

Python Functions, Modules & Packages

Python Built-in Function:

- 1) Python has several functions that are readily available for use.
- 2) Python has a set of built - in function.

Types:

1) abs()

- The abs() function returns the absolute value of the specified number.

Syntax: abs(*n*)

2) all()

- The all() function returns True if all items in an iterable are true, otherwise it returns False.
- If the iterable object is empty, the all() function also returns True.

Syntax: all(*iterable*)

3) boolen()

Return the boolean value of 1

```
x = bool(1)
```

4) dict()

Create a dictionary containing personal information

```
x = dict(name = "John", age = 36, country = "Norway")
```

5) len()

Return the length of object.

```
Mylist=["apple","banana","cherry"]  
x = len(mylist)
```

6) bin ()

This function returns the binary value of a number.

Code:

```
a=5  
print(bin(a))
```

Output: 0b101

7) round()

It gives a roundoff value for a number, i.e. gives, the nearest integer value for a number. This function accepts one argument, either decimal, float, or integer, and gives roundoff output.

Code:

```
print(round(4.5))  
print(round(-7.7))
```

8) bool()

This function returns the Boolean value of an object.

Code:

```
print(bool(0))  
print(bool(-4.5))  
print(bool(None))  
print(bool("False"))
```

Output:

```
False  
True  
False
```

9) len ()

This function gives the length of the object as an output.

Syntax –

len([object])

Here objects must be either a sequence or a collection.

Note- The interpreter throws an error in case it encounters no argument given to the function.

Code:

```
stringobj = 'PALINDROME'
print(len(stringobj))
tupleobj = ('a', 'e', 'i', 'o', 'u')
print(len(tupleobj))
listobj = ['1', '2', '3', 'o', '10u']
print(len(listobj))
```

Output:

```
10
5
5
```

10. max()

This function returns the largest number in the given iterable object or the maximum of two or more numbers given as arguments.

Syntax –

max(iterable) or max(num1,num2...)

Here iterable can be a list, tuple, dictionary, or any sequence or collection.

Code:

```
num = [11, 13, 12, 15, 14]
print('Maximum is:', max(num))
```

Output:

15

Note- In case no arguments are given to the function, then ValueError is thrown by the interpreter.

11. min()

This function returns the minimum value from the collection object or the values defined as arguments.

Syntax –

```
min([iterable])
```

Code:

```
print(min(2,5,3,1,0,99))
sampleObj = ['B','a','t','A']
print(min(sampleObj))
```

Output:

0

Data Conversion Function:

Function Converting what to what

Example

int() string, floating point → integer

```
>>> int('2014')
2014
>>> int(3.141592)
3
```

float() string, integer → floating point number

```
>>> float('1.99')
1.99
>>> float(5)
5.0
```

str() integer, float, list, tuple, dictionary → string

```
>>> str(3.141592)
'3.141592'
>>> str([1,2,3,4])
'[1, 2, 3, 4]'
```

list() string, tuple, set, dictionary → list

```
>>> list('Mary') #list of char in 'Mary'
['M', 'a', 'r', 'y']
>>> list((1,2,3,4))#(1,2,3,4) is a tuple
[1, 2, 3, 4]
>>> list({1, 2, 3}) # {1, 2, 3} is a set
[1, 2, 3]
```

tuple() string, list, set → tuple

```
>>> tuple('Mary')
('M', 'a', 'r', 'y')
>>> tuple([1,2,3,4]) #[]for list, ( ) for tuple
(1, 2, 3, 4)
>>> set('alabama') #unique character set from a string
{'b', 'm', 'l', 'a'}
```

set() string, list, tuple → set

```
>>> set([1, 2, 3, 3, 3, 2]) #handy for
removing duplicates from list
{1, 2, 3}
```

Math Function:

List of Functions in Python Math Module

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
ldexp(x, i)	Returns $x * (2^{**i})$
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e^{**x}
expm1(x)	Returns $e^{**x} - 1$
log(x[, b])	Returns the logarithm of x to the base b (defaults to e)
log1p(x)	Returns the natural logarithm of 1+x
log2(x)	Returns the base-2 logarithm of x
log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
acos(x)	Returns the arc cosine of x
asin(x)	Returns the arc sine of x
atan(x)	Returns the arc tangent of x
atan2(y, x)	Returns $\text{atan}(y / x)$
cos(x)	Returns the cosine of x
hypot(x, y)	Returns the Euclidean norm, $\sqrt{x^2 + y^2}$
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
degrees(x)	Converts angle x from radians to degrees
radians(x)	Converts angle x from degrees to radians
acosh(x)	Returns the inverse hyperbolic cosine of x
asinh(x)	Returns the inverse hyperbolic sine of x

atanh(x)	Returns the inverse hyperbolic tangent of x
cosh(x)	Returns the hyperbolic cosine of x
sinh(x)	Returns the hyperbolic cosine of x
tanh(x)	Returns the hyperbolic tangent of x
erf(x)	Returns the error function at x
erfc(x)	Returns the complementary error function at x
gamma(x)	Returns the Gamma function at x
lgamma(x)	Returns the natural logarithm of the absolute value of the Gamma function at x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

Python Function:

- 1) Function are sub program which are used to compute a value or perform a task.
- 2) Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.
- 3) Python allows us to divide a large program into the basic building blocks known as a function.
- 4) A function can be called multiple times to provide reusability and modularity to the Python program.
- 5) The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.
- 6) Python provide us various inbuilt functions like **range()** or **print()**. Although, the user can create its functions, which can be called user-defined functions.**Stay**

There are mainly two types of functions.

