**What is Python?**

Python is a high-level, general-purpose, and very popular programming language. Python programming language (latest Python 3) is being used in web development, and Machine Learning applications

Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

[Python](https://www.geeksforgeeks.org/python-programming-language/learn-python-tutorial/) was developed by Guido van Rossum in the early 1990s and its latest version is 3.11.0, we can simply call it Python3. Python 3.0 was released in 2008. and is interpreted language i.e it’s not compiled and the interpreter will check the code line by line.

## Writing your first Python Program to Learn Python Programming

There are two ways you can execute your Python program:

1. First, we write a program in a file and run it one time.
2. Second, run a code line by line.

# Python Program to print Hello World

print("Hello Python Code")

[Comments in Python](https://www.geeksforgeeks.org/python-comments/)

Comments in Python are the lines in the code that are ignored by the interpreter during the execution of the program. Also, Comments enhance the readability of the code and help the programmers to understand the code very carefully.

* Python

|  |
| --- |
| # sample comment  # This is Python Comment  name = " Hello Python Code "  print(name) |

## [Keywords in Python](https://www.geeksforgeeks.org/python-keywords/)

Keywords in Python are reserved words that can not be used as a variable name, function name, or any other identifier.

| **Keywords** | | |
| --- | --- | --- |
| **and** | **False** | [**nonlocal**](https://www.geeksforgeeks.org/python-nonlocal-keyword/) |
| [**as**](https://www.geeksforgeeks.org/python-as-keyword/) | [**finally**](https://www.geeksforgeeks.org/finally-keyword-in-python/) | [**not**](https://www.geeksforgeeks.org/python-not-keyword/) |
| [**assert**](https://www.geeksforgeeks.org/python-assert-keyword/) | [**for**](https://www.geeksforgeeks.org/python-for-loops/) | **or** |
| [**break**](https://www.geeksforgeeks.org/python-break-statement/) | **from** | [**pass**](https://www.geeksforgeeks.org/python-pass-statement/) |
| [**class**](https://www.geeksforgeeks.org/python-classes-and-objects/) | [**global**](https://www.geeksforgeeks.org/global-keyword-in-python/) | [**raise**](https://www.geeksforgeeks.org/python-raise-keyword/) |
| [**continue**](https://www.geeksforgeeks.org/python-continue-statement/) | **if** | [**return**](https://www.geeksforgeeks.org/python-return-statement/) |
| [**def**](https://www.geeksforgeeks.org/python-def-keyword/) | [**import**](https://www.geeksforgeeks.org/import-module-python/) | **True** |
| [**del**](https://www.geeksforgeeks.org/python-del-to-delete-objects/) | [**is**](https://www.geeksforgeeks.org/is-keyword-in-python/) | [**try**](https://www.geeksforgeeks.org/python-try-except/) |
| **elif** | [**in**](https://www.geeksforgeeks.org/python-in-keyword/) | [**while**](https://www.geeksforgeeks.org/python-while-loop/) |
| **else** | [**lambda**](https://www.geeksforgeeks.org/python-lambda/) | [**with**](https://www.geeksforgeeks.org/with-statement-in-python/) |
| [**except**](https://www.geeksforgeeks.org/python-try-except/) | [**None**](https://www.geeksforgeeks.org/python-none-keyword/) | [**yield**](https://www.geeksforgeeks.org/python-yield-keyword/) |

# Python 3 basics

Python 3 is a popular high-level programming language used for a wide variety of applications. Here are some basics of Python 3 that you should know:

1. Variables: In Python 3, variables are created by assigning a value to a name. For example, x = 5 creates a variable called x and assigns the value 5 to it.
2. Data types: Python 3 supports several built-in data types, including integers, floats, strings, booleans, lists, tuples, and dictionaries.
3. Operators: Python 3 supports a variety of operators, including arithmetic operators (+, -, \*, /), comparison operators (>, <, ==, !=), and logical operators (and, or, not).
4. Control flow statements: Python 3 supports several control flow statements, including if-else statements, for loops, and while loops. These statements allow you to control the flow of execution in your code.
5. Functions: In Python 3, functions are created using the def keyword. For example, def my\_function(x): creates a function called my\_function that takes one argument called x.
6. Input and output: In Python 3, you can use the input() function to get user input, and the print() function to output text to the console.
7. Modules: Python 3 supports modules, which are collections of functions and variables that can be imported and used in other Python code. You can import modules using the import keyword.

