Unit
4

Functions

Functions are a great deal, even for other programming languages. Functions are important in Python at the point that we have **built-in functions** (functions pre-defined in Python).

4.1 Introduction to Functions

In Python, functions help to split programs into pieces(modules), thereby making them easier to manage and maintain.

Functions enable us to implement a very powerful algorithm design paradigm called "**Divide-and-Conquer**" that basically breaks down an idea into two or more sub-ideas, and makes them simple enough to implement.

User Defined Functions

Create a Function

A Python function has the following syntax:

```
def function_name(arg1, arg2,...,argN):
    # function code
```

To define a Python function, use def keyword. Here's the simplest possible function that prints 'Hello, World!' on the screen.

```
def hello():
print('Hello, World!')
```

Call a Function

The def statement only creates a function but does not call it. After the def has run, you can can call (run) the function by adding parentheses after the function's name.

```
def hello():
print('Hello, World!')
hello()
```

Output :

```
Hello, World!
```

To be more thorough, let's consider the below function that multiplies two numbers and returns the result.

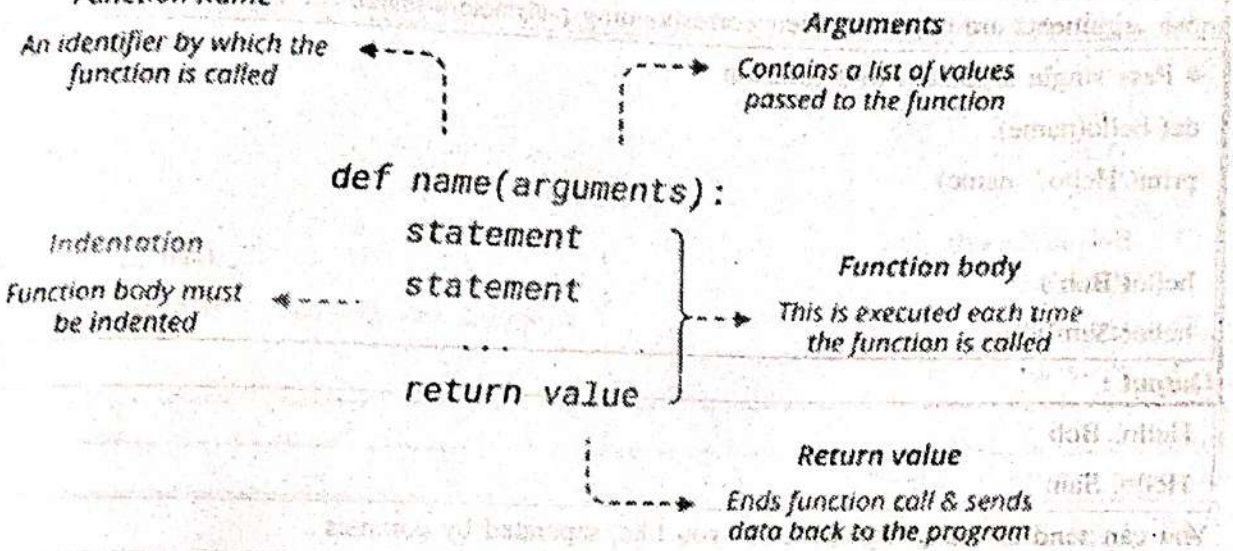


Figure : Function definition with annotations

We can see that a function has the following key-parts

- **def keyword :** The "def keyword" is used to write functions.
- **function name :** The function's name created by the **def statement**. This permits us to define functions once and call them in many parts of our code.
- **function parameters :** The parameters are used to pass data into the function's body.
- **Colon :** The colon(:) is a cue for the function's body. That is, the function body gets indented after the colon.
- **function code :** The function code also called the **function body** contains statements that get executed when the function gets called..

For example, the simple function defined below returns the square of the value you pass into it.

```
def square(n):
    return n**2
square(3)
```

Output

```
9
```

The syntax for this function definition includes the name, square, and a parenthesized list of formal parameters.

Arguments and Parameters

A parameter is a placeholder (variable) that is placed inside parentheses in a function definition while an argument is a value that is passed to the function when it is called.

Pass Arguments

You can send information to a function by passing values, known as arguments. Arguments are declared after the function name in parentheses. When you call a function with arguments, the values of those arguments are copied to their corresponding parameters inside the function.

```
# Pass single argument to a function
def hello(name):
print('Hello,', name)

hello('Bob')
hello('Sam')
```

Output :

```
Hello, Bob
Hello, Sam
```

You can send as many arguments as you like, separated by commas ,.

```
#. Pass two arguments
def func(name, job):
print(name, 'is a', job)

func('Bob', 'developer')
```

Output :

```
Bob is a developer
```

Define Parameters with Default Values

In Python, if a function is defined with parameters and the caller doesn't pass in arguments that match the number of parameters, then a TypeError will occur.