- **User-define functions** - The user-defined functions are those define by the **user** to perform the specific task.
- **Built-in functions** - The built-in functions are those functions that are **pre-defined** in Python.

Advantage of Functions in Python:

- 1) Using functions, we can avoid rewriting the same logic/code again and again in a program.
- 2) We can call Python functions multiple times in a program and anywhere in a program.
- 3) Function facilitates ease of code maintenance.
- 4) Divide large task into many small task so it will help you to debug code.
- 5) You can remove or add new feature to a function anytime.
- 6) We can track a large Python program easily when it is divided into multiple functions.
- 7) Reusability is the main achievement of Python functions.
- 8) However, Function calling is always overhead in a Python program.

1) Function Definition

Creating a Function

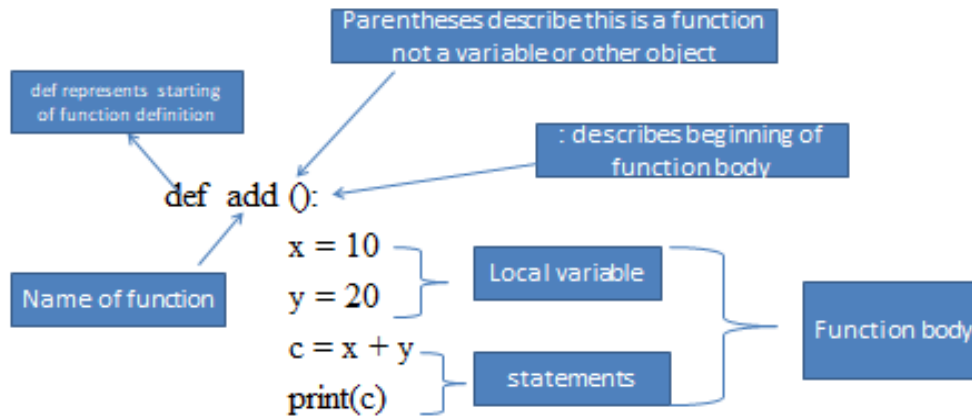
Python provides the **def** keyword to define the function. The syntax of the define function is given below.

Syntax:

```
def my_function(parameters):  
    function_block  
return expression
```

Let's understand the syntax of functions definition.

- 1) The **def** keyword, along with the function name is used to define the function.
- 2) The identifier rule must follow the function name.
- 3) A function accepts the parameter (argument), and they can be optional.
- 4) The function block is started with the colon (:), and block statements must be at the same indentation.
- 5) The **return** statement is used to return the value. A function can have only one **return**.



2) Function Calling

- 1) In Python, after the function is created, we can call it from another function.
- 2) A function runs only when we call it, function can not run on its own.
- 3) A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

4) Syntax:

```
function_name()  
function_name(arg1, arg2, ....)
```

Ex:-

```
add()  
add(10)  
add(20.50)
```

How function work:

1) def add():

```
x = 10
y = 20
c = x + y
print(c)
```

2) add():

```
def add(y):
```

```
    x = 10
    c = x + y
    print(c)
```

```
add(20)
```

parameter 'y' do not know which type of value they are about to receive till the value is passed at the time of calling the fun. Determine runtime not compile time.

Consider the following example of a simple example that prints the message "Hello World".

```
def hello_world():
    print("hello world")
hello_world()
```

#function definition

function calling

The return statement

- 1) The return statement is used at the end of the function and returns the result of the function.
- 2) It terminates the function execution and transfers the result where the function is called. The return statement cannot be used outside of the function.

Syntax

```
return [expression_list]
```

- 3) It can contain the expression which gets evaluated and value is returned to the caller function.
- 4) If the return statement has no expression or does not exist itself in the function then it returns the **None** object.

Consider the following example:

Ex. 1 Creating function with return statement

```
def sum():                # Defining function
    a = 10
    b = 20
    c = a+b
    return c
print("The sum is:",sum()) # calling sum() function in print statement
```

In the above code, we have defined the function named **sum**, and it has a statement.

c = a+b, which computes the given values, and the result is returned by the return statement to the caller function.

Ex. 2 Creating function without return statement

```
def sum():                # Defining function
    a = 10
    b = 20
    c = a+b
    print(sum())          # calling sum() function in print statement
```

In the above code, we have defined the same function without the return statement as we can see that the **sum()** function returned the **None** object to the caller function.

Arguments in function:

- 1) The arguments are types of information which can be passed into the function. The arguments are specified in the parentheses.
- 2) **An argument is the value that are sent to the function when it is called.**
- 3) We can pass any number of arguments, but they must be separate them with a comma.