### Advantages of Python 3:

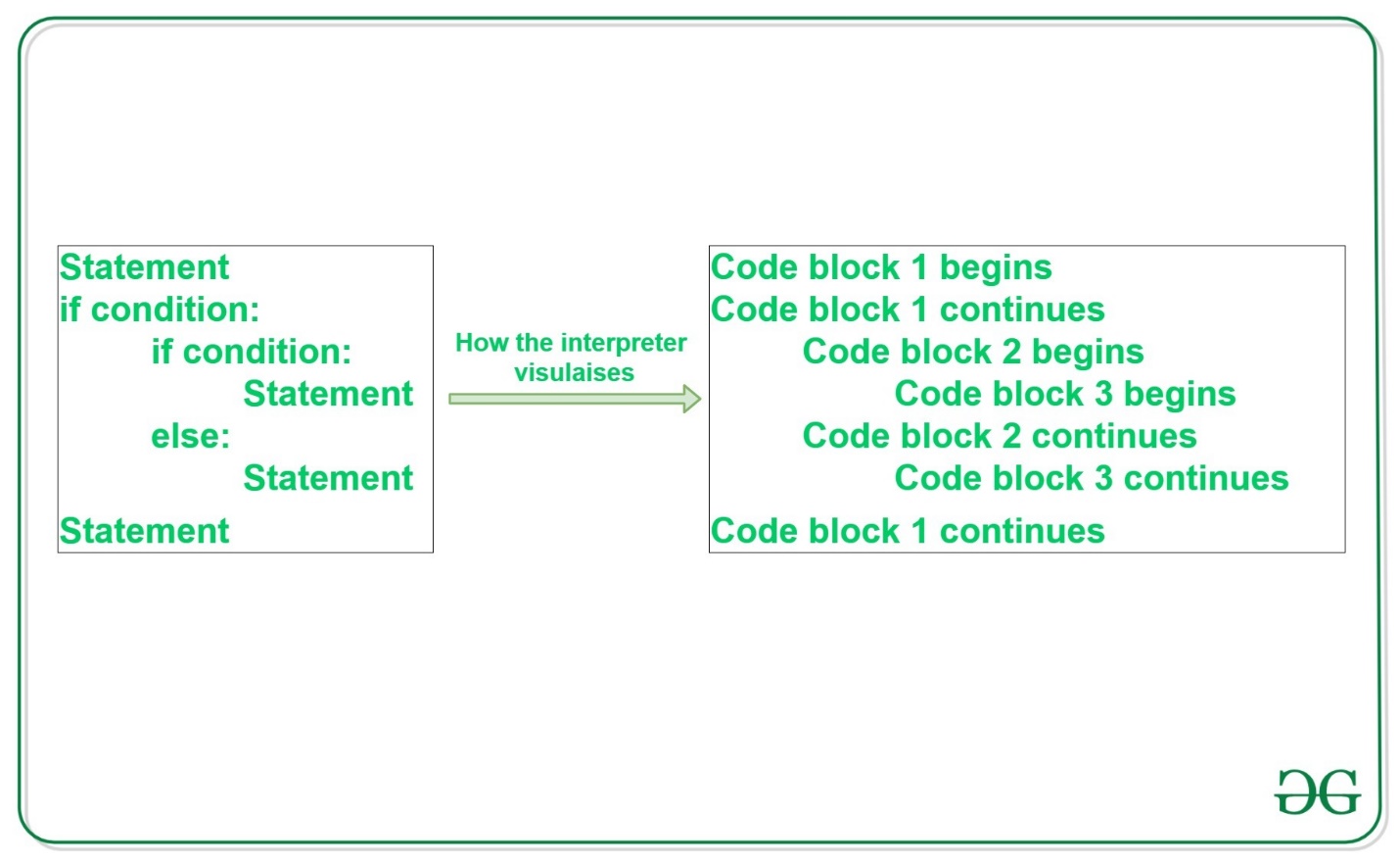
1. Python 3 has a simple syntax that is easy to learn and read, making it a good choice for beginners.
2. Python 3 is a high-level language that has a large standard library and many third-party libraries available, making it a versatile language that can be used for a wide variety of applications.
3. Python 3 supports multiple programming paradigms, including object-oriented, functional, and procedural programming.
4. Python 3 is an interpreted language, meaning that it does not need to be compiled before running, making it easy to write and test code quickly.
5. Python 3 has good support for data analysis and scientific computing, with libraries such as NumPy and Pandas.