Example : Check the sample code below.

```
# define function with two parameters.
def display(x, y):
print('X: ', x)
```

```
print('Y: ', y)

if __name__ == '__main__':
    # function called and passed only one argument
display(4)
```

Output :

```
Type Erro r: display() missing 1 required positional argument: 'y'
```

It is possible that we define our function with parameters but pass in some default values into the body of the function when we don't provide them with arguments.

Example : Function Parameters with Default Values

```
def display(x, y=0):
print('X: ', x)
print('Y: ', y)

if __name__ == '__main__':
    # function called and passed only one argument
display(4)
display(4,5)
```

Output :

```
X:  4
Y:  0
X:  4
Y:  5
```

4.2 Scope of a Variable

(3)

Local Variable

Local variables are the ones that are defined and declared inside a function. We cannot call this variable outside the function.

```
# This function uses global variable s
def f():
    s = "Welcome to India"
    print(s)
f()
```

Output :

```
Welcome to India
```

Global Variable

Global variables are the ones that are defined and declared outside a function, and we need to use them inside a function.

```
# This function has a variable with
# name same as s.
def f():
    print(s)
# Global scope
s = "I love India"
f()
```

Output :

```
I love India
```

Global keyword is a keyword that allows a user to modify a variable outside of the current scope. It is used to create global variables from a non-global scope i.e. inside a function. Global keyword is used inside a function only when we want to do assignments or when we want to change a variable. Global is not needed for printing and accessing.

Rules of global keyword :

- If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.
- Variables that are only referenced inside a function are implicitly global.
- We Use global keyword to use a global variable inside a function.
- There is no need to use global keyword outside a function.

Example :

```
# Python program to modify a global
# value inside a function
x = 15
def change():
    # using a global keyword
    global x
    # increment value of x by 5
    x = x + 5
print("Value of x inside a function :", x)
change()
print("Value of x outside a function :", x)
```

Output :

```
Value of x inside a function : 20
Value of x outside a function : 20
```

4.1 Python Standard Library

A Python library is a reusable chunk of code that you may want to include in your programs/ projects. Essentially, then, a library is a collection of modules. A package is a library that can be installed using a package manager like rubygems or npm.

Python Standard Libraries are the collection of script modules that are accessible to a Python program to make the programming process simpler and to remove the need to rewrite commonly used commands again and again. They can be used by calling/importing them at the beginning of a python script.

It is important to become familiar with the Python Standard Library as so many problems can be solved quickly if you are familiar with the things that these libraries can do.

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.

Using libraries and frameworks will make our life painless while working on large, complex projects. So. it is essential to explore the various libraries/frameworks and decide on which to use before delving into any big project.

Some of the commonly used Python libraries/frameworks are :

Library/Framework	Description	Domain Commonly Used
Django, Flask	Open-source frameworks that allow us to develop web applications.	Web application
Tensorflow, Keras	Open-source libraries which allows us to create large scale AI based projects.	Artificial Intelligence
Numpy, Panda	Libraries that is mostly used to perform scientific computations.	Machine Learning, Data Science,
PyQT5, Tkinter, wxPython	Graphical User Interface (GUI) framework for Python.	Desktop Applications
Pygame, PyKyra	An open-source library and framework respectfully, that is highly used to build multimedia applications like games.	Game development
Matplotlib	It is a multi-platform plotting library built on NumPy arrays, that helps in the visualization of data.	Data Science, Machine Learning.
Scikit-learn	It is an open-sourced library designed on top of SciPy that incorporates various machine learning algorithms like classification, regression, clustering, etc.	Machine Learning

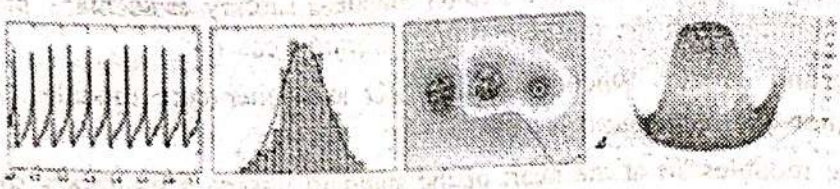
1. Matplotlib

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

matplotlib

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

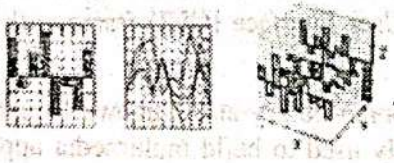
For simple plotting the pyplot interface provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

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



overview // get pandas // documentation // community // talks

Python Data Analysis Library

VERSIONS

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.

