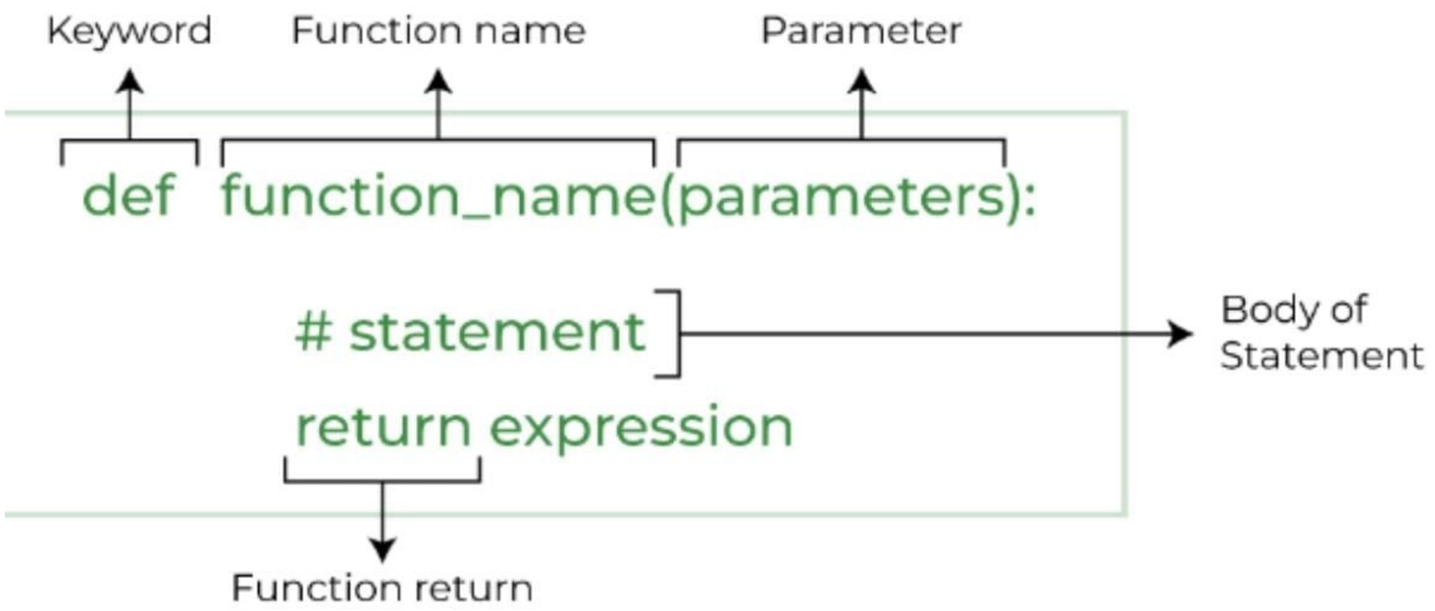
Python Functions

Python Function 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.
Some Benefits of Using Functions

- Increase Code Readability
- Increase Code Reusability

Python Function Declaration

The syntax to declare a function is:



Types of Functions in Python

There are mainly two types of functions in Python.

- **Built-in library function:** These are Standard functions in Python that are available to use.

Python provides a lot of built-in functions that eases the writing of code. In this article, you will find the list of all built-in functions of Python.

Python Built-in Functions:

Function Name	Description
Python abs()	Return the absolute value of a number
Python all()	Return true if all the elements of a given iterable(List, Dictionary, Tuple, set, etc) are True else it returns False
Python any()	Returns true if any of the elements of a given iterable(List, Dictionary, Tuple, set, etc) are True else it returns False
Python ascii()	Returns a string containing a printable representation of an object
Python bin()	Convert integer to binary string

Python bool()	Return or convert a value to a Boolean value i.e., True or False
Python bytearray()	Returns a bytearray object which is an array of given bytes
Python bytes()	Converts an object to immutable byte represented object of given size and data
Python callable()	Returns True if the object passed appears to be callable
Python chr()	Returns a string representing a character whose Unicode code point is an integer
Python classmethod()	Returns a class method for a given function
Python compile()	Returns a Python code object
Python complex()	Creates Complex Number