# Indentation in Python

Indentation is a very important concept of Python because without properly indenting the Python code, you will end up seeing IndentationError and the code will not get compiled.

## Python Indentation

Python indentation refers to adding white space before a statement to a particular block of code. In another word, all the statements with the same space to the right, belong to the same code block.



# Python program showing

# indentation

site = 'gfg'

if site == 'gfg':

    print('Logging on to PYTHON...')

else:

    print('retype the URL.')

print('All set !')

## [Python Variable](https://www.geeksforgeeks.org/python-variables/)

Python Variable is containers that store values. Python is not “statically typed”. An Example of a Variable in Python is a representational name that serves as a pointer to an object. Once an object is assigned to a variable, it can be referred to by that name.

### **Rules for Python variables**

* A Pythonvariable name must start with a letter or the underscore character.
* A Python variable name cannot start with a number.
* A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ ).
* Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
* The [reserved words(keywords)](https://www.geeksforgeeks.org/python-keywords-and-identifiers/) in Python cannot be used to name the variable in Python.

**Example**

* Python

|  |
| --- |
| # An integer assignment  age = 45  # A floating point  salary = 1456.8    # A string  name = "John"  print(age)  print(salary)  print(name) |

**Output**

45

1456.8

John

## [Python Data Types](https://www.geeksforgeeks.org/python-data-types/)

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are classes and variables are instances (objects) of these classes.



**Example:**This code assigns variable **‘x’** different values of various data types in Python.

* Python

|  |
| --- |
| x = "Hello World" # string  x = 50  # integer  x = 60.5  # float  x = 3j  # complex  x = ["one", "two", "three"]  # list  x = ("one", "two", "three ")  # tuple  x = {"name": "Suraj", "age": 24} # dict  x = {" one", "two", "three "} # set  x = True  # bool  x = b"Geeks" # binary |

OutPut:->

50

60.5

3j

['one', 'two', 'three']

('one', 'two', 'three ')

{'name': 'Suraj', 'age': 24}

{'three ', 'two', ' one'}

True

<class 'bytes'>

[Python Input/Output](https://www.geeksforgeeks.org/input-and-output-in-python/)

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’>. It does not evaluate the expression it just returns the complete statement as String, and will print it.

|  |
| --- |
| # Python program show input and Output  val = input("Enter your value: ")  print(val) |

## Python Operators

In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

### [**Arithmetic Operators**](https://www.geeksforgeeks.org/python-arithmetic-operators/)

Python Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, and division.

### Precedence of Arithmetic Operators

The precedence of Arithmetic Operators in Python is as follows:

1. P – Parentheses
2. E – Exponentiation
3. M – Multiplication (Multiplication and division have the same precedence)
4. D – Division
5. A – Addition (Addition and subtraction have the same precedence)
6. S – Subtraction

Example

|  |
| --- |
| a = 9  b = 4  add = a + b  sub = a - b  mul = a \* b  mod = a % b  p = a \*\* b  print(add)  print(sub)  print(mul)  print(mod)  print(p) |

**Output**

13

5

36

1

6561

### [**Logical Operators**](https://www.geeksforgeeks.org/python-logical-operators/)

Python [Logical operators](https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/) perform **Logical AND**, **Logical OR**, and**Logical NOT** operations. It is used to combine conditional statements.

