunction()  
print(x)

* We can delete the variable using the **del** keyword. The syntax is given below. del <variable\_name>

x=30  
print(x) #30  
del x  
print(x) #NameError: name 'x' is not defined

* Python, compared to the other programming languages, does not support long int or float data types. It uses the int data type to handle all integer values. There is no special data type for storing larger numbers in Python.

Printing single and multiple variables

a,b,c=1,2,3  
print(a) #1  
print(a,b,c) #1 2 3

**Python Data Types**

Every value has a datatype, and variables can hold values. we don't have to characterize the sort of variable while announcing it. The interpreter binds the value implicitly to its type.



Numbers: Numeric values are stored in numbers. Python supports three kinds of numerical data.

* Int: Whole number worth can be any length, like numbers 10, 2, 29, - 20, - 150, and so on. An integer can be any length you want in Python. It’s worth having a place with int.
* Float: Float stores drifting point numbers like 1.9, 9.902, 15.2, etc. It can be accurate to within 15 decimal places.
* Complex: An intricate number contains an arranged pair, i.e., x + iy, where x and y signify the genuine and non-existent parts separately. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Boolean: True and False are the two default values for the Boolean type. These qualities are utilized to decide the given assertion valid or misleading. The class book indicates this. False can be represented by the 0 or the letter "F," while true can be represented by any value that is not zero.

a,b=1+3j,False  
print(type(a),type(b))  
#<class 'complex'> <class 'bool'>

Strings: Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes.

str='Hello'  
str1="Hello Python"  
str2="""Generally triple quotes are used for Doc strings"""  
print(str)  
print(str1)  
print(str2)

## Strings indexing and splitting: The indexing of the Python strings starts from 0. The slice operator [] is used to access the individual characters of the string. However, we can use the : (colon) operator in Python to access the substring from the given string. The upper range given in the slice operator is always exclusive. We can do the negative slicing in the string; it starts from the rightmost character, which is indicated as -1. The second rightmost index indicates -2, and so on.

string\_varible(start:stop:step)  🡪  The beginning indicates the beginning record position of the rundown. The stop signifies the last record position of the rundown. Within a start, the step is used to skip the nth element: stop.

str='Advika'  
print(str[0]+str[4]+str[5]) #Aka  
print(str[6]) #IndexError: string index out of range  
#########  
str1='Sohan'  
print(str1[:])#Sohan (Gives entire string)  
print(str1[0:])#Sohan (Gives entire string as nothing mentioned in upper range)  
print(str1[:5])#Sohan (Nothing mentioned in lower, takes from start)  
print(str1[2:4])#ha (4 is not included, so gives only str1[2],str1[3])

str='Aadhya'  
print(str[1:5:2]) #ah (1 is start 5 is end(exclusive) with 2 steps)  
print(str[-6:-2:2]) #ad

########################  
#Sohan -5 -4 -3 -2 -1  
print(str1[-4]) #o  
print(str1[:-1]) #Soha (str[-1] wont be included)  
print(str1[-5:]) #Sohan ( full string if no upper limit is mentioned)  
print(str1[-4:-1]) #oha  
#######################################  
print(str1[::-1]) #nahoS (Reversing string)  
#In this particular example, the slice statement [::-1] means start at  
# the end of the string and end at position 0, move with the step -1,  
# negative one, which means one step backwards.  
print(str1[::-3]) #n0 (Three steps backward)  
print(str1[::-2]) #nhs (Three steps backward)

## Reassigning Strings and deleting strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e., A string can only be replaced with new string since its content cannot be partially replaced. Strings are immutable in Python.  We cannot delete or remove the characters from the string.  But we can delete the entire string using the **del** keyword.

str='Aadhya'  
#str[2]='r' #TypeError: 'str' object does not support item assignment  
str='Prema'  
print(str) #Prema  
del str[0] #TypeError: 'str' object doesn't support item deletion  
del str  
print(str) #nothing prints