Consider the following example, which contains a function that accepts a string as the argument.

Ex. 1

```
def func (name):           #defining the function
    print("Hi ",name)
func("Pratyush")          #calling the function
```

Ex. 2 #Python function to calculate the sum of two variables

```
def sum (a,b):              #defining the function
    return a+b;

a = int(input("Enter a: "))  #taking values from the user
b = int(input("Enter b: "))
print("Sum = ",sum(a,b))     #printing the sum of a and b
```

Call by reference in Python

- 1) In Python, call by reference means passing the actual value as an argument in the function.
- 2) All the functions are called by reference, i.e., all the changes made to the reference inside the function revert back to the original value referred by the reference.

Ex 1 Passing Immutable Object (List)

```
def change_list(list1):     #defining the function
    list1.append(20)
    list1.append(30)
    print("list inside function = ",list1)

list1 = [10,30,40,50]       #defining the list
change_list(list1)          #calling the function
```

```
print("list outside function = ",list1)
```

Output:

```
list inside function = [10, 30, 40, 50, 20, 30]
list outside function = [10, 30, 40, 50, 20, 30]
```

Ex 2 Passing Mutable Object (String)

```
def change_string (str):                                #defining the function
    str = str + " Hows you "
    print("printing the string inside function :",str)
    string1 = "Hi I am there"

change_string(string1)                                   #calling the function
print("printing the string outside function :",string1)
```

Output:

```
printing the string inside function : Hi I am there Hows you
printing the string outside function : Hi I am there
```

Types of arguments

There may be several types of arguments which can be passed at the time of function call.

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

1) Required Arguments:

1) Till now, we have learned about function calling in Python. However, we can provide the arguments at the time of the function call.

2) As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition.

3) If either of the arguments is not provided in the function call, or the position of the arguments is changed, the Python interpreter will show the error.

Consider the following example.

Example 1

```
def func(name):  
    message = "Hi "+name  
    return message  
name = input("Enter the name:")  
print(func(name))
```

Output:

```
Enter the name: Neer  
Hi Neer
```

Example 2

#the function simple_interest accepts three arguments and returns the simple interest accordingly

```
def simple_interest(p,t,r):  
    return (p*t*r)/100  
p = float(input("Enter the principle amount: "))  
r = float(input("Enter the rate of interest: "))  
t = float(input("Enter the time in years: "))  
print("Simple Interest: ",simple_interest(p,r,t))
```

Output:

```
Enter the principle amount: 5000  
Enter the rate of interest: 5  
Enter the time in years: 3  
Simple Interest: 750.0
```

Example 3

```
#the function calculate returns the sum of two arguments a and b
def calculate(a,b):
    return a+b
calculate(10)    # this causes an error as we are missing a required arguments b.
```

Output:

TypeError: calculate() missing 1 required positional argument: 'b'

2) Default Arguments:

- 1) Python allows us to initialize the arguments at the function definition.
- 2) If the value of any of the arguments is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.
 - Default arguments are values that are provided while defining functions.
 - The assignment operator = is used to assign a default value to the argument.
 - Default arguments become optional during the function calls.
 - If we provide a value to the default arguments during function calls, it overrides the default value.
 - The function can have any number of default arguments.
 - Default arguments should follow non-default arguments.

Example 1

```
def print(name,age=6):
    print("My name is",name,"and age is",age)
print(name = "Neer")
```

Output:

My name is Neer and age is 6

Example 2

```
def print(name,age=7):  
    print("My name is",name,"and age is",age)  
print(name = "Pratyush")      #the variable age is not passed into the function however the  
                              #default value of age is considered in the function  
print(age = 10,name="David")  #the value of age is overwritten here,  
                              10 will be printed as age
```

Output:

My name is john and age is 22
My name is David and age is 10

3) Variable-length Arguments (*args):

- 1) In large projects, sometimes we may not know the number of arguments to be passed in advance.
- 2) In such cases, Python provides us the flexibility to offer the comma-separated values which are internally treated as tuples at the function call. By using the variable-length arguments, we can pass any number of arguments.
- 3) However, at the function definition, we define the variable-length argument using the ***args** (star) as ***<variable - name >**.

Consider the following example.

Example

```
def print(*names):  
    print("type of passed argument is ",type(names))  
    print("printing the passed arguments...")  
    for name in names:  
        print(name)  
print("Pratyush","Neer","Reyansh","John")
```


Output:

```
type of passed argument is <class 'tuple'>
printing the passed arguments...
Pratyush
Neer
Reyansh
John
```

In the above code, we passed ***names** as variable-length argument. We called the function and passed values which are treated as tuple internally. The tuple is an iterable sequence the same as the list. To print the given values, we iterated ***arg** **names** using for loop.

4) Keyword arguments(**kwargs):

- 1) Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.
- 2) The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Functions can also be called using keyword arguments of the form `kwarg=value`.

During a function call, values passed through arguments don't need to be in the order of parameters in the function definition. This can be achieved by keyword arguments. But all the keyword arguments should match the parameters in the function definition.

Consider the following example.