* Python

|  |
| --- |
| a = True  b = False  print(a and b)  print(a or b)  print(not a) |

**Output**

False

True

False

### [**Bitwise Operators**](https://www.geeksforgeeks.org/python-bitwise-operators/)

Python [Bitwise operators](https://www.geeksforgeeks.org/python-bitwise-operators/) act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

* Python

|  |
| --- |
| a = 10  b = 4  print(a & b)  print(a | b)  print(~a)  print(a ^ b)  print(a >> 2)  print(a << 2) |

**Output**

0

14

-11

14

2

40

### [**Assignment Operators**](https://www.geeksforgeeks.org/assignment-operators-in-python/)

Python [Assignment operators](https://www.geeksforgeeks.org/assignment-operators-in-python/) are used to assign values to the variables.

* Python

|  |
| --- |
| a = 10  b = a  print(b)  b += a  print(b)  b -= a  print(b)  b \*= a  print(b)  b <<= a  print(b) |

**Output**

10

20

10

100

102400

## Python If Else

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But if we want to do something else if the condition is false, we can use the else statement with the if statement to execute a block of code when the if condition is false.

### Example 1: [**Python IF-Else**](https://www.geeksforgeeks.org/python-if-else/)

* Python

|  |
| --- |
| i = 20  if (i < 15):      print("i is smaller than 15")      print("i'm in if Block")  else:      print("i is greater than 15")      print("i'm in else Block")  print("i'm not in if and not in else Block") |

**Output**

i is greater than 15

i'm in else Block

i'm not in if and not in else Block

### Example 2: [**Python if-elif-else ladder**](https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/)

* Python

|  |
| --- |
| i = 20  if (i == 10):      print("i is 10")  elif (i == 15):      print("i is 15")  elif (i == 20):      print("i is 20")  else:      print("i is not present") |

**Output**

i is 20

## [Python For Loop](https://www.geeksforgeeks.org/python-for-loops/)

Python For loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set, or Dictionary. Here we will see a “for” loop in conjunction with the range() function to generate a sequence of numbers starting from 0, up to (but not including) 10, and with a step size of 2. For each number in the sequence, the loop prints its value using the print() function.

* Python

|  |
| --- |
| for i in range(0, 10, 2):      print(i) |

**Output**

0

2

4

6

8

## [Python While Loop](https://www.geeksforgeeks.org/python-while-loop/)

In this example, the condition for while will be True as long as the counter variable (count) is less than 3.

* Python

|  |
| --- |
| # Python program to illustrate while loop  count = 0  while (count < 3):      count = count + 1      print("Hello Geek") |

**Output**

Hello Geek

Hello Geek

Hello Geek

# break, continue and pass in Python

Using loops in Python automates and repeats the tasks in an efficient manner. But sometimes, there may arise a condition where you want to exit the loop completely, skip an iteration or ignore that condition. These can be done by **loop control statements**. Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

 Python supports the following control statements:

* [Break statement](https://www.geeksforgeeks.org/python-break-statement/)
* [Continue statement](https://www.geeksforgeeks.org/python-continue-statement/)
* [Pass statement](https://www.geeksforgeeks.org/python-pass-statement/)

## Break Statement in Python

The break statement in [Python](https://www.geeksforgeeks.org/python-programming-language/) is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available. If the break statement is present in the nested loop, then it terminates only those loops which contain the break statement.

