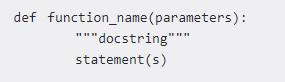
**CHAPTER 4**

**PYTHON FUNCTIONS**

**What is a function in Python?**

In Python, function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable. Furthermore, it avoids repetition and makes code reusable.

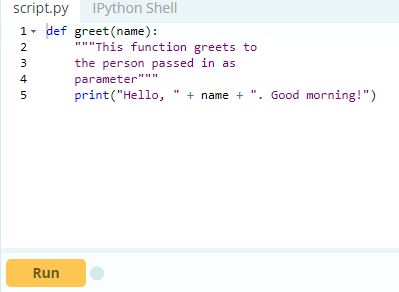
**Syntax of Function**



Above shown is a function definition which consists of following components.

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same [rules of writing identifiers in Python](https://www.programiz.com/python-programming/keywords-identifier#rules).
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional docu mentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

**Example of a function**

****

**How to call a function in python?**

Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

>>> greet('Paul')

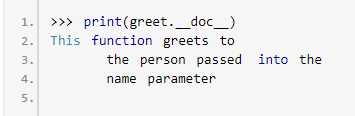
Hello, Paul. Good morning!

**Note:**Try running the above code into the Python shell to see the output.

**Docstring**

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does. Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as \_\_doc\_\_ attribute of the function. For example: Try running the following into the Python shell to see the output.



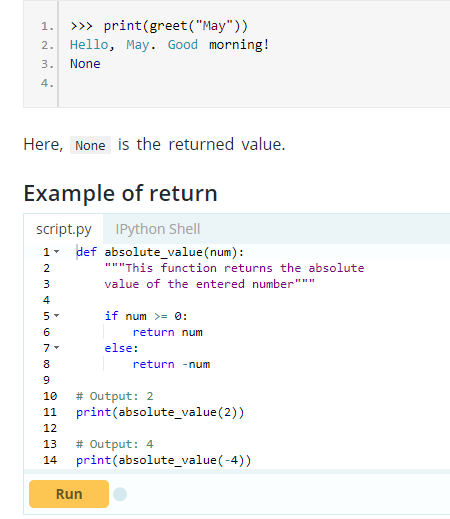
**The return statement**

The return statement is used to exit a function and go back to the place from where it was called.

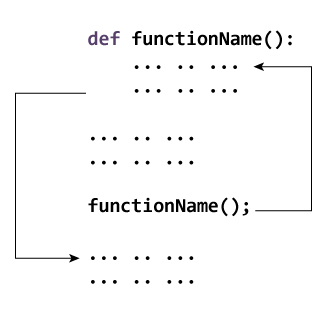
**Syntax of return**

return [expression\_list]

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

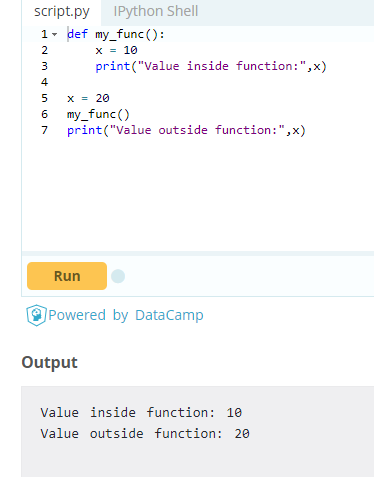


**How Function works in Python?**



**Scope and Lifetime of variables**

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope. Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes. They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls. Here is an example to illustrate the scope of a variable inside a function.



Here, we can see that the value of x is 20 initially. Even though the function my\_func()changed the value of x to 10, it did not effect the value outside the function. This is because the variable x inside the function is different (local to the function) from the one outside. Although they have same names, they are two different variables with different scope.

On the other hand, variables outside of the function are visible from inside. They have a global scope. We can read these values from inside the function but cannot change (write) them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword global.

**Types of Functions**

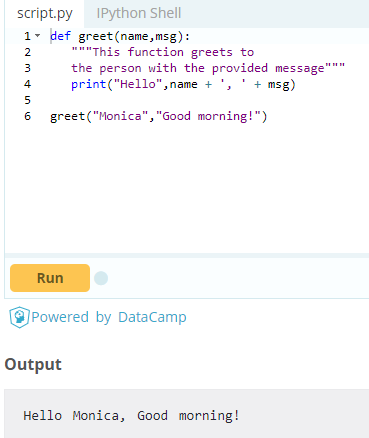
Basically, we can divide functions into the following two types:

1. [Built-in functions](https://www.programiz.com/python-programming/built-in-function) - Functions that are built into Python.
2. [User-defined functions](https://www.programiz.com/python-programming/user-defined-function) - Functions defined by the users themselves.