String Operators

|  |  |
| --- | --- |
| Operator | Description |
| + | concatenation operator used to join the strings given either side of the operator. |
| \* | repetition operator: It concatenates the multiple copies of the same string. |
| [] | slice operator. It is used to access the sub-strings of a particular string. |
| [:] | range slice operator. It is used to access the characters from the specified range. |
| in | membership operator. It returns if a particular sub-string is present in the specified string. |
| not in | It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string. |
| r/R | It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". |
| % | It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. We will discuss how formatting is done in python. |

str='Advika'  
str1='Sohan'  
print(str + ' ' + str1) #Advika Sohan  
print(str1\*3) #SohanSohanSohan  
print(str[2]) #v  
print(str1[2:4]) #ha  
print('a' in str) #True  
print('S' not in str1)  
print(r'C://Python directory') #C://Python directory  
print('My name is %s'%(str)) #My name is Advika

Escape Sequence: Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the SyntaxError as it contains both single and double-quotes. We have two ways: using triple quotes or backslash(\)

str='Hi 'Aadhya'! "How are you?"'  
print(str) #SyntaxError: invalid syntax  
str1='''Hi 'Aadhya'! "How are you?"''' #using triple quotes  
print(str1) #Hi 'Aadhya'! "How are you?"  
str2='Hi \'Aadhya\'! "How are you?"' #escaping single quote  
print(str2) #Hi 'Aadhya'! "How are you?"  
str3="Hi 'Aadhya'! \"How are you?\"" #escaping double quote  
print(str3) #Hi 'Aadhya'! "How are you?"

The list of an escape sequence is given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.** | **Escape Sequence** | **Description** | **Example** |
| 1. | \newline | It ignores the new line. | print('Advika \  Sohan') #Advika Sohan |
| 2. | \\ | Backslash | print('\\Advika') #\Advika (normal \interprets as escape sequence) print('\Aadhya') #\Aadhya but throws SyntaxWarning: invalid escape sequence '\A' |
| 3. | \' | Single Quotes | print('\'')#' |
| 4. | \\'' | Double Quotes | print("\"") #" |
| 5. | \a | ASCII Bell | print("\a") # |
| 6. | \b | ASCII Backspace(BS) | print('Love\bSohan') #LovSohan |
| 7. | \f | ASCII Formfeed | print('Love\fSohan') #LoveSohan |
| 8. | \n | ASCII Linefeed | print('Love\nSohan')  #Love #Sohan |
| 9. | \r | ASCII Carriege Return(CR) | print('Love you \rSohan') #Sohan |
| 10. | \t | ASCII Horizontal Tab | print('Love\tSohan') #Love Sohan |
| 11. | \v | ASCII Vertical Tab | print('Love\vSohan') #Love Sohan |
| 12. | \ooo | Character with octal value | print("\110\145\154\154\157") #Hello |
| 13 | \xHH | Character with hex value. | print("\x48\x65\x6c\x6c\x6f") #Hello |

The format() method: The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument.

print('{} and {} are cousins'.format('Avi','Acchu')) #using curly braces  
#Avi and Acchu are cousins  
print('{a} and {b} are good'.format(a='Avi',b='Acchu')) #using key attributes  
#Avi and Acchu are good  
print('{1} and {0} are sweet'.format('Avi','Acchu')) #positioning  
#Acchu and Avi are sweet  
a,b,c=2,24.5,'Advika'  
print('My name is %s.My age is %.2f. I love number %d'%(c,b,a))  
#My name is Advika.My age is 24.50. I love number 2

**String Functions**

**Lists:** In Python, the sequence of various data types is stored in a list. A list is a collection of different kinds of values or items. Since Python lists are mutable, we can change their elements after forming. The comma (,) and the square brackets [enclose the List's items] serve as separators. A list is a collection of items separated by commas and denoted by the symbol [].