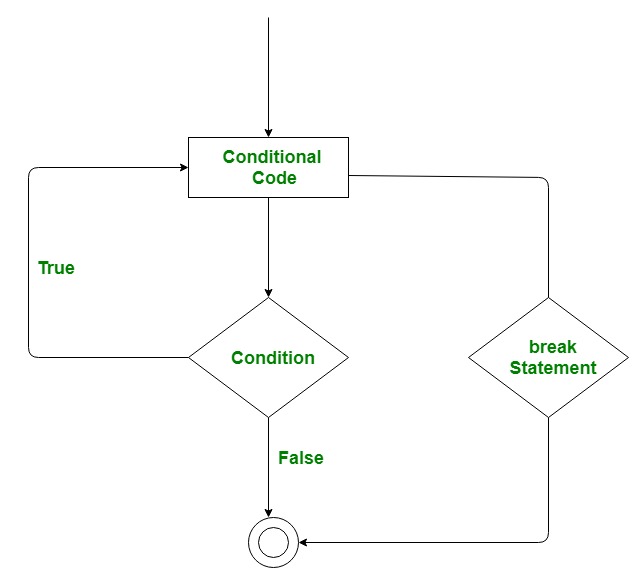
### **Syntax of Break Statement**

The break statement in Python has the following syntax:

for / while loop:  
 # statement(s)  
 if condition:  
 break  
 # statement(s)  
# loop end

### Working on Python Break Statement

The working of the break statement in Python is depicted in the following flowchart:



**EXAMPLE 1--**

# Python program to demonstrate

# break statement

s = 'Hello Python code'

# Using for loop

for letter in s:

    print(letter)

    # break the loop as soon it sees 'e'

    # or 's'

    if letter == 'e' or letter == 's':

        break

print("Out of for loop")

print()

i = 0

# Using while loop

while True:

    print(s[i])

    # break the loop as soon it sees 'e'

    # or 's'

    if s[i] == 'e' or s[i] == 's':

        break

    i += 1

print("Out of while loop")

**EXAMPLE 2--**

# Python program to demonstrate

# break statement with nested

# for loop

# first for loop

for i in range(1, 5):

    # second for loop

    for j in range(2, 6):

        # break the loop if

        # j is divisible by i

        if j%i == 0:

            break

        print(i, " ", j)

## Continue Statement in Python

Continue is also a loop control statement just like the break statement. continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop. As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

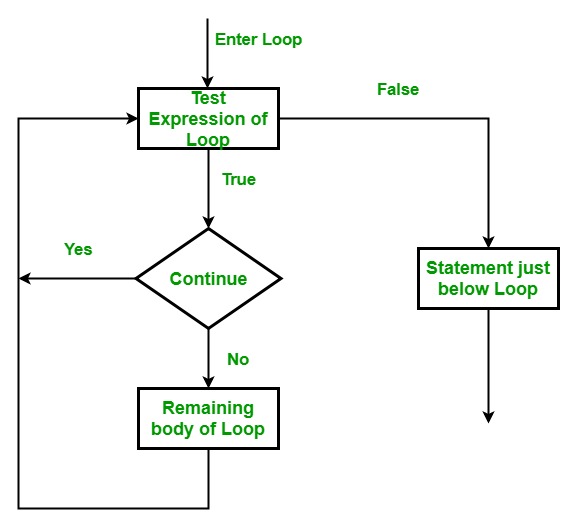
### **Syntax of Continue Statement**

The continue statement in Python has the following syntax:

for / while loop:  
 # statement(s)  
 if condition:  
 continue  
 # statement(s)

### Working of Python Continue Statement

The working of the continue statement in Python is depicted in the following flowchart:



|  |
| --- |
| # Python program to  # demonstrate continue  # statement    # loop from 1 to 10  for i in range(1, 11):        # If i is equals to 6,      # continue to next iteration      # without printing      if i == 6:          continue      else:          # otherwise print the value          # of i          print(i, end = " ") |

**Output:**

1 2 3 4 5 7 8 9 10

## Pass Statement in Python

## Suppose we have a [loop](https://www.programiz.com/python-programming/looping-technique) or a [function](https://www.programiz.com/python-programming/function) that is not implemented yet, but we want to implement it in the future. In such cases, we can use the pass statement.

As the name suggests pass statement simply does nothing. The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like a null operation, as nothing will happen if it is executed. Pass statements can also be used for writing empty loops. Pass is also used for empty control statements, functions, and classes.

