Python Functions, Modules, and Packages

Contents

[Python Functions 3](#_Toc98167600)

[Python Lambda Functions 5](#_Toc98167601)

[Python Modules 6](#_Toc98167602)

[Built-in Modules 6](#_Toc98167603)

[Python Modules and OO Concepts 8](#_Toc98167604)

[Is a class library an API? 8](#_Toc98167605)

[Life of a Python package / module / object 9](#_Toc98167606)

[PIP – Package Installer for Python into your environment (system) 9](#_Toc98167607)

[Import (bind) package to your source code 9](#_Toc98167608)

[Create and Access a Python Package 10](#_Toc98167609)

[Creating and Exploring Packages 10](#_Toc98167610)

[To create a package in Python, we need to follow these three simple steps: 10](#_Toc98167611)

[Example of Creating Package 10](#_Toc98167612)

[Various ways of Accessing the Packages 12](#_Toc98167613)

[References 15](#_Toc98167614)

[5 Examples of Excellent API Documentation (and Why We Think So) 15](#_Toc98167615)

Python Functions, Modules, Classes, Objects, Packages, and Inheritance

# Python Functions

|  |  |
| --- | --- |
| A function is a block of code which only runs when it is called.  You can pass data, known as parameters, into a function.  A function can return data as a result. |  |
| **Create a Function**  def my\_function():   print("Hello from a function") | **Call a Function**  def my\_function():   print("Hello from a function")  **my\_function()** |
| **Passing Argument / Parameters to a Function**   * Information can be passed into functions as arguments. * Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma * The terms parameter and argument can be used for the same thing: information that are passed into a function.   **From a function's perspective:**   * **Parameter** - the variable listed inside the parentheses in the function definition. * **Argument** - the value that is sent to the function when it is called. | def my\_function(**fname**):   print(fname + " Refsnes")  my\_function(**"Emil"**) my\_function(**"Tobias"**) my\_function(**"Linus"**) |
| **Return a value** | def my\_function(x):    return x + 5 |
| 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 | Traceback (most recent call last):  File "demo\_function\_args\_error.py", line 4, in <module>  my\_function("Emil")  TypeError: my\_function() missing 1 required positional argument: 'lname' |
| **Unknown number of (Arbitrary) Arguments, \*args:**   * If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition. * This way the function will receive a tuple of arguments, and can access the items accordingly: * **Arbitrary Arguments are often shortened to \*args in Python documentations.** | If the number of arguments is unknown, add a \* before the parameter name to create a tuple list argument:  def my\_function(\*kids):   print("The youngest child is " + kids[2])  my\_function("Emil", "Tobias", "Linus") |
| **Keyword Arguments**   * You can also send arguments with the key = value pair syntax. * This way the order of the arguments does not matter. * **The phrase Keyword Arguments are often shortened to kwargs in Python documentations.** | def my\_function(child3, child2, child1):   print("The youngest child is " + child3)  my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus") |
| **Arbitrary Keyword Arguments, \*\*kwargs**   * If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition. * This way the function will receive a dictionary of arguments, and can access the items accordingly: | If the number of keyword arguments is unknown, add a double \*\* before the parameter name:  def my\_function(\*\*kid):   print("His last name is " + kid["lname"])  my\_function(fname = "Tobias", lname = "Refsnes") |
| **Default Parameter Value**   * The following example shows how to use a default parameter value. * If we call the function without argument, it uses the default value: | def my\_function(**country = "Norway"**):   print("I am from " + country)  my\_function("Sweden") my\_function("India") my\_function() my\_function("Brazil") |
| **Passing a List as an Argument**   * You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function. * E.g. if you send a List as an argument, it will still be a List when it reaches the function: | def my\_function(food):  for x in food:  print(x)  fruits = ["apple", "banana", "cherry"]  my\_function(fruits) |
| **Return values**   * To let a function return a value, use the return statement: | def my\_function(x):  return x + 5  print(my\_function(3)) print(my\_function(5)) print(my\_function(9)) |
| **pass Statement**   * function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error. | def myfunction():   pass  SyntaxError: unexpected EOF while parsing |
| **Recursion**   * Function recursion is defined as a function can call itself. * This has the benefit of meaning that you can loop through data to reach a result. | def tri\_recursion(k):  if(k > 0):  result = k + tri\_recursion(k - 1)  print(result)  else:  result = 0  return result  print("\n\nRecursion Example Results")  tri\_recursion(6) |

# Python Lambda Functions

|  |  |
| --- | --- |
| A lambda function is a small anonymous function.  A lambda function can take any number of arguments, but can only have one expression. | **Syntax**  Var = lambda arguments : expression |
| Example:  x = lambda a : a + 10 print(x(5))  x = lambda a, b, c : a + b + c print(x(5, 6, 2)) | Functions within Functions:  def myfunc(n):   return lambda a : a \* n  mydoubler = myfunc(2) mytripler = myfunc(3)  print(mydoubler(11)) print(mytripler(11)) |