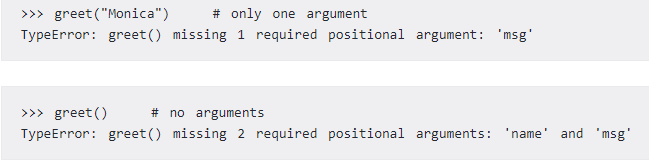
# **Python Function Arguments**

# **Arguments**

In [user-defined function](https://www.programiz.com/python-programming/user-defined-function) topic, we learned about defining a function and calling it. Otherwise, the function call will result into an error. Here is an example.



Here, the function greet() has two parameters. Since, we have called this function with two arguments, it runs smoothly and we do not get any error. If we call it with different number of arguments, the interpreter will complain. Below is a call to this function with one and no arguments along with their respective error messages.



**Variable Function Arguments**

Up until now functions had fixed number of arguments. In Python there are other ways to define a function which can take variable number of arguments. Three different forms of this type are described below.

**Python Default Arguments**

Function arguments can have default values in Python. We can provide a default value to an argument by using the assignment operator (=). Here is an example.



In this function, the parameter name does not have a default value and is required (mandatory) during a call. On the other hand, the parameter msg has a default value of "Good morning!". So, it is optional during a call. If a value is provided, it will overwrite the default value. 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.

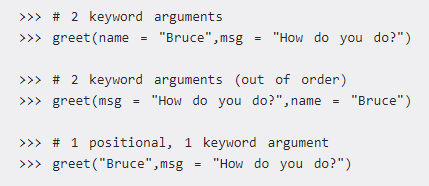
This means to say, non-default arguments cannot follow default arguments. For example, if we had defined the function header above as:

def greet(msg = "Good morning!", name):

We would get an error as:

SyntaxError: non-default argument follows default argument**Python Keyword Arguments**

When we call a function with some values, these values get assigned to the arguments according to their position. For example, in the above function greet(), when we called it as greet("Bruce","How do you do?"), the value "Bruce" gets assigned to the argument name and similarly "How do you do?"to msg. Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.



As we can see, we can mix positional arguments with keyword arguments during a function call. But we must keep in mind that keyword arguments must follow positional arguments. Having a positional argument after keyword arguments will result into errors. For example the function call as follows:



Will result into error as:

SyntaxError: non-keyword arg after keyword arg

**Python Arbitrary Arguments**

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments. In the function definition we use an asterisk (\*) before the parameter name to denote this kind of argument. Here is an example.



Here, we have called the function with multiple arguments. These arguments get wrapped up into a tuple before being passed into the function. Inside the function, we use a for loop to retrieve all the arguments back.

# **Python Recursion**

In this article, you will learn to create a recursive function; a function that calls itself.

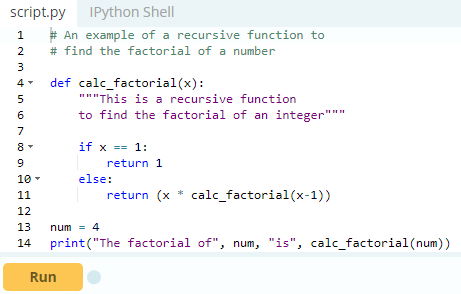
What is recursion in Python?

Recursion is the process of defining something in terms of itself. A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

**Python Recursive Function**

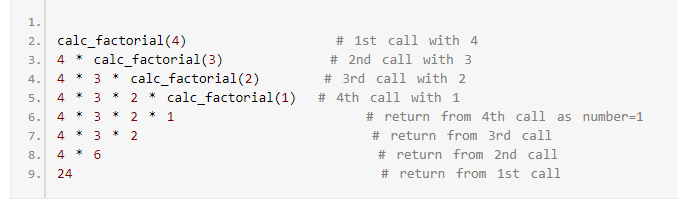
We know that in Python, a [function](https://www.programiz.com/python-programming/function) can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions. Following is an example of recursive function to find the factorial of an integer. Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720.