Example 1

#function func is called with the name and message as the keyword arguments

```
def func(name,message):
```

```
    print("printing the message with",name,"and ",message)
```

```
    func(name = "Neer",message="hello") #name and message is copied with
```

the values John and hello respectively

Output:

printing the message with Neer and hello

Example 2 providing the values in different order at the calling

```
def simple_interest(p,t,r):  
    return (p*t*r)/100  
print("Simple Interest: ",simple_interest(t=10,r=10,p=1900))
```

Output:

Simple Interest: 1900.0

If we provide the different name of arguments at the time of function call, an error will be thrown.

Consider the following example.

The Python allows us to provide the mix of the required arguments and keyword arguments at the time of function call. However, the required argument must not be given after the keyword argument, i.e., once the keyword argument is encountered in the function call, the following arguments must also be the keyword arguments.

Consider the following example.

Example 4

```
def func(name1,message,name2):  
    print("printing the message with",name1,",",message,",and",name2)  
func("Neer",message="hello",name2="Pratyush")  
  
#the first arg is not the keyword argument
```

Output:

printing the message with Neer , hello ,and Pratyush

Parameter Passing:

- 1) Parameters allow functions to require additional pieces of information in order to be called
- 2) Parameters are specified within the parenthesis of function definition.
- 3) parameters look a lot like variable declarations.
- 4) parameters are local variables to the function. Their names are scoped inside of the function body's block.
- 5) when a function declares parameters, it is declaring you must give extra pieces of information In order to call.
- 6) Syntax:

```
def function_name(parameters):  
    statement 1  
    statement n
```

- 7) Ex.

```
def sum(int : x, int : y )  
  
    if x > y:  
        return x  
    else:  
        return y
```

Scope of variables

- 1) The scopes of the variables depend upon the location where the variable is being declared.
- 2) The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables

2. Local variables

3) The variable defined outside any function is known to have a global scope, whereas the variable defined inside a function is known to have a local scope.

Consider the following example.

Ex. 1 Local Variable

```
def print_message():  
    message = "hello "           # the variable message is local to the function itself  
    print(message)  
print_message()  
print(message)                  # this will cause an error since a local variable cannot be accessible here
```

Ex. 2 Global Variable

```
def calculate(*args):  
    sum=0  
    for arg in args:  
        sum = sum +arg  
    print("The sum is",sum)  
sum=0  
calculate(10,20,30)             #60 will be printed as the sum  
print("Value of sum outside the function:",sum)    # 0 will be printed Output:
```

Output:

The sum is 60

Value of sum outside the function: 0

Python Modules

1) A python module can be defined as a python program file which contains a python code including python functions, class, or variables.

2) In other words, we can say that our python code file saved with the extension (.py) is treated as the module. We may have a runnable code inside the python module.

3) Modules in Python provides us the flexibility to organize the code in a logical way.

4) To use the functionality of one module into another, we must have to import the specific module.

Example

In this example, we will create a module named as file.py which contains a function func that contains a code to print some message on the console

Let's create the module named as **file.py**.

```
#displayMsg prints a message to the name being passed.
```

```
def displayMsg(name)
```

```
    print("Hi "+name);
```

Here, we need to include this module into our main module to call the method displayMsg() defined in the module named file.

Loading the module in our python code

We need to load the module in our python code to use its functionality. Python provides two types of statements as defined below.

1. The import statement
2. The from-import statement

1) The import statement

1) The import statement is used to import all the functionality of one module into another.

2) we can use the functionality of any python source file by importing that file as the module into another python source file.

3) We can import multiple modules with a single import statement, but a module is loaded once regardless of the number of times, it has been imported into our file.

The syntax to use the import statement is given below.

```
import module1,module2,..... module n
```

4) Hence, if we need to call the function displayMsg() defined in the file file.py, we have to import that file as a module into our module as shown in the example below.

Example:

```
import file;
name = input("Enter the name: ")
file.displayMsg(name)
```

Output:

```
Enter the name: Pratyush
Hi Pratyush
```

2) The from-import statement

1) Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module.

2) This can be done by using from? import statement. The syntax to use the from-import statement is given below.

from < module-name> **import** <name 1>, <name 2>...,<name n>

Consider the following module named as calculation which contains three functions as summation, multiplication, and divide.

calculation.py:

#place the code in the calculation.py

```
def sum(a,b):
    return a+b
```

```
def mul(a,b):
    return a*b;
```

```
def div(a,b):
    return a/b;
```

Main.py:

```
from calculation import sum
```

```
#it will import only the sum() from calculation.py
```

```
a = int(input("Enter the first number: "))
```

```
b = int(input("Enter the second number: "))
```

```
print("Sum = ",sum(a,b))
```

```
#we do not need to specify module name while accessing sum()
```

Output:

```
Enter the first number: 10
Enter the second number: 20
Sum = 30
```

The from...import statement is always better to use if we know the attributes to be imported from the module in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using *.

Consider the following syntax.

```
from <module> import *
```

Renaming a module

Python provides us the flexibility to import some module with a specific name so that we can use this name to use that module in our python source file.