# Python Modules

|  |  |
| --- | --- |
| * **Python module is a collection of one or many functions** * **A module is a text file with a file extension of .py** * **Modules are also called code library** * **Modules can be used by other modules with the import statement** * **Functions within a module can be references with the syntax module\_name.function\_name.** * **The module variables are of all types (arrays, dictionaries, objects etc):** | Save this code in the file mymodule.py  person1 = {  "name": "John",  "age": 36,  "country": "Norway"  }  Import the module named mymodule, and access the person1 dictionary:  import mymodule  a = mymodule.person1["age"]  print(a) |
| * **You can create an alias when you import a module, by using the as keyword:** | Create an alias for mymodule called mx:  import mymodule as mx  a = mx.person1["age"] print(a) |

## Built-in Modules

|  |  |
| --- | --- |
| * Import and use the **platform** module: | import platform  x = platform.system()  print(x) |
| **dir() Function of platform module:**   * built-in function to list all the function names (or variable names) in a module. * The dir() function can be used on all modules, also the ones you create yourself. | import platform  x = dir(platform)  print(x) |
| * import only parts from a module, by using the from keyword. * **When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"]** | **mymodule has one function and one dictionary:**  def greeting(name):  print("Hello, " + name)  person1 = {  "name": "John",  "age": 36,  "country": "Norway"  }  **Import only the person1 dictionary from the module:**  from mymodule import person1  print (person1["age"]) |

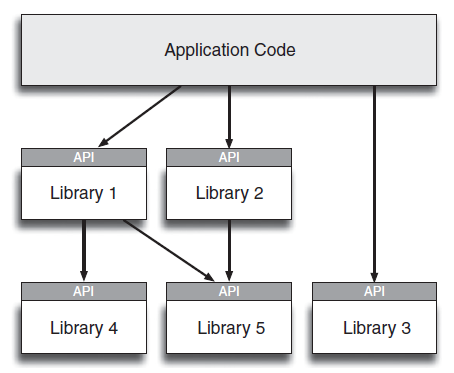
<https://www.w3schools.com/python/python_modules.asp>

C:\Users\My Name>python demo\_module1.py

## Python Modules and OO Concepts

### Is a class library an API?

An API is the way you access a library (or any set of classes). In object-oriented programming , a class library is a collection of prewritten classes or coded templates, any of which can be specified and used by a programmer when developing an application program. you could use a library in a variety of projects.



# Life of a Python package / module / object

## PIP – Package Installer for Python into your environment (system)

|  |  |
| --- | --- |
| “package” is a collection of modules to be installed (i.e. as a synonym for a distribution) in your environment for imported and called by your applications.  Your applications are dependent on having the proper functions needed to build, run and deploy that application within the Python environment. | Python “Virtual Environments” allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally. If you are looking to safely install global command line tools, see Installing stand alone command line tools |
| >> python -m pip install "SomePackage" | [Installing Packages](https://packaging.python.org/tutorials/installing-packages/) |

## Import (bind) package to your source code

|  |  |
| --- | --- |
| * Import modules in your Python source code makes the imported models available (binded) to you applicaqiton based on environment, privileges, version levels, container/module paths/locations. * Python code in one module gains access to the code in another module by the process of importing it. * import statement is the most common way of invoking the import capabilities, but it is not the only way. Functions such as importlib.import\_module() and built-in \_\_import\_\_() can also be used to invoke the import capabilities. |  |
|  | [The import system](https://docs.python.org/3/reference/import.html) |

# Create and Access a Python Package

<https://www.geeksforgeeks.org/create-access-python-package/>

Packages are a way of structuring many packages and modules which helps in a well-organized hierarchy of data set, making the directories and modules easy to access. Just like there are different drives and folders in an OS to help us store files, similarly packages help us in storing other sub-packages and modules, so that it can be used by the user when necessary.

## Creating and Exploring Packages

To tell Python that a particular directory is a package, we create a file named \_\_init\_\_.py inside it and then it is considered as a package and we may create other modules and sub-packages within it. This \_\_init\_\_.py file can be left blank or can be coded with the initialization code for the package.

To create a package in Python, we need to follow these three simple steps:

1. First, we create a directory and give it a package name, preferably related to its operation.
2. Then we put the classes and the required functions in it.
3. Finally we create an \_\_init\_\_.py file inside the directory, to let Python know that the directory is a package.

## Example of Creating Package

Let’s look at this example and see how a package is created. Let’s create a package named Cars and build three modules in it namely, Bmw, Audi and Nissan.

