**Functions**

In Python, a function is a block of **reusable code** that performs a **specific task** and functions also allow programmers to

* break down your program into smaller, modular pieces,
* making your code more organized,
* easier to **understand** and easier to **maintain**.
* And it starts with **def** keyword

Python functions may contain attributes or logical structure or both.

Below is the list of different types of functions in Python.

1. **Built-in Functions:**
2. **User-defined Functions:**
3. Simple Function
4. Lambda Function
5. **Other Function**
6. Higher-order Functions:

* map()
* filter()
* reduce()

1. Recursive Functions
2. Generator Functions
3. Decorator Functions
4. Instance Method
5. Class Method
6. \_\_init\_\_ Method

**Built-in Functions:**

These are in build functions that are always available in Python, no matter which modules are imported. Examples print(), len(), input(), range(), type(), str(), int(), float(), list(), dict(), sorted(), sum(), max(), min(), enumerate(), zip(), etc.

**User-defined Functions:**

“User Defined” function is a block of code that only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. In Python a function is defined using the **def** keyword:

|  |
| --- |
| Simple Function and how to call a function |
| def my\_function(a):   print("Hello from a function")  **my\_function(a)** |
| Hello from a function |

Parameters or Arguments?  
Parameters: A parameter is the variable listed inside the parentheses in the function definition.  
Arguments: An argument is the value that is sent to the function when it is called.

**Arguments: -**

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less. The same is shown below.

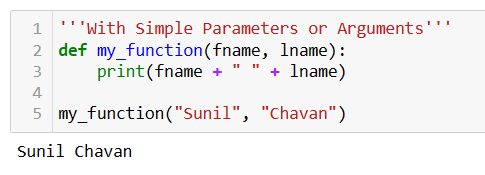
However, there are more types of arguments as are listed below.

**Different types of Arguments: -**

1. Positional Argument
2. Arbitrary Arguments, \*args
3. Keyword Arguments
4. Arbitrary Keyword Arguments, \*\*kwargs

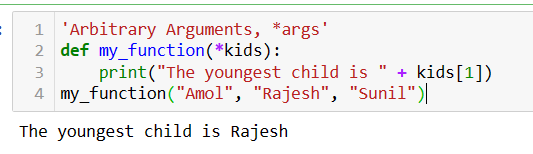
Positional Argument

Positional arguments are passed to a function based on their position in the function call.



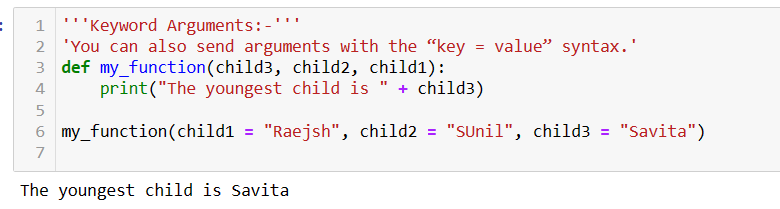
**Arbitrary Arguments, \*args:-**

When programmers don’t know the Exact No. of the Argument

****

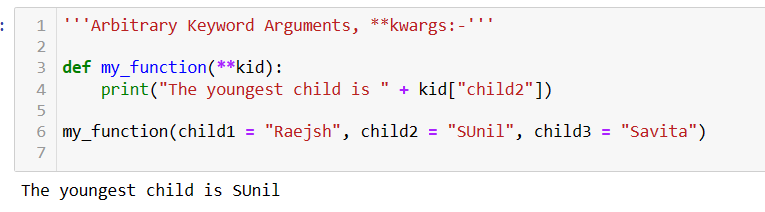
**Keyword Arguments: -**

You can also send arguments with the “key = value” syntax.