The syntax to rename a module is given below.

```
import <module-name> as <specific-name>
```

Example

```
#the module calculation of previous example is imported in this example as cal.
```

```
import calculation as cal;
```

```
a = int(input("Enter a: "));
```

```
b = int(input("Enter b: "));
```

```
print("Sum = ",cal.summation(a,b))
```

Using dir() function

The dir() function returns a sorted list of names defined in the passed module. This list contains all the sub-modules, variables and functions defined in this module.

Consider the following example.

Example

```
import json
List = dir(json)
print(List)
```

The reload() function

- 1) As we have already stated that, a module is loaded once regardless of the number of times it is imported into the python source file.
- 2) However, if you want to reload the already imported module to re-execute the top-level code, python provides us the reload() function.
- 3) The syntax to use the reload() function is given below.

```
reload(<module-name>)
```

for example, to reload the module calculation defined in the previous example, we must use the following line of code.

1. reload(calculation)

Scope of variables

- 1) In Python, variables are associated with two types of scopes. All the variables defined in a module contain the global scope unless or until it is defined within a function.
- 2) All the variables defined inside a function contain a local scope that is limited to this function itself. We can not access a local variable globally.
- 3) If two variables are defined with the same name with the two different scopes, i.e., local and global, then the priority will always be given to the local variable.

Consider the following example.

Example

```
name = "john"
def print_name(name):
    print("Hi",name) #prints the name that is local to this function only.
name = input("Enter the name?")
print_name(name)
```

Python Packages

- 1) The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules.
- 2) The packages are used to categorize the application level code efficiently.

Let's create a package named Employees in your home directory. Consider the following steps.

1. Create a directory with name Employees on path **/home**.
2. Create a python source file with name ITEmployees.py on the path **/home/Employees**.

ITEmployees.py

```
def getITNames():
    List = ["Neer", "Pratyush", "John", "Abc"]
    return List;
```

3. Similarly, create one more python file with name BPOEmployees.py and create a function getBPONames().

4. Now, the directory **Employees** which we have created in the first step contains two python modules. To make this directory a package, we need to include one more file here, that is `__init__.py` which contains the import statements of the modules defined in this directory.

`__init__.py`

1. **from** ITEmployees **import** getITNames
2. **from** BPOEmployees **import** getBPONames

5. Now, the directory **Employees** has become the package containing two python modules. Here we must notice that we must have to create `__init__.py` inside a directory to convert this directory to a package.

6. To use the modules defined inside the package **Employees**, we must have to import this in our python source file. Let's create a simple python source file at our home directory (/home) which uses the modules defined in this package.

Test.py

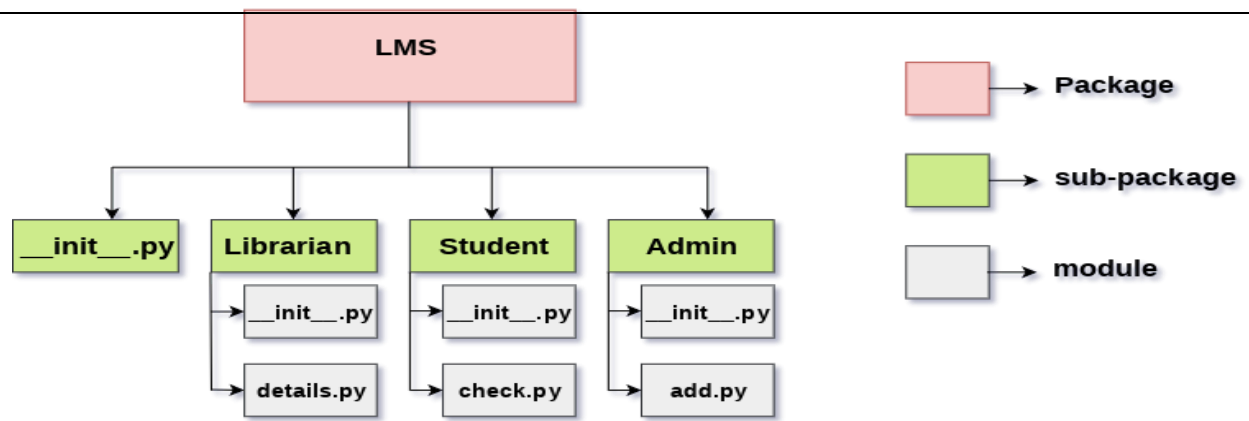
1. **import** Employees
2. **print**(Employees.getNames())

Output:

```
['John', 'David', 'Nick', 'Martin']
```

We can have sub-packages inside the packages. We can nest the packages up to any level depending upon the application requirements.

The following image shows the directory structure of an application Library management system which contains three sub-packages as Admin, Librarian, and Student. The sub-packages contain the python modules.



Python Standard Library

The Python Standard Library is a collection of exact syntax, token, and semantics of Python. It comes bundled with core Python distribution. We mentioned this when we began with an introduction.

It is written in C, and handles functionality like I/O and other core modules. All this functionality together makes Python the language it is.

More than 200 core modules sit at the heart of the standard library. This library ships with Python.

But in addition to this library, you can also access a growing collection of several thousand components from the Python Package Index (PyPI). We mentioned it in the previous blog.