stud=['Advika',16,'Graduate']  
Dep=['CS',35,'CH',20]  
HOD\_CS=['Barun',5]  
HOD\_CH=['Venkat',6]  
#printing student details  
print('Student name:%s Id:%d Category:%s'%(stud[0],stud[1],stud[2]))  
#Student name:Advika Id:16 Category:Graduate  
print('Dep1:%s Id:%d Dep2:%s Id:%d'%(Dep[0],Dep[1],Dep[2],Dep[3]))  
#Dep1:CS Id:35 Dep2:CH Id:20  
print('CS HOD:%s Id:%d' %(HOD\_CS[0],HOD\_CS[1]))  
#CS HOD:Barun Id:5  
print('CH HOD:%s Id:%d' %(HOD\_CH[0],HOD\_CH[1]))  
#CH HOD:Venkat Id:6

Comparing Lists

a=[1,2,3]  
b=[2,1,3]  
c=[1,2,3]  
print(a==b) #False (ordering is important)  
print(a is b) #False  
print(a==c) #True  
print(a is c) #False  
d,e=[],[]  
print(d==e) #True  
print(d is e) #False

Indexing and splitting: The indexing procedure is carried out similarly to string processing.

list=[0,1,2,3,4,5]  
print(list[1]) #1  
print(list[:]) #[0, 1, 2, 3, 4, 5]  
print(list[2:5]) #[2, 3, 4]  
print(list[2:5:2]) #[2, 4]  
print(list[:-1]) #[0, 1, 2, 3, 4]  
print(list[-5:-1:3]) #[1, 4]  
print(list[::-1]) #[5, 4, 3, 2, 1, 0]  
print(list[::-3]) #[5, 2]

Updating values in list : Due to their mutability and the slice and assignment operator's ability to update their values, lists are Python's most adaptable data structure. Python's append() and insert() methods can also add values to a list.

list=[0,1,2,3,4,5]  
list[2]=10  
print(list) #[0, 1, 10, 3, 4, 5]  
list[1:4]=(10,11,12)  
print(list) #[0, 10, 11, 12, 4, 5]  
list[-1]=15  
print(list) #[0, 10, 11, 12, 4, 15]  
list[2:5:2] =[100,200]  
print(list) #[0, 10, 100, 12, 200, 15]  
list[::-1] =['a','b','c','d','e','f']  
print(list) #['f', 'e', 'd', 'c', 'b', 'a']  
list.append('z')  
print(list) #['f', 'e', 'd', 'c', 'b', 'a', 'z']  
list.insert(5,9)  
print(list) #['f', 'e', 'd', 'c', 'b', 9, 'a', 'z']

list=[0,1,2,3,4,5]  
del list[0]  
print(list) #[1, 2, 3, 4, 5]  
list.remove(3)  
print(list) #[1, 2, 4, 5]  
del list[1:3]  
print(list) #[1, 5]

**Python Keywords**Every scripting language has designated words or keywords, with definitions and usage guidelines. Python contains thirty-five keywords in the most recent version, i.e., Python 3.8. Here we have shown a complete list of Python keywords for the reader's reference.

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

import keyword #importing keyword library which has lists  
print( keyword.kwlist ) # displaying the complete list using "kwlist()."   
#['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break',   
# 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally',   
# 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',   
# 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

help("keywords")  
#Here is a list of the Python keywords. Enter any keyword to get more help.  
help("else")  
#explanation for else

How to Identify Python Keywords:

* Write Code on a Syntax Highlighting IDE
* Verify Keywords with Script in a REPL
* Look for a SyntaxError

Value Keywords: True, False, None

In Python script, the True Python keyword represents the Boolean true state. False is a keyword equivalent to True, except it has the negative Boolean state of false.True and False are those keywords that can be allocated to variables or parameters and are compared directly.

print(4==4) #True  
print(True==3) #False  
print(True+False+True) #2  
print(False==0) #True