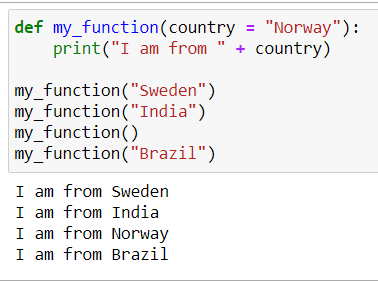
**Arbitrary Keyword Arguments, \*\*kwargs: -**

When programmers don’t know the Exact No. of Arguments in “Key=Value” Syntax. Pass “ \*\* ” before the parameter name The function will receive a dictionary of arguments



**Default Parameter Value:**

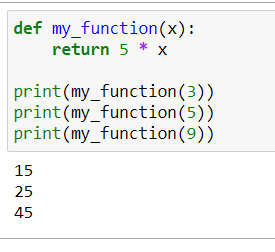
If we call the function without argument, it uses the default value, The following example shows how to use a default parameter value.



**Return Values**

To let a function, return a value, use the return statement.

In Python, the return statement is used within a function to specify the value that the function should return to the caller. When a return statement is encountered, it immediately terminates the execution of the function and sends the specified value back to the caller.

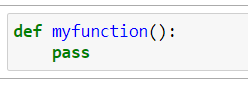


**What is the difference between return and print statements in Python functions?**

The return statement is used to exit a function and optionally return a value to the caller. It terminates the execution of the function. On the other hand, the print statement is used to display output to the console. It does not affect the flow of the function and is primarily used for debugging or displaying information.

**The pass Statement**

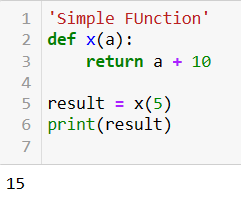
function definitions cannot be empty, but for some reason, a function definition with no content is put in the ”pass” statement to avoid getting an error.

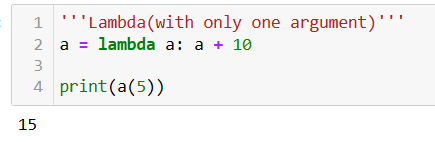


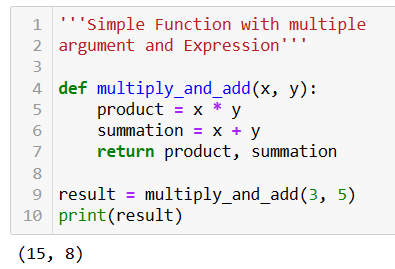
**Lambda Function(Anonymous Function):-** These are small, **inline functions** defined using the **lambda** keyword. They are typically used for short functions. Lambda functions can take **any number of arguments.**

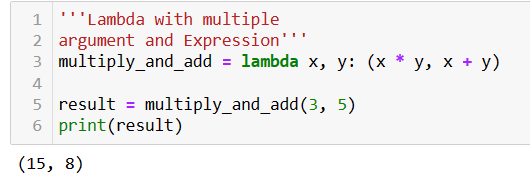
Why Use Lambda Functions?  
 Lambda function can be used as a stand-alone or an anonymous function inside another function. Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

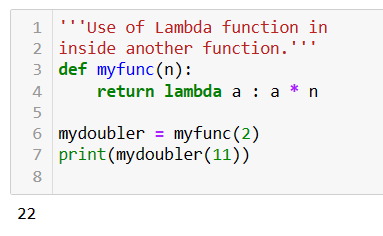
Syntax of lambda function  
**lambda arguments : expression   
lambda arguments : true\_value if condition else false\_value**











**Higher-Order Functions (filter(), map(), reduce())**

The **filter()** This function takes a function and an iterable as input(argument) and returns an iterator yielding those items of the iterable for which the function returns true.

**Here's the syntax: filter(function, iterable)**

The **map()** This function takes a function and iterables as input and applies the function to each item of the iterable(s). It returns an iterator that yields the results.