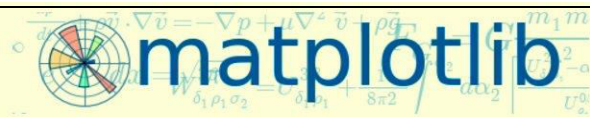
Important Python Libraries

Next, we will see twenty Python libraries list that will take you places in your journey with Python.

These are also the Python libraries for Data Science.

1. Matplotlib

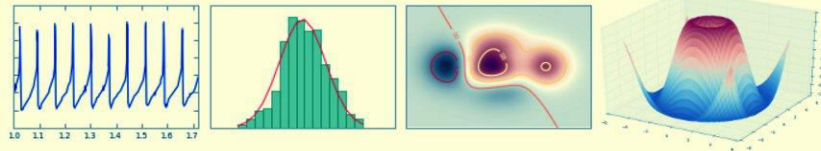
Matplotlib helps with data analyzing, and is a numerical plotting library. We talked about it in Python for Data Science.



[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) »

Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and [ipython](#) shell (ala MATLAB[®] or Mathematica[®]), web application servers, and six graphical user interface toolkits.



matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the [screenshots](#), [thumbnail gallery](#), and [examples](#) directory

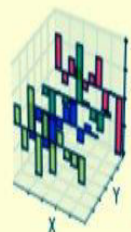
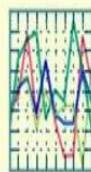
For simple plotting the [pyplot](#) interface provides a MATLAB-like interface, particularly when combined with [IPython](#). For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

2. Pandas

Like we've said before, Pandas is a must for data-science.

It provides fast, expressive, and flexible data structures to easily (and intuitively) work with structured (tabular, multidimensional, potentially heterogeneous) and time-series data.

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



[overview](#) // [get pandas](#) // [documentation](#) // [community](#) // [talks](#)

Python Data Analysis Library

VERSIONS

Release

0.17.0 - October 2015

[download](#) // [docs](#) // [pdf](#)

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the [Python](#) programming language.

Python Libraries Tutorial – Pandas

3. Requests

Requests is a Python Library that lets you send HTTP/1.1 requests, add headers, form data, multipart files, and parameters with simple Python dictionaries.



Requests

 Star **15,782**

Requests is an elegant and simple HTTP library for Python, built for human beings.

 Buy Requests Pro

Get Updates

Receive updates on new releases and upcoming projects.

Requests: HTTP for Humans

Release v2.8.1. ([Installation](#))

Requests is an [Apache2 Licensed](#) HTTP library, written in Python, for human beings.

Python's standard `urllib2` module provides most of the HTTP capabilities you need, but the API is thoroughly **broken**. It was built for a different time — and a different web. It requires an *enormous* amount of work (even method overrides) to perform the simplest of tasks.


Things shouldn't be this way. Not in Python.

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type": "User"... '
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

Python Libraries Tutorial- Requests

4. NumPy

It has advanced math functions and a rudimentary scientific computing package.



NumPy

[Scipy.org](#)

NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

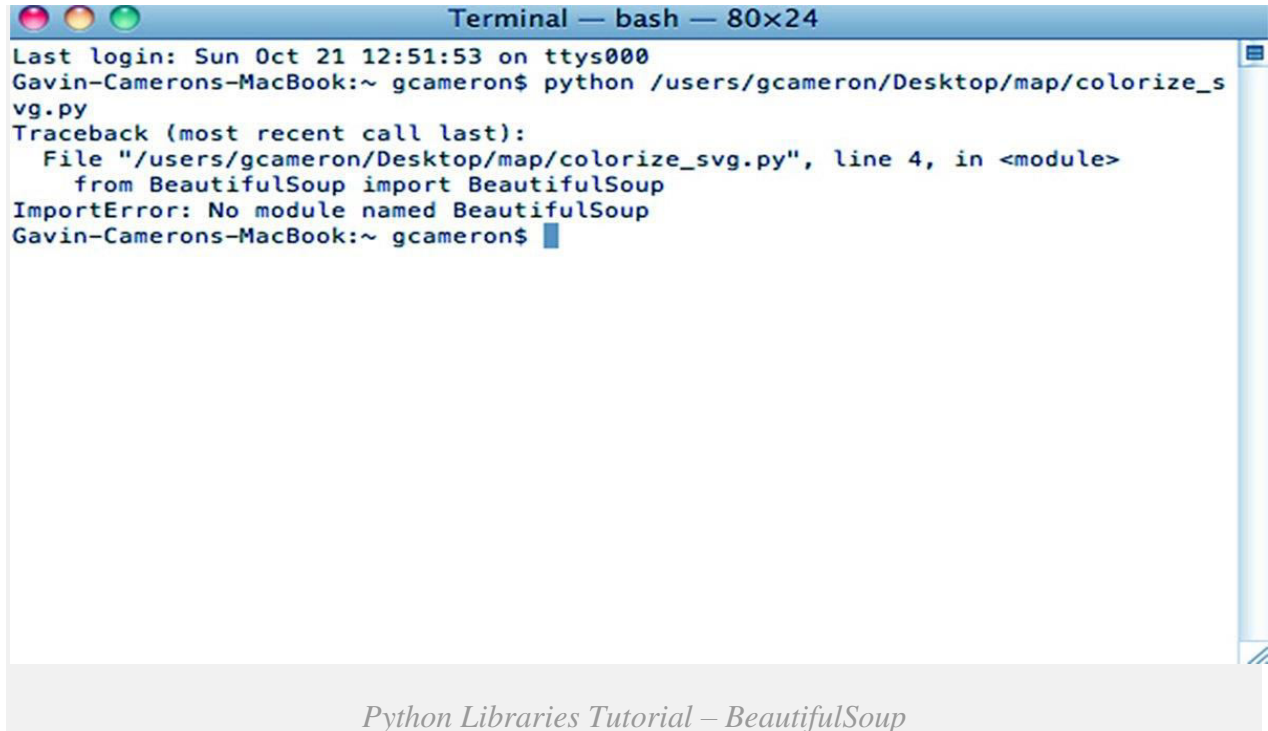
- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