### **Syntax of Pass Statement**

The pass statement in Python has the following syntax:

function/ condition / loop:  
 pass

**Example:**

n = 10

# use pass inside if statement

if n > 10:

pass

print('Hello')

n = 10

if n > 10:

# write code later

print('Hello')

In this example, we will use the pass statement with an empty for loop and an empty [Python function](https://www.geeksforgeeks.org/python-functions/). We just declared a function and write the pass statement in it. When we try to call this function, it will execute and not generate an error.

Then we use the pass statement with an if condition within a for loop. When the value of “i” becomes equal to ‘k’, the pass statement did nothing, and hence the letter ‘k’ is printed.

* Python3

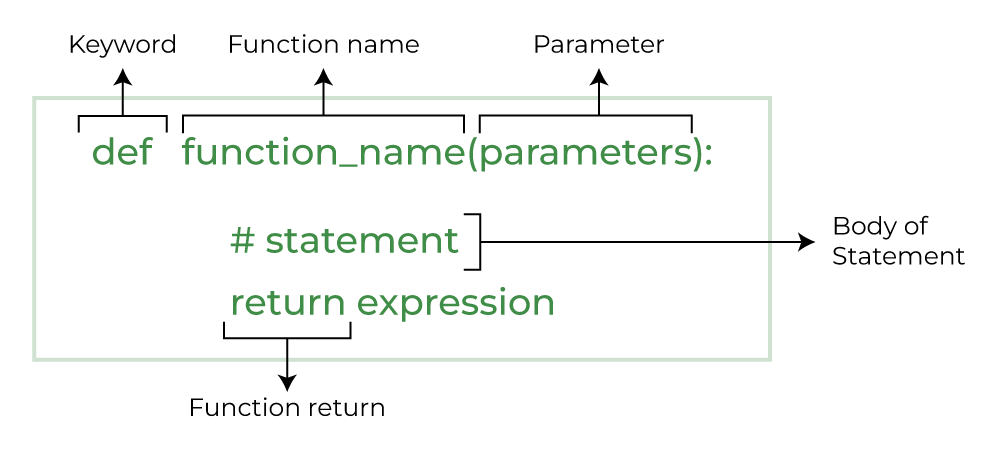
|  |
| --- |
| # Python program to demonstrate  # pass statement    s = "HELLO"    # Empty loop  for i in s:      # No error will be raised      pass    # Empty function  def fun():      pass    # No error will be raised  fun()    # Pass statement  for i in s:      if i == 'L':          print('Pass executed')          pass      print(i) |

**Output:**

h  
e  
L  
Pass executed  
L  
O

## [Python Functions](https://www.geeksforgeeks.org/python-functions/)

**Python Functions** is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.



***Note:****There are mainly two types of functions in Python.*

1. ***Built-in library function:****These are*[*Standard functions*](https://www.geeksforgeeks.org/python-built-in-functions/)*in Python that are available to use.*
2. ***User-defined function:****We can create our own functions based on our requirements.*

**Example**

* Python

|  |
| --- |
| # A simple Python function to check  # whether x is even or odd  def evenOdd(x):      if (x % 2 == 0):          print("even")      else:          print("odd")      # Driver code to call the function  evenOdd(2)  evenOdd(3) |

## ****Creating a Function in Python****

We can define a function in Python, using the **def** keyword. We can add any type of functionalities and properties to it as we require. By the following example, we can understand how to write a function in Python. In this way we can create Python function definition by using def keyword.

*# A simple Python function*

**def** fun():

print("Welcome to GFG")

### Calling a Function in Python

After creating a function in Python we can call it by using the name of the functions Python followed by parenthesis containing parameters of that particular function. Below is the example for calling def function Python.

*# A simple Python function*

**def** fun():

print("Welcome to GFG")

*# Driver code to call a function*

fun()

**Output:**

Welcome to GFG

### Python Function with Parameters

If you have experience in C/C++ or Java then you must be thinking about the return type of the function and data type of arguments. That is possible in Python as well (specifically for Python 3.5 and above).