None is a Python keyword that means "nothing." None is known as nil, null, or undefined in different computer language

print(None==0)#False  
print(None=='')#False  
a=None  
b=None  
print(a==b)#True  
print(a is b) #True

* If a no\_return\_function returns nothing, it will simply return a None value.
* If a function does not have a return clause, it will give None as the default output.
* def no\_return(): #No return value function  
   p=10\*2  
  def is\_even(num): #Function with return value  
   r=num%2  
   if (r==0):  
   return True  
   else:  
   return False  
  def is\_odd(num): #Function with return value but for only one condition  
   r=num%2  
   if (r==0):  
   return False  
    
  print(no\_return()) #None  
  print(is\_even(10)) #True  
  print(is\_odd(11)) #None

Operator Keywords: and, or, not, in, is

* The Python keyword and determines whether both the left-hand side and right-hand side operands and are true or false. The outcome will be True if both components are true. If one is false, the outcome will also be False.   
  Eg <component1> and <component2>
* The or keyword in Python is utilized to check if, at minimum, 1 of the inputs is true   
  Eg <component1> or <component2>
* The not keyword in Python is utilized to acquire a variable's contrary Boolean value:
* The in keyword of Python is a robust confinement checker, also known as a membership operator. If you provide it an element to seek and a container or series to seek into, it will give True or False, depending on if that given element was located in the given container: <an\_element> in <a\_container>
* print(True and False) #False  
  print( True or False) #True  
  print(not False) #True  
  print('p' in 'stamp') #True  
  container='Aadhya'   
  print('p' in container) #False
* == (Equality Operator): The == operator checks for equality of values between two objects.It compares the values of the objects, not their identities. It is used to determine if the contents of the objects are the same.
* is (Identity Operator): The is operator checks for identity of objects, i.e., whether two variables refer to the same object in memory. It compares the memory address of the objects, not their values.It is used to determine if two variables refer to the exact same object.
* **Mutable objects** like lists and dictionaries can be modified after creation. When you create a new list or dictionary, even if it has the same content as another, they are stored independently in memory. This is because they are changeable, and any modification to one should not affect the other. Therefore, even if two blank dictionaries or lists have the same content, they are not identical entities in memory.
* **Immutable objects** like strings and tuples cannot be modified after creation. When you create an immutable object, Python may reuse the same object in memory if it already exists. This is possible because the content of immutable objects cannot change, so it's safe to share them across different variables. Therefore, two equal strings or tuples are also identical because they refer to the same unique memory region.
* a=50  
  b=a  
  c=50  
  print(a==b) #True  
  print(a is b) #True  
  print(a==c) #True  
  print(a is c) #True  
  #In Python, small integers and some small strings are cached and reused for memory efficiency.  
    
  a=[1,2,3]  
  b=[1,2,3]  
  print(a==b) #True  
  print(a is b) #False  
    
  a=None  
  b=None  
  print(a==b) #True  
  print(a is b) #True  
  #True, False, None are same everywhere  
    
  a=[]  
  b=[]  
  print(a==b) #True  
  print(a is b) #False  
    
  a=""  
  b=""  
  print(a==b) #True  
  print(a is b) #True

Mutability immutability in detail

Exception Handling Keywords - try, except, raise, finally, and assert

* try: This keyword is designed to handle exceptions. When there is some kind of error, the program inside the "try" block is verified, but the code in that block is not executed.
* except: As previously stated, this operates in conjunction with "try" to handle exceptions.
* finally: Whatever the outcome of the "try" section, the "finally" box is implemented every time.
* **raise:** The raise keyword could be used to specifically raise an exception.
* **assert:** This method is used to help in troubleshooting. Often used to ensure that code is correct. Nothing occurs if an expression is interpreted as true; however, if it is false, "AssertionError" is raised. An output with the error, followed by a comma, can also be printed.  
  -----------------------------------------------------------------------------------------------------------------

The nonlocal Keyword  
Nested functions or inner functions always has access to outer functions but cannot modify it.

global: The global keyword is used inside a function to declare that a variable inside the function refers to a global variable defined outside the function. When you use global within a function to modify a variable, you're indicating that the variable exists in the global scope and should be modified there, rather than creating a new local variable with the same name. If a variable is modified using global inside a function, its value will be changed globally.