**Here's the syntax: map(function, iterable1)**

The **reduce(fun,seq)** This function is not a built-in function like filter() and map(), but it exists in the functools module. It applies a rolling computation to sequential pairs of values in an iterable. Here's the syntax:

**from functools import reduce**

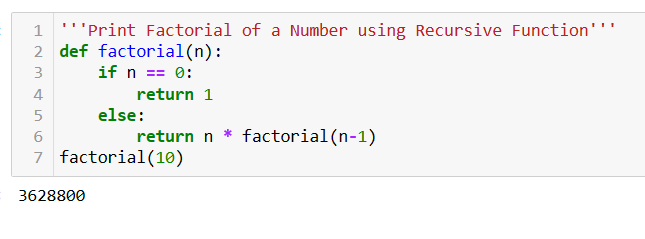
**reduce(function, iterable)**

|  |
| --- |
| **map()** |
|  |
|  |

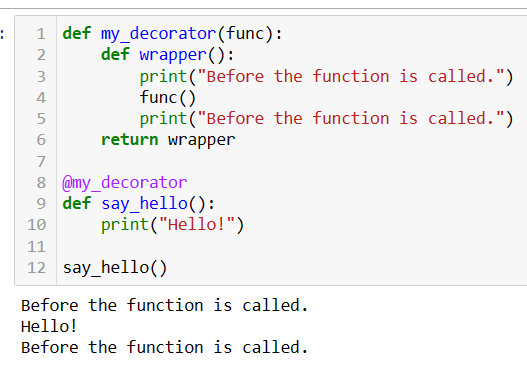
|  |
| --- |
| **reduce()** |
|  |
|  |

**Recursive Function :**

These are functions that call themselves during their execution.

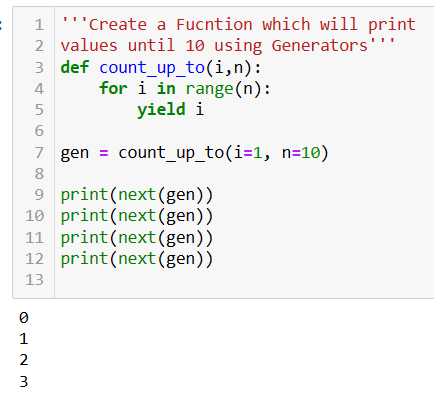


**Decorators:** A decorator is a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure.



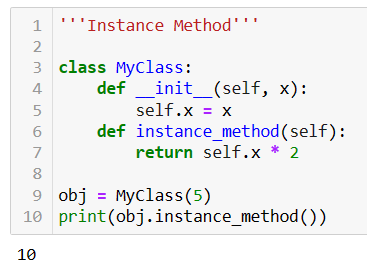
**Generator Functions:**

These are functions that use the yield keyword to return a series of values iteratively, rather than returning them all at once. Generator functions it is **Memory Efficiency**



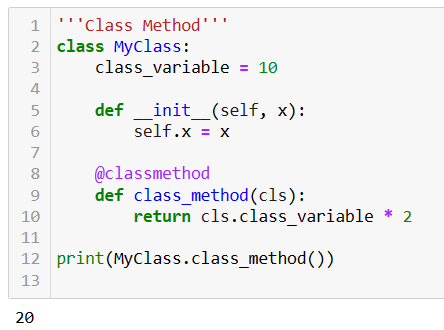
**Instance Methods:**

1. Instance methods are methods that are bound to an instance of a class.
2. They have access to the instance attributes and can modify them.
3. They are called with an instance of the class and can operate on the specific data associated with that instance.
4. Instance methods are defined by including the self-parameter as the first parameter in the method definition. self refers to the instance of the class on which the method is being called.



**Class Methods:**

1. Class methods are methods that are bound to the class itself rather than to any instance of the class.
2. They have access only to the class variables and cannot modify instance variables directly.
3. They are defined using the @classmethod decorator and take the class itself (cls) as the first parameter, conventionally named cls.
4. Class methods are often used for methods that perform operations on the class itself, like creating instances or accessing class-level variables.



**\_\_init\_\_ :**

In Python, \_\_init\_\_ is a special method (also known as a dunder method, short for "double underscore") used for initializing newly created objects. It's called the constructor method and is automatically invoked when a new class instance is created. The \_\_init\_\_ method is where you typically initialize instance variables or perform any other setup that needs to be done when an object is created.