**Example of recursive function**

****

In the above example, calc\_factorial() is a recursive functions as it calls itself. When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function call multiples the number with the factorial of number 1 until the number is equal to one. This recursive call can be explained in the following steps.



Our recursion ends when the number reduces to 1. This is called the base condition. Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

**Advantages of Recursion**

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

**Disadvantages of Recursion**

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

# **Python Anonymous/Lambda Function**

# **What are lambda functions in Python?**

In Python, anonymous function is a [function](https://www.programiz.com/python-programming/function) that is defined without a name. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.

**How to use lambda Functions in Python?**

A lambda function in python has the following syntax.

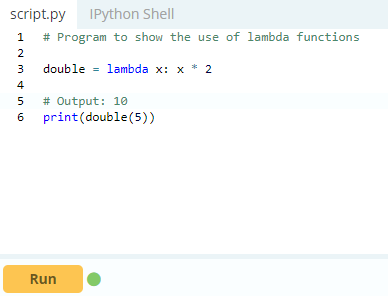
**Syntax of Lambda Function in python**

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

**Example of Lambda Function in python**

Here is an example of lambda function that doubles the input value.



In the above program, lambda x: x \* 2 is the lambda function. Here x is the argument and x \* 2 is the expression that gets evaluated and returned. This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement

double = lambda x: x \* 2

is nearly the same as

def double(x):

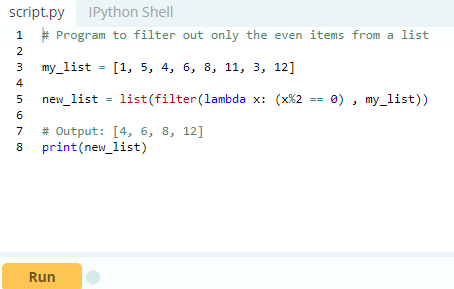
return x \* 2

### Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as [arguments](https://www.programiz.com/python-programming/function-argument)). Lambda functions are used along with built-in functions like filter(), map() etc.

#### **Example use with filter()**

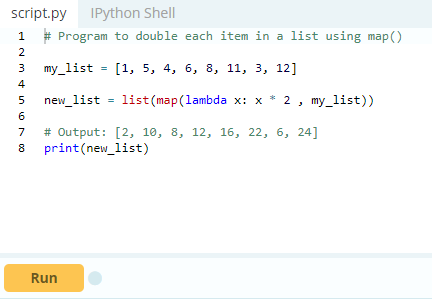
The filter() function in Python takes in a function and a list as arguments. The function is called with all the items in the list and a new list is returned which contains items for which the function evaluate to True. Here is an example use of filter() function to filter out only even numbers from a list.



#### **Example use with map()**

The map() function in Python takes in a function and a list. The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Here is an example use of map() function to double all the items in a list.

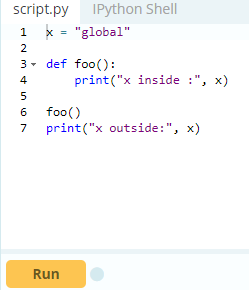


# **Python Global, Local and Nonlocal variables**

**Global Variables**

In Python, a variable declared outside of the function or in global scope is known as global variable. This means, global variable can be accessed inside or outside of the function. Let's see an example on how a global variable is created in Python.

**Example 1: Create a Global Variable**

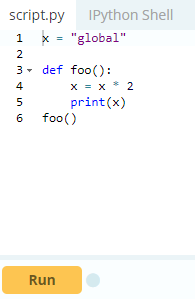


When we run the code, the will output be:

x inside : global

x outside: global

In above code, we created x as a global variable and defined a foo() to print the global variable x. Finally, we call the foo() which will print the value of x. What if you want to change value of x inside a function?



When we run the code, the will output be:

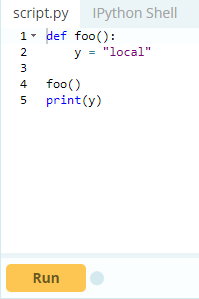
UnboundLocalError: local variable 'x' referenced before assignment

The output shows an error because Python treats x as a local variable and x is also not defined inside foo(). To make this work we use global keyword, to learn more visit [Python Global Keyword](https://www.programiz.com/python-programming/global-keyword).