**Python Function Syntax with Parameters**

def function\_name(parameter: data\_type) -> return\_type:  
 """Docstring"""  
 # body of the function  
 return expression

The following example uses [arguments and parameters](https://www.geeksforgeeks.org/deep-dive-into-parameters-and-arguments-in-python/) that you will learn later in this article so you can come back to it again if not understood.

**def** add(num1: int, num2: int) -> int:

*"""Add two numbers"""*

num3 = num1 + num2

**return** num3

*# Driver code*

num1, num2 = 5, 15

ans = add(num1, num2)

print(f"The addition of **{**num1**}** and **{**num2**}** results **{**ans**}**.")

**Output:**

The addition of 5 and 15 results 20.

***Note:****The following examples are defined using syntax 1, try to convert them in syntax 2 for practice.*

*# some more functions*

**def** is\_prime(n):

**if** n **in** [2, 3]:

**return** **True**

**if** (n == 1) **or** (n % 2 == 0):

**return** **False**

r = 3

**while** r \* r <= n:

**if** n % r == 0:

**return** **False**

r += 2

**return** **True**

print(is\_prime(78), is\_prime(79))

**Output:**

False True

## Python Function Arguments

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

In this example, we will create a simple function in Python to check whether the number passed as an argument to the function is even or odd.

Python3

*# A simple Python function to check*

*# whether x is even or odd*

**def** evenOdd(x):

**if** (x % 2 == 0):

print("even")

**else**:

print("odd")

*# Driver code to call the function*

evenOdd(2)

evenOdd(3)

**Output:**

even  
odd

### Types of Python Function Arguments

Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following function argument types in Python:

* **Default argument**
* **Keyword arguments (named arguments)**
* **Positional arguments**
* **Arbitrary arguments** (variable-length arguments \*args and \*\*kwargs)

### **Default Arguments**

A [default argument](https://www.geeksforgeeks.org/default-arguments-in-python/) is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments to write functions in Python.

Python3

*# Python program to demonstrate*

*# default arguments*

**def** myFun(x, y=50):

print("x: ", x)

print("y: ", y)

*# Driver code (We call myFun() with only*

*# argument)*

myFun(10)

**Output:**

x: 10  
y: 50

Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

### **Keyword Arguments**

The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

*# Python program to demonstrate Keyword Arguments*

**def** student(firstname, lastname):

print(firstname, lastname)

*# Keyword arguments*

student(firstname='Geeks', lastname='Practice')

student(lastname='Practice', firstname='Geeks')

**Output:**

Geeks Practice  
Geeks Practice

### Positional Arguments

We used the [Position argument](https://www.geeksforgeeks.org/keyword-and-positional-argument-in-python/#:~:text=age%20is%20%2020-,Positional%2DOnly%20Arguments,-Position%2Donly%20arguments) during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places, as shown in the Case-2 example below, where 27 is assigned to the name and Suraj is assigned to the age.

**def** nameAge(name, age):

print("Hi, I am", name)

print("My age is ", age)

*# You will get correct output because*

*# argument is given in order*

print("Case-1:")

nameAge("Suraj", 27)

*# You will get incorrect output because*

*# argument is not in order*

print("**\n**Case-2:")

nameAge(27, "Suraj")

**Output:**

**Case-1:**  
Hi, I am Suraj  
My age is 27  
**Case-2:**  
Hi, I am 27  
My age is Suraj

### Arbitrary Keyword  Arguments

In Python Arbitrary Keyword Arguments, [\*args, and \*\*kwargs](https://www.geeksforgeeks.org/args-kwargs-python/) can pass a variable number of arguments to a function using special symbols. There are two special symbols:

* \*args in Python (Non-Keyword Arguments)
* \*\*kwargs in Python (Keyword Arguments)

**Example 1:** Variable length non-keywords argument

*# Python program to illustrate*

*# \*args for variable number of arguments*

**def** myFun(\*argv):

**for** arg **in** argv:

print(arg)

myFun('Hello', 'Welcome', 'to', 'GeeksforGeeks')

**Output:**

Hello  
Welcome  
to  
GeeksforGeeks

**Example 2:**Variable length keyword arguments

*# Python program to illustrate*

*# \*kwargs for variable number of keyword arguments*

**def** myFun(\*\*kwargs):

**for** key, value **in** kwargs.items():

print("**%s** == **%s**" % (key, value))

*# Driver code*

myFun(first='Geeks', mid='for', last='Geeks')

**Output:**

first == Geeks  
mid == for  
last == Geeks

### Docstring

The first string after the function is called the Document string or [Docstring](https://www.geeksforgeeks.org/python-docstrings/) in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function.