The nonlocal keyword is used inside a nested function (a function defined within another function) to indicate that a variable refers to a variable defined in the nearest enclosing scope that is not global.It allows you to modify a variable in the outer (enclosing) scope from within a nested function, without using the global scope.   
nonlocal is useful when you want to modify a variable that is in an enclosing function's scope, but not in the global scope.  
  
#Example without using nonlocal

def outside\_function():  
 y=20  
 def inside\_func():  
 print(y) #20 --inner functions can access variables declared in outside functions  
 x=30  
 print(x) #30  
 y=40 #UnboundLocalError: cannot access local variable 'y' where it is not associated with a value  
 print(y)  
 inside\_func()  
 print(y) #20  
 print(x) #name x is not defined -- outer functions cannot access inner functions  
outside\_function()

Here without use of nonlocal, we cannot modify outside variable in inner functions.  
Remember, inner functions can always access outer function variable, but cannot modify it.

#Example with using nonlocal

def outside\_function():  
 y=20  
 def inside\_func():  
 nonlocal y  
 print(y) #20 --inner functions can access variables declared in outside functions  
 x=30  
 print(x) #30  
 y=40 #we can modify y after declaring it as nonlocal variable  
 print(y) #40  
 inside\_func()  
 print(y) #40 --here y value changed because of nearest enclosing scope  
 print(x) #name x is not defined -- outer functions cannot access inner functions  
outside\_function()

Now, we are able to modify outer function value in inner function as we have declared non local variable.

#Example to show diff between nonlocal and global

x = 30  
def outer\_greater\_function():  
 y=50 #declaring a variable  
 def outer\_function():  
 y=10   
 def inner\_function():  
 global x  
 nonlocal y #declaring global variable  
 x=40  
 y =20  
 print(y) #20  
 inner\_function() #calling inner function  
 print(y) #20 --y changes here as it is nearest enclosing scope  
  
 outer\_function() #calling outer function  
 print(y) #50 --y doesn't change here as it is outside of nearest sope  
 print(x) #40 --x changed globally  
outer\_greater\_function()

#Example to show case where nonlocal and global acts same

x = 30  
def outer\_greater\_function():  
 y=50 #declaring a variable  
 def outer\_function():  
 nonlocal y  
 y=10  
 def inner\_function():  
 global x  
 nonlocal y #declaring global variable  
 x=40  
 y =20  
 print(y) #20  
 inner\_function() #calling inner function  
 print(y) #20 --y changes here as it is nearest enclosing scope  
  
 outer\_function() #calling outer function  
 print(y) #20 --y change here as it is nearest enclosing scope (using nonlocal)  
 print(x) #40 --x changed globally  
outer\_greater\_function()

Instead of using nonlocal everywhere, it is suggested to use global. Nonlocal works as global if we declare it in every enclosing loop.

The pass Keyword

In Python, a null sentence is called a pass. It serves as a stand-in for something else. When it is run, nothing occurs.

def add():  
#IndentationError: expected an indented block after function definition on line 1  
  
def add1():  
 pass

The return Keyword

The return expression is used to leave a function and generate a result.  
The None keyword is returned by default if we don't specifically return a value. The accompanying example demonstrates this.