Python delattr()	Delete the named attribute from the object
Python dict()	Creates a Python Dictionary
Python dir()	Returns list of the attributes and methods of any object
Python divmod()	Takes two numbers and returns a pair of numbers consisting of their quotient and remainder
Python enumerate()	Adds a counter to an iterable and returns it in a form of enumerating object
Python eval()	Parses the expression passed to it and runs python expression(code) within the program
Python exec()	Used for the dynamic execution of the program
Python filter()	Filters the given sequence with the help of a function that tests each element in the sequence to be true or not

Python float()	Return a floating-point number from a number or a string
Python format()	Formats a specified value
Python frozenset()	Returns immutable frozenset
Python getattr()	Access the attribute value of an object
Python globals()	Returns the dictionary of current global symbol table
Python hasattr()	Check if an object has the given named attribute and return true if present
Python hash()	Encode the data into unrecognizable value
Python help()	Display the documentation of modules, functions, classes, keywords, etc

Python hex()	Convert an integer number into its corresponding hexadecimal form
Python id()	Return the identity of an object
Python input()	Take input from user as a string
Python int()	Converts a number in given base to decimal
Python isinstance()	Checks if the objects belong to certain class or not
Python issubclass()	Check if a class is a subclass of another class or not
Python iter()	Convert an iterable to iterator
Python len()	Returns the length of the object
Python list()	Creates a list in Python

Python locals()	Returns the dictionary of the current local symbol table
Python map()	Returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable
Python max()	Returns the largest item in an iterable or the largest of two or more arguments
Python memoryview()	Returns memory view of an argument
Python min()	Returns the smallest item in an iterable or the smallest of two or more arguments
Python next()	Receives the next item from the iterator
Python object()	Returns a new object
Python oct()	returns octal representation of integer in a string format.

Python open()	Open a file and return its object
Python ord()	Returns the Unicode equivalence of the passed argument
Python pow()	Compute the power of a number
Python print()	Print output to the console
Python property()	Create property of a class
Python range()	Generate a sequence of numbers
Python repr()	Return intable version of the object
Python reversed()	Returns an iterator that accesses the given sequence in the reverse order

Python round()	Rounds off to the given number of digits and returns the floating-point number
Python set()	Convert any of the iterable to sequence of iterable elements with distinct elements
Python setattr()	Assign the object attribute its value
Python slice()	Returns a slice object
Python sorted()	Returns a list with the elements in sorted manner, without modifying the original sequence
Python staticmethod()	Converts a message into static message
Python str()	Returns the string version of the object
Python sum()	Sums up the numbers in the list

Python super()	Returns a temporary object of the superclass
Python tuple()	Creates a tuple in Python
Python type()	Returns the type of the object
Python vars()	Returns the __dict__ attribute for a module, class, instance, or any other object
Python zip()	Maps the similar index of multiple containers
Python __import__()	Imports the module during runtime

- User-defined function: We can create our own functions based on our requirements.

Creating a function in Python

We can create a Python function using the def keyword.

Ex—1

Calling a Python Function

After creating a function we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

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).

Defining and calling a function with parameters

```
def function_name(parameter: data_type) -> return_type:
    """Docstring"""
    # body of the function
    return expression
```

Ex—2

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 to check whether the number passed as an argument to the function is even or odd.

Ex—3

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 4 types of function arguments.*

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

Let’s discuss each type in detail.

Default Arguments

A default argument 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.

Ex-4

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.

Ex—5

Positional Arguments

We used the Position argument 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.

Ex—6

Arbitrary Keyword Arguments

In Python Arbitrary Keyword Arguments, *args, and **kwargs 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)

Ex—7

Docstring

The first string after the function is called the Document string or Docstring 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__)

Ex-- 8

Python Function within Functions

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

Ex – 9

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.

Ex –10