**Syntax:** print(function\_name.\_\_doc\_\_)

**Example:**Adding Docstring to the function

*# A simple Python function to check*

*# whether x is even or odd*

**def** evenOdd(x):

*"""Function to check if the number is even or odd"""*

**if** (x % 2 == 0):

print("even")

**else**:

print("odd")

*# Driver code to call the function*

print(evenOdd.\_\_doc\_\_)

**Output:**

Function to check if the number is even or odd

## Python Function within Functions

A function that is defined inside another function is known as the **inner function** or **nested function**. Nested functions can access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

Python3

*# Python program to*

*# demonstrate accessing of*

*# variables of nested functions*

**def** f1():

s = HELLO INDIA'

**def** f2():

print(s)

f2()

*# Driver's code*

f1()

**Output:**

HELLO INDIA

## Anonymous Functions in Python

In Python, an [anonymous function](https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/) means that a function is without a name. As we already know the def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

Python3

*# Python code to illustrate the cube of a number*

*# using lambda function*

**def** cube(x): **return** x\*x\*x

cube\_v2 = **lambda** x : x\*x\*x

print(cube(7))

print(cube\_v2(7))

**Output:**

343  
343

## Recursive Functions in Python

**Recursion**in Python refers to when a function calls itself. There are many instances when you have to build a recursive function to solve **Mathematical and Recursive Problems.**

Using a recursive function should be done with caution, as a recursive function can become like a non-terminating loop. It is better to check your exit statement while creating a recursive function.

Python3

**def** factorial(n):

**if** n == 0:

**return** 1

**else**:

**return** n \* factorial(n - 1)

print(factorial(4))

**Output**

24

Here we have created a recursive function to calculate the factorial of the number. You can see the end statement for this function is when n is equal to 0.

## Return Statement in Python Function

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller. **The syntax for the return statement is:**

return [expression\_list]

The return statement can consist of a variable, an expression, or a constant which is returned at the end of the function execution. If none of the above is present with the return statement a None object is returned.

**Example:** Python Function Return Statement

**def** square\_value(num):

*"""This function returns the square*

*value of the entered number"""*

**return** num\*\*2

print(square\_value(2))

print(square\_value(-4))

**Output:**

4  
16

## ****Pass by Reference and Pass by Value****

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function Python, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

Python3

*# Here x is a new reference to same list lst*

**def** myFun(x):

x[0] = 20

*# Driver Code (Note that lst is modified*

*# after function call.*

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

**Output:**

[20, 11, 12, 13, 14, 15]

When we pass a reference and change the received reference to something else, the connection between the passed and received parameters is broken.

For example, consider the below program as follows:

**def** myFun(x):

*# After below line link of x with previous*

*# object gets broken. A new object is assigned*

*# to x.*

x = [20, 30, 40]

*# Driver Code (Note that lst is not modified*

*# after function call.*

lst = [10, 11, 12, 13, 14, 15]

myFun(lst)

print(lst)

**Output:**

[10, 11, 12, 13, 14, 15]

Another example demonstrates that the reference link is broken if we assign a new value (inside the function).

**def** myFun(x):

*# After below line link of x with previous*

*# object gets broken. A new object is assigned*

*# to x.*

x = 20

*# Driver Code (Note that x is not modified*

*# after function call.*

x = 10

myFun(x)

print(x)

**Output:**

10

**Exercise:**

**def** swap(x, y):

temp = x

x = y

y = temp

*# Driver code*

x = 2

y = 3

swap(x, y)

print(x)

print(y)

**Output:**

2  
3