NumPy is licensed under the [BSD license](#), enabling reuse with few restrictions.

5. BeautifulSoup

It may be a bit slow, BeautifulSoup has an excellent XML- and HTML- parsing library for beginners.

A screenshot of a macOS Terminal window titled "Terminal — bash — 80x24". The window shows the output of a Python command. The prompt is "Gavin-Camerons-MacBook:~ gcameron\$". The command entered is "python /users/gcameron/Desktop/map/colorize_svg.py". The output shows a traceback for an ImportError: "No module named BeautifulSoup". The traceback indicates the error occurred in the file "/users/gcameron/Desktop/map/colorize_svg.py" at line 4, within the module's execution. The prompt returns to "Gavin-Camerons-MacBook:~ gcameron\$".

```
Terminal — bash — 80x24
Last login: Sun Oct 21 12:51:53 on ttys000
Gavin-Camerons-MacBook:~ gcameron$ python /users/gcameron/Desktop/map/colorize_s
vg.py
Traceback (most recent call last):
  File "/users/gcameron/Desktop/map/colorize_svg.py", line 4, in <module>
    from BeautifulSoup import BeautifulSoup
ImportError: No module named BeautifulSoup
Gavin-Camerons-MacBook:~ gcameron$
```

Python Libraries Tutorial – BeautifulSoup

6. Pyglet

Pyglet is an excellent choice for an object-oriented programming interface in developing games.

In fact, it also finds use in developing other visually-rich applications for Mac OS X, Windows, and Linux.

In the 90s, when people were bored, they resorted to playing Minecraft on their computers. Pyglet is the engine behind Minecraft.



pyglet: a cross-platform windowing and multimedia library for Python.

[home](#) | [download](#) | [documentation](#) | [contribute](#)

pyglet provides an object-oriented programming interface for developing games and other visually-rich applications for Windows, Mac OS X and Linux.

Some of the features of pyglet are:

- **No external dependencies or installation requirements.** For most application and game requirements, *pyglet* needs nothing else besides Python, simplifying distribution and installation.
- **Take advantage of multiple windows and multi-monitor desktops.** *pyglet* allows you to use as many windows as you need, and is fully aware of multi-monitor setups for use with fullscreen games.
- **Load images, sound, music and video in almost any format.** *pyglet* can optionally use AVbin to play back audio formats such as MP3, OGG/Vorbis and WMA, and video formats such as DivX, MPEG-2, H.264, WMV and Xvid.
- **pyglet is provided under the BSD open-source license**, allowing you to use it for both commercial and other open-source projects with very little restriction.

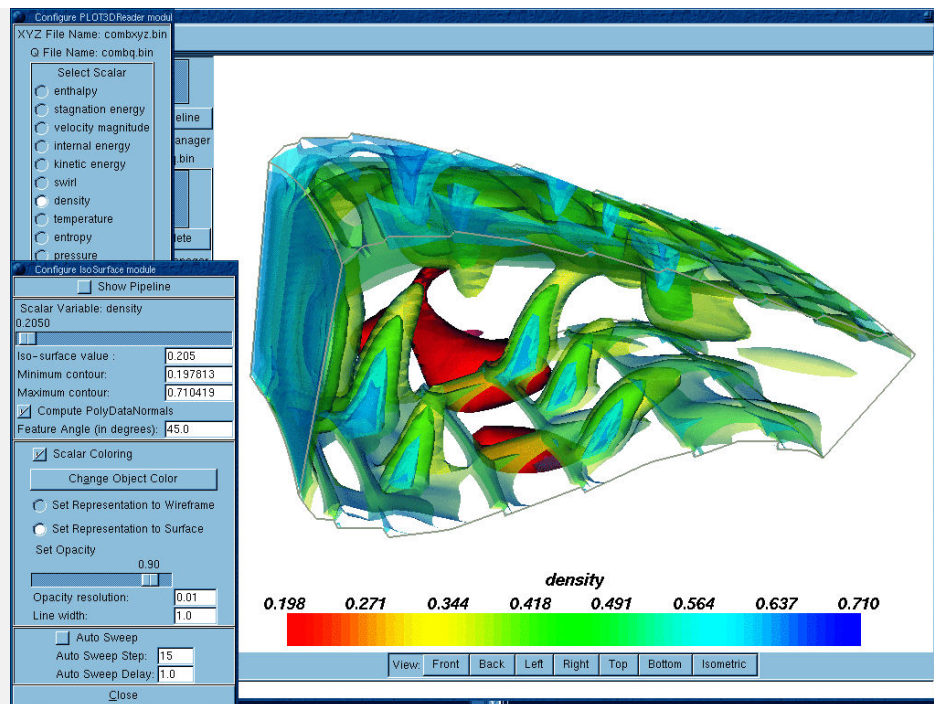
Please join us on the mailing list, or download the SDK today!