**Local Variables**

A variable declared inside the function's body or in the local scope is known as local variable.

**Example 2: Accessing local variable outside the scope**

****

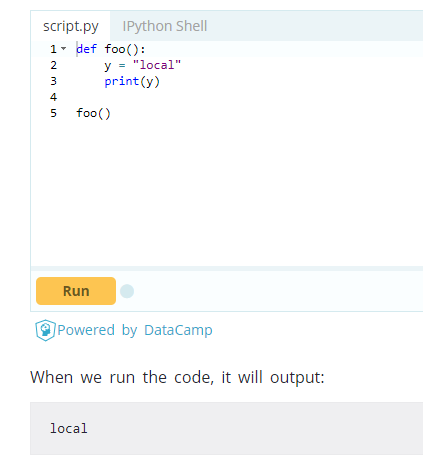
When we run the code, the will output be:

Name Error: name 'y' is not defined

The output shows an error, because we are trying to access a local variable y in a global scope whereas the local variable only works inside foo() or local scope. Let's see an example on how a local variable is created in Python.

**Example 3: Create a Local Variable**

Normally, we declare a variable inside the function to create a local variable.

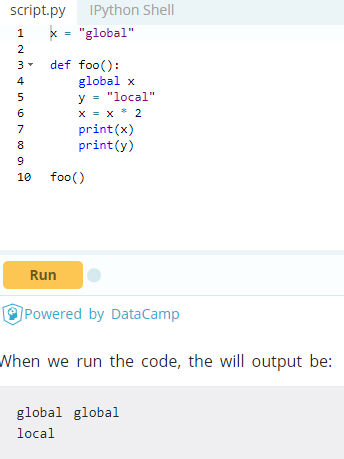


Let's take a look to the [earlier problem](https://www.programiz.com/python-programming/global-local-nonlocal-variables#change-x) where x was a global variable and we wanted to modify x inside foo().

**Global and local variables**

Here, we will show how to use global variables and local variables in the same code.

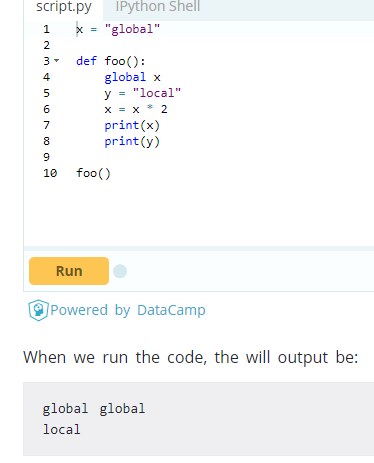
**Example 4: Using Global and Local variables in same code**

****

In the above code, we declare x as a global and y as a local variable in the foo(). Then, we use multiplication operator \* to modify the global variable x and we print both x and y.

After calling the foo(), the value of x becomes global global because we used the x \* 2to print two times global. After that, we print the value of local variable y i.e local.

**Example 5: Global variable and Local variable with same name**

****

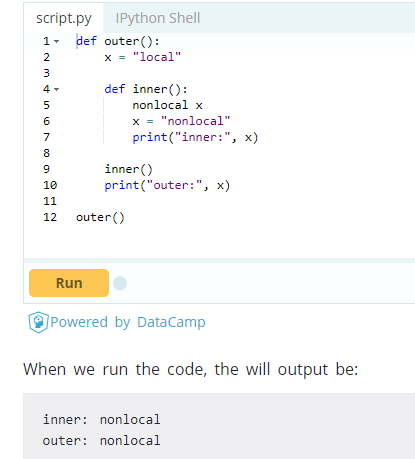
In above code, we used same name x for both global variable and local variable. We get different result when we print same variable because the variable is declared in both scopes, i.e. the local scope inside foo() and global scope outside foo().

When we print the variable inside the foo() it outputs local x: 10, this is called local scope of variable. Similarly, when we print the variable outside the foo(), it outputs global x: 5, this is called global scope of variable.

**Nonlocal Variables**

Nonlocal variable are used in nested function whose local scope is not defined. This means, the variable can be neither in the local nor the global scope. Let's see an example on how a global variable is created in Python. We use nonlocal keyword to create nonlocal variable.