The del Keyword  
The del keyword is used to remove any reference to an object.

var1=var2=20  
print(var1,var2) #20 20  
del var2  
print(var1,var2) #name 'var2' is not defined. Did you mean: 'var1'?  
  
list=['A','B','C','D']  
del list[2]  
print(list) #['A', 'B', 'D']

**Python Operators**

## The operator is a symbol that performs a specific operation between two operands, according to one definition. 1) Arithmetic Operators +,-,\*,/,//,\*\*,%

// (Floor division): It provides the quotient's floor value, which is obtained by dividing the two operands.

a,b=5,3  
print('Sum: ',a+b) #8  
print('diff: ',a-b) #2  
print('product: ',a\*b) #15  
print('quotient: ',a/b) #1.6666666666666667  
print('Rem: ',a%b) #2  
print('Exp: ',a\*\*b) #125  
print('F\_quo: ',a//b) #1

## 2) Comparison operator Comparison operators compare the values of the two operands and return a true or false Boolean value in accordance. ==,!=, <,>, <=,>=

a,b=5,3  
print('Equal or not:',a==b) #False  
print('Not equal or not:',a!=b) #True  
print('Less than or not:',a<b) #False  
print('Greater or not:',a>b) #True  
print('Less than/equal to or not:',a<=b) #False  
print('Greater than/equal to or not:',a>=b) #True

## 3) Assignment Operators

Using the assignment operators, the right expression's value is assigned to the left operand.  
+=, -=, \*=, /=, \*\*=, %=, //=

a,b=5,3  
print('Before',a) #5  
a+=b  
print('After addition', a) #8  
a-=b  
print('After Subtraction', a) #5  
a\*=b  
print('After multiplication', a) #15  
a/=b  
print('After division', a) #5.0  
a\*\*=b  
print('After exponential', a) #125  
a//=b  
print('After floor division', a) #41.0  
a%=b  
print('After remaninder', a) #2.0

## 4) Bitwise Operators

The two operands' values are processed bit by bit by the bitwise operators. The examples of Bitwise operators are bitwise OR (|), bitwise AND (&), bitwise XOR (^), negation (~), Left shift (<<), and Right shift (>>).

**Bitwise XOR:** XOR is a bitwise operator, and it stands for "exclusive or." It performs logical operation. If input bits are the same, then the output will be 0 else 1

**Bitwise right shift:** Shifts the bits of the number to the right and fills 0 on voids left (fills 1 in the case of a negative number) as a result. Similar effect as of dividing the number with some power of two.  
Example 1: a = 10 = 0000 1010 (Binary)  
a >> 1 = 0000 0101 = 5  
Example 2: a = -10 = 1111 0110 (Binary)  
a >> 1 = 1111 1011 = -5

**Bitwise left shift:**Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.  
Example 1: a = 5 = 0000 0101 (Binary)  
a << 1 = 0000 1010 = 10  
a << 2 = 0001 0100 = 20

Example 2: b = -10 = 1111 0110 (Binary)  
b << 1 = 1110 1100 = -20  
b << 2 = 1101 1000 = -40

a,b=10,4  
#a=1010 b=0100  
print(a&b) #0 --0000  
print(a|b) #14 --1110  
print(a^b) #14 --1110  
print(~b) #-5  
print(a<<b) #160 -- 1010 0000  
print(a>>b) #0 -- 0000 00000 (a can be viewed as 0000 1010)

## 5) Logical Operators In the case of logical AND, if the first one is 0, it does not depend upon the second one.  This is diff between logical and bitwise. and, or, not

a,b=10,4  
print(a>b and a==b) #False  
print(a>b or a==b) #True  
print(not(a==b)) #True

## 6) Membership Operators

The result is true if the value is in the data structure; otherwise, it returns false.  
in, not in

name='Aadhya'  
hobbies=['Singing', 'dancing','music']  
sal={25000,50000,200000}  
print('d' in name) #True  
print('music' in hobbies) #True  
print('i' in hobbies) # False --In lists/ tuples/dictionary -- searches for exact match not character  
print(50000 in sal) #True  
print('s' not in name) #True

## 7) Identity Operators

Is: If the references on both sides point to the same object, it is determined to be true.