1. **First we create a directory and name it Cars.**
2. **Then we need to create modules**. To do this we need to create a file with the name Bmw.py and create its content by putting this code into it.

|  |
| --- |
| # Python code to illustrate the Modules  class Bmw:      # First we create a constructor for this class      # and add members to it, here models      def \_\_init\_\_(self):          self.models = ['i8', 'x1', 'x5', 'x6']        # A normal print function      def outModels(self):          print('These are the available models for BMW')          for model in self.models:              print('\t%s ' % model) |

1. Then we create another file with the name Audi.py and add the similar type of code to it with different members.

|  |
| --- |
| # Python code to illustrate the Module  class Audi:      # First we create a constructor for this class      # and add members to it, here models      def \_\_init\_\_(self):          self.models = ['q7', 'a6', 'a8', 'a3']        # A normal print function      def outModels(self):          print('These are the available models for Audi')          for model in self.models:              print('\t%s ' % model) |

1. Then we create another file with the name Nissan.py and add the similar type of code to it with different members.

|  |
| --- |
| # Python code to illustrate the Module  class Nissan:      # First we create a constructor for this class      # and add members to it, here models      def \_\_init\_\_(self):          self.models = ['altima', '370z', 'cube', 'rogue']        # A normal print function      def outModels(self):          print('These are the available models for Nissan')          for model in self.models:              print('\t%s ' % model) |

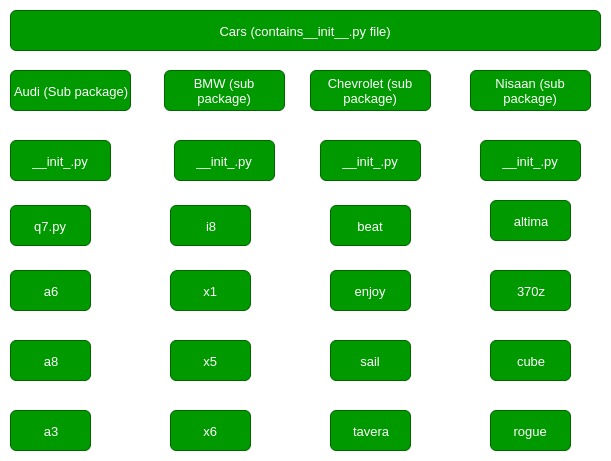
1. **Finally we create the \_\_init\_\_.py file.** This file will be placed inside Cars directory and can be left blank or we can put this initialisation code into it.

|  |
| --- |
| from Bmw import Bmw  from Audi import Audi  from Nissan import Nissan |

1. Now, let’s use the package that we created. To do this make a sample.py file in the same directory where Cars package is located and add the following code to it:

|  |
| --- |
| # Import classes from your brand new package  from Cars import Bmw  from Cars import Audi  from Cars import Nissan    # Create an object of Bmw class & call its method  ModBMW = Bmw()  ModBMW.outModels()    # Create an object of Audi class & call its method  ModAudi = Audi()  ModAudi.outModels()    # Create an object of Nissan class & call its method  ModNissan = Nissan()  ModNissan.outModels() |

## Various ways of Accessing the Packages

Let’s look at this example and try to relate packages with it and how can we access it.  


* 1. **import in Packages**  
     Suppose the cars and the brand directories are packages. For them to be a package they all must contain \_\_init\_\_.py file in them, either blank or with some initialization code. Let’s assume that all the models of the cars to be modules. Use of packages helps importing any modules, individually or whole.  
     Suppose we want to get Bmw i8. The syntax for that would be:

'import' Cars.Bmw.x5

While importing a package or sub packages or modules, Python searches the whole tree of directories looking for the particular package and proceeds systematically as programmed by the dot operator.  
If any module contains a function and we want to import that. For e.g., a8 has a function get\_buy(1) and we want to import that, the syntax would be:

import Cars.Audi.a8

Cars.Audi.a8.get\_buy(1)

While using just the import syntax, one must keep in mind that the last attribute must be a subpackage or a module, it should not be any function or class name.

* 1. **‘from…import’ in Packages**  
     Now, whenever we require using such function we would need to write the whole long line after importing the parent package. To get through this in a simpler way we use ‘from’ keyword. For this we first need to bring in the module using ‘from’ and ‘import’:

from Cars.Audi import a8

Now we can call the function anywhere using

a8.get\_buy(1)

There’s also another way which is less lengthy. We can directly import the function and use it wherever necessary. First import it using:

from Cars.Audi.a8 import get\_buy

Now call the function from anywhere:

get\_buy(1)

* 1. **‘from…import \*’ in Packages**  
     While using the **from…import** syntax, we can import anything from submodules to class or function or variable, defined in the same module. If the mentioned attribute in the import part is not defined in the package then the compiler throws an ImportError exception.  
     Importing sub-modules might cause unwanted side-effects that happens while importing sub-modules explicitly. Thus we can import various modules at a single time using \* syntax. The syntax is:

from Cars.Chevrolet import \*

This will import everything i.e., modules, sub-modules, function, classes, from the sub-package.

This article is contributed by **Chinmoy Lenka**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](http://www.contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

# References

### [5 Examples of Excellent API Documentation (and Why We Think So)](https://nordicapis.com/5-examples-of-excellent-api-documentation/)

[Framework vs Library vs Platform vs API vs SDK vs Toolkits vs IDE](https://shashvatshukla.medium.com/framework-vs-library-vs-platform-vs-api-vs-sdk-vs-toolkits-vs-ide-50a9473999db)