**Example 6: Create a nonlocal variable**

****

In the above code there is a nested function inner(). We use nonlocal keyword to create nonlocal variable. The inner() function is defined in the scope of another function outer().

Note : If we change value of nonlocal variable, the changes appears in the local variable.

# **Python Global Keyword**

In this article, you’ll learn about the global keyword, global variable and when to use global keywords. Before reading this article, make sure you have got some basics of [Python Global, Local and Nonlocal Variables](https://www.programiz.com/python-programming/global-local-nonlocal-variables).

## Introduction to global Keyword

In Python, global keyword allows you to modify the variable outside of the current scope. It is used to create a global variable and make changes to the variable in a local context.

### 

### Rules of global Keyword

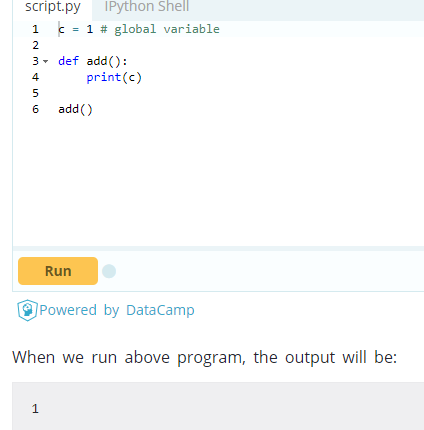
The basic rules for global keyword in Python are:

* When we create a variable inside a function, it’s local by default.
* When we define a variable outside of a function, it’s global by default. You don’t have to use global keyword.
* We use global keyword to read and write a global variable inside a function.
* Use of global keyword outside a function has no effect

### Use of global Keyword (With Example)

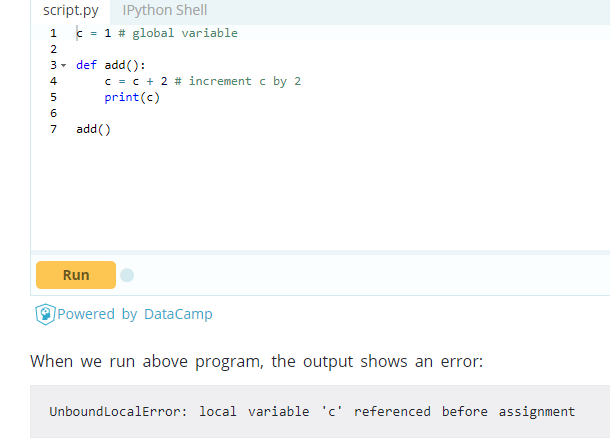
Let’s take an example.

#### **Example 1: Accessing global Variable From Inside a Function**



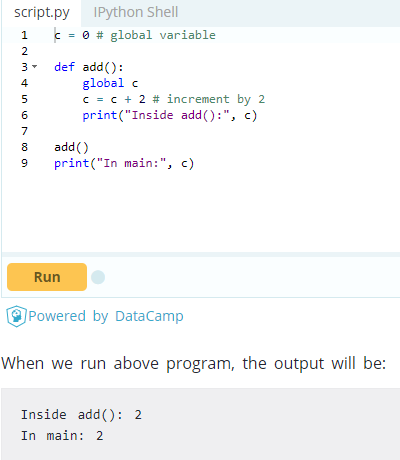
However, we may have some scenarios where we need to modify the global variable from inside a function.

#### **Example 2: Modifying Global Variable From Inside the Function**



This is because we can only access the global variable but cannot modify it from inside the function. The solution for this is to use the global keyword**Example 3: Changing Global Variable From Inside a Function using global**

In the above program, we define c as a global keyword inside the add() function. Then, we increment the variable c by 1, i.e c = c + 2. After that, we call the add()function. Finally, we print global variable c. As we can see, change also occured on the global variable outside the function, c = 2.



## Global Variables Across Python Modules

In Python, we create a single module config.py to hold global variables and share information across Python modules within the same program. Here is how we can share global variable across the python modules.