Is not: If the references on both sides do not point at the same object, it is determined to be true.

name='Aadhya'  
p\_name='Aadhya'  
sal={1,2,3}  
p\_sal=sal  
o\_sal={1,2,3}  
print(name is p\_name) #True  
print(sal is p\_sal) #True  
print(sal is o\_sal) #False  
print(o\_sal is not p\_sal) #True  
  
  
print(id(sal), id(p\_sal),id(o\_sal), id(name), id(p\_name))  
#3059833032704 3059833032704 3059833036736 3059832984688 3059832984688

### **Python Comments** In Python, there are 3 types of comments. Single-Line Comments: A single-line comment of Python is the one that has a hashtag # at the beginning of it and continues until the finish of the line.  Multi-Line Comments**:** With Multiple Hashtags (#) or Using String Literals(triple quotes “””) Because Python overlooks string expressions that aren't allocated to a variable, we can utilize them as comments. On running code, there will be no output Python Docstring: The strings enclosed in triple quotes that come immediately after the defined function are called Python docstring. The docstring is then readily accessible in Python using the \_\_doc\_\_ attribute.

def add(x, y):  
 *"""This function adds the values of x and y"""* return x + y  
  
  
# Displaying the docstring of the add function  
print(add.\_\_doc\_\_) #This function adds the values of x and y

#This is a comment  
#it will not display  
*"""  
This also will not be displayed as nothing runs  
"""*# Process finished with exit code 0

# **How to dynamically take inputs from user** input (): This function first takes the input from the user and converts it into a string. The type of the returned object always will be <class ‘str’>.  if you enter an integer value still input() function converts it into a string. You need to explicitly convert it into an integer in your code using [typecasting](https://www.geeksforgeeks.org/taking-input-from-console-in-python/).

a=10  
b=int(input("Enter a number")) #10  
print(a+b) #20

Using [split()](https://www.geeksforgeeks.org/python-string-split/) method :   
This function helps in getting multiple inputs from users. It breaks the given input by the specified separator. If a separator is not provided, then any white space is a separator. Generally, users use a split () method to split a Python string but one can use it in taking multiple inputs. Syntax: input().split(separator, maxsplit)

a,b=input("Enter two numbers:").split()  
print(int(a)+int(b))

a,b=[int(x) for x in input("Enter two numbers:").split()]  
print(a+b)

# **Python If-else statements**

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.   
The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.  
The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

# Program to check whether a number is even or not  
num=int(input('Enter number:'))   
if num%2==0:  
 print('Even')  
else:  
 print('Odd')

# Program to check voting eligibility  
age=int(input('Enter age:'))  
if age>=18:  
 print('Eligible to vote')  
else:  
 print('Have to wait more ',(18-age),'years')

# Program to check largest number  
a,b,c=input("Enter three numbers").split()  
a=int(a)  
b=int(b)  
c=int(c)  
if (a>b and a>c):  
 print(a,' is largest')  
elif (b>c):  
 print(b, ' is largest')  
else:  
 print(c,' is largest')

Or simply

a,b,c=[int(x) for x in input("Enter three numbers").split()]

# **Python Loops**

|  |  |  |
| --- | --- | --- |
| Sr.No. | Name of the loop | Loop Type & Description |
| 1 | While loop | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| 2 | For loop | This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable. |
| 3 | Nested loops | We can iterate a loop inside another loop. |

|  |  |  |
| --- | --- | --- |
| Sr.No. | Name of the control statement | Description |
| 1 | Break statement | This command terminates the loop's execution and transfers the program's control to the statement next to the loop. |
| 2 | Continue statement | This command skips the current iteration of the loop. The statements following the continue statement are not executed once the Python interpreter reaches the continue statement. |
| 3 | Pass statement | The pass statement is used when a statement is syntactically necessary, but no code is to be executed. |