Python Libraries Tutorial – Pyglet

7. SciPy

Next up is SciPy, one of the libraries we have been talking so much about. It has a number of user-friendly and efficient numerical routines.

These include routines for optimization and numerical integration.

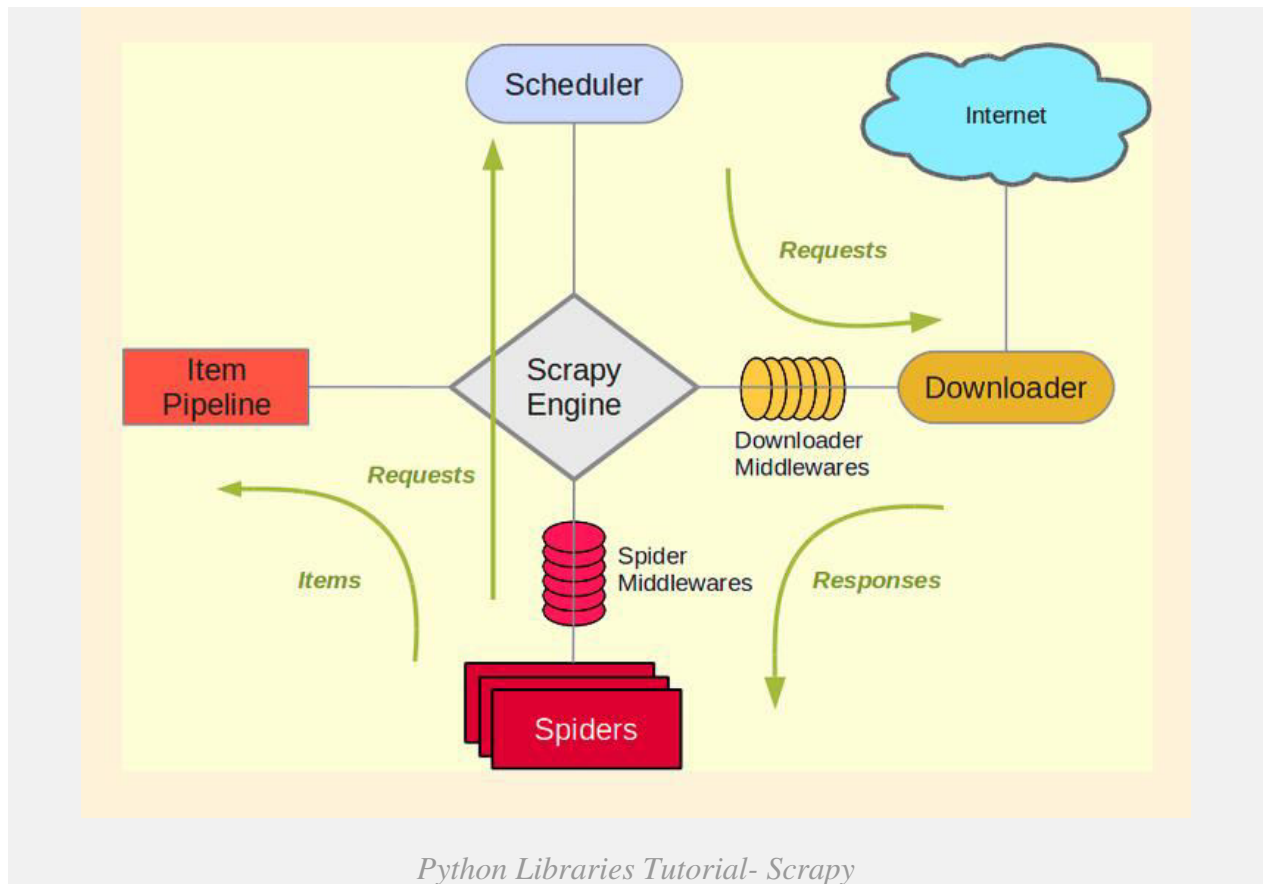


Python Libraries Tutorial- SciPy

8. Scrapy

If your motive is fast, high-level screen scraping and web crawling, go for Scrapy.

You can use it for purposes from data mining to monitoring and automated testing.



9. PyGame

PyGame provides an extremely easy interface to the Simple Directmedia Library (SDL) platform-independent graphic, audio, and input libraries.