#### **Example 4 : Share a global Variable Across Python Modules**

Create a config.py file, to store global variables

1. a = 0
2. b = "empty"

Create a update.py file, to change global variables

1. import config
2. config.a = 10
3. config.b = "alphabet"

Create a main.py file, to test changes in value

1. import config
2. import update
3. print(config.a)
4. print(config.b)

When we run the main.py file, the output will be

10

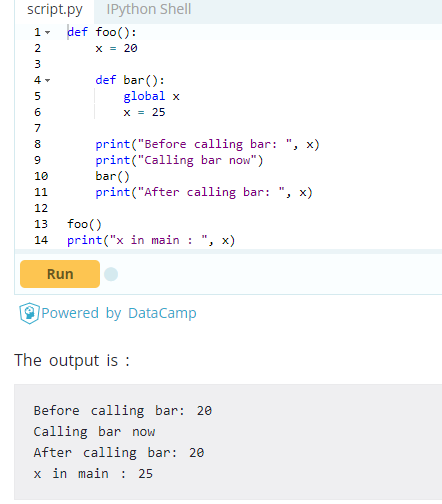
alphabet

In the above, we create three files: config.py, update.py and main.py. The module config.py stores global variables of a and b. In update.py file, we import the config.py module and modify the values of a and b. Similarly, in main.py file we import both config.py and update.py module. Finally, we print and test the values of global variables whether they are changed or not.

## Global in Nested Functions

Here is how you can use a global variable in nested function.

#### Example 5: Using a Global Variable in Nested Function



In the above program, we declare global variable inside the nested function bar(). Inside foo() function, x has no effect of global keyword. Before and after calling bar(), the variable x takes the value of local variable i.e x = 20. Outside of the foo() function, the variable x will take value defined in the bar() function i.e x = 25.

This is because we have used global keyword in x to create global variable inside the bar() function (local scope). If we make any changes inside the bar() function, the changes appears outside the local scope, i.e. foo().

# **Python Modules**

# Modules refer to a file containing Python statements and definitions. A file containing Python code, for e.g.: example.py, is called a module and its module name would be example. We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code. We can define our most used functions in a module and import it, instead of copying their definitions into different programs. Let us create a module. Type the following and save it as example.py.

1. # Python Module example
2. def add(a, b):
3. """This program adds two
4. numbers and return the result"""
5. result = a + b
6. return result

Here, we have defined a [function](https://www.programiz.com/python-programming/function) add() inside a module named example. The function takes in two numbers and returns their sum.

**How to import modules in Python?**

We can import the definitions inside a module to another module or the interactive interpreter in Python. We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

>>> import example

This does not enter the names of the functions defined in example directly in the current symbol table. It only enters the module name example there. Using the module name we can access the function using the dot.operator. For example:

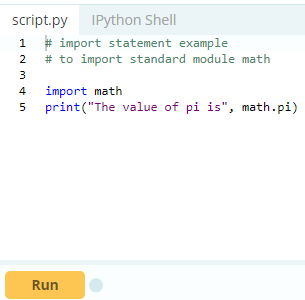
>>> example.add(4,5.5)

9.5

Python has a ton of standard modules available. You can check out the full list of [Python standard modules](http://docs.python.org/3/py-modindex.html) and what they are for. These files are in the Lib directory inside the location where you installed Python. Standard modules can be imported the same way as we import our user-defined modules. There are various ways to import modules. They are listed as follows.

**Python import statement**

We can import a module using import statement and access the definitions inside it using the dot operator as described above. Here is an example.

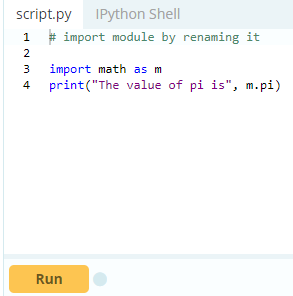


When you run the program, the output will be:

The value of pi is 3.141592653589793

**Import with renaming**

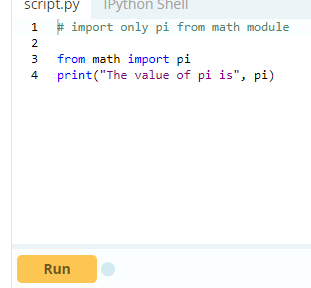
We can import a module by renaming it as follows.

****

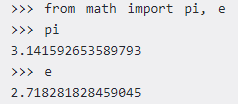
We have renamed the math module as m. This can save us typing time in some cases. Note that the name math is not recognized in our scope. Hence, math.pi is invalid, m.pi is the correct implementation.

**Python from...import statement**

We can import specific names from a module without importing the module as a whole. Here is an example.

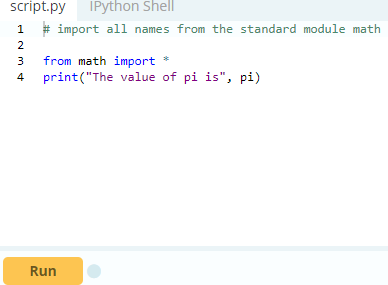


We imported only the attribute pi from the module. In such case we don't use the dot operator. We could have imported multiple attributes as follows.



**Import all names**

We can import all names(definitions) from a module using the following construct.

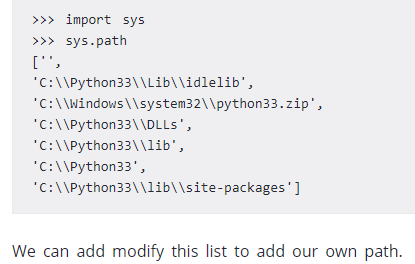


We imported all the definitions from the math module. This makes all names except those beginning with an underscore, visible in our scope. Importing everything with the asterisk (\*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

**Python Module Search Path**

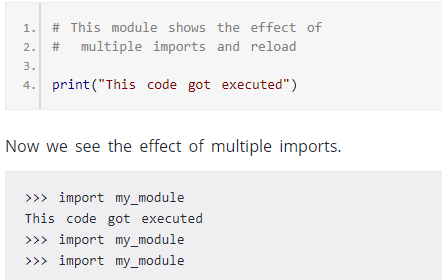
While importing a module, Python looks at several places. Interpreter first looks for a built-in module then (if not found) into a list of directories defined in sys.path. The search is in this order.

* The current directory.
* PYTHONPATH (an environment variable with a list of directory).
* The installation-dependent default directory.

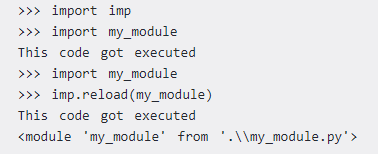


**Reloading a module**

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works. Suppose we have the following code in a module named my\_module.

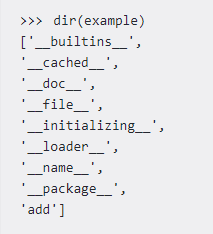


We can see that our code got executed only once. This goes to say that our module was imported only once. Now if our module changed during the course of the program, we would have to reload it.One way to do this is to restart the interpreter. But this does not help much. Python provides a neat way of doing this. We can use the reload() function inside the impmodule to reload a module. This is how its done.

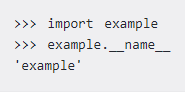


**The dir() built-in function**

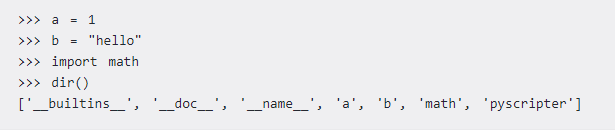
We can use the dir() function to find out names that are defined inside a module. For example, we have defined a function add() in the module example that we had in the beginning.



Here, we can see a sorted list of names (along with add). All other names that begin with an underscore are default Python attributes associated with the module (we did not define them ourself). For example, the \_\_name\_\_ attribute contains the name of the module.



All the names defined in our current namespace can be found out using the dir() function without any arguments.



# **Python Package**

We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and [modules](https://www.programiz.com/python-programming/modules) for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear. Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

A directory must contain a file named \_\_init\_\_.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file. Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.



**Importing module from a package**

We can import modules from packages using the dot (.) operator.

For example, if want to import the start module in the above example, it is done as follows.

  
Now if this module contains a [function](https://www.programiz.com/python-programming/function) named select\_difficulty(), we must use the full name to reference it.



If this construct seems lengthy, we can import the module without the package prefix as follows.



We can now call the function simply as follows.



Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.



Now we can directly call this function.



Although easier, this method is not recommended. Using the full [namespace](https://www.programiz.com/python-programming/namespace) avoids confusion and prevents two same identifier names from colliding.

While importing packages, Python looks in the list of directories defined in sys.path, similar as for [module search path](https://www.programiz.com/python-programming/modules#search).