Release
0.17.0 - October 2015
download // docs // pdf

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.

It also lets you access the response data in the same way.

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.

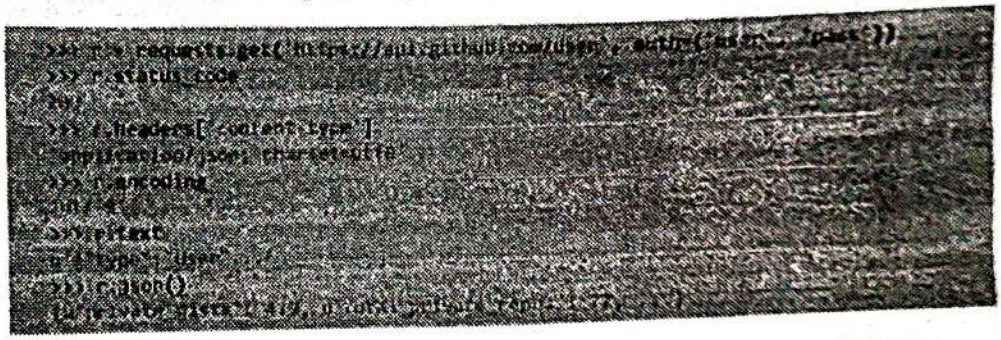
Requests: HTTP for Humans

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.



4. NumPy

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

SciPy.org

NumPy

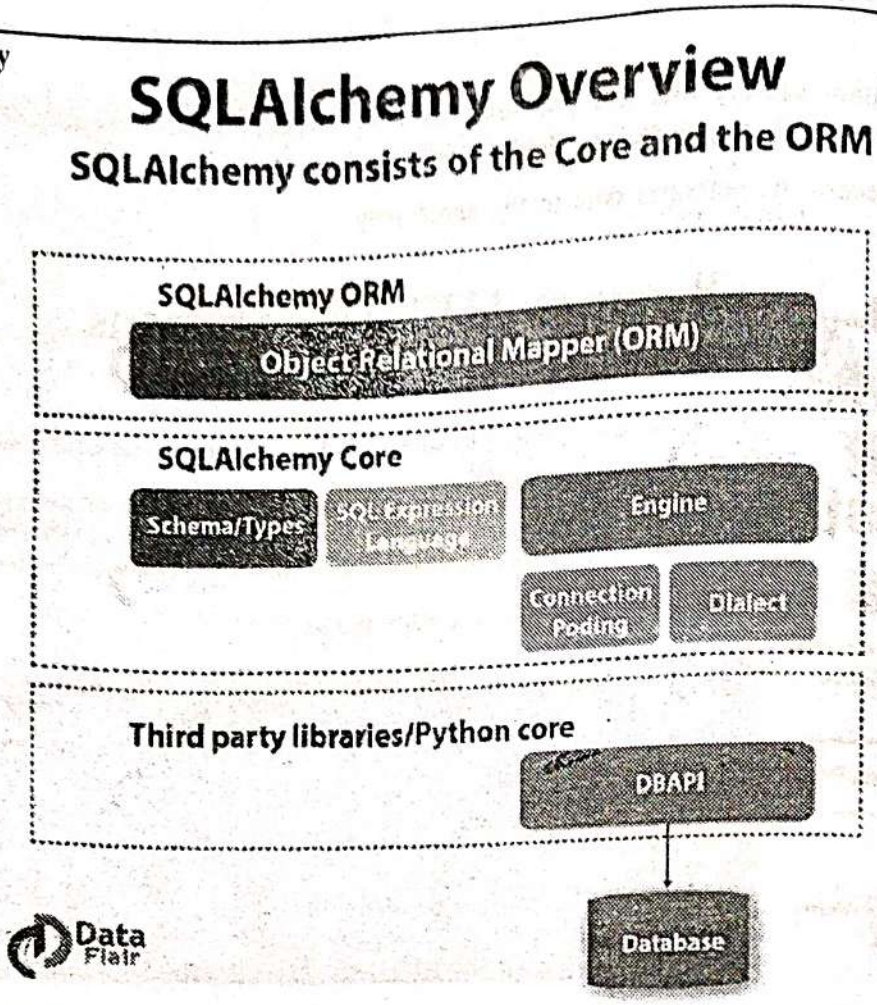
NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- 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. SQLAlchemy



SQLAlchemy is a library with well-known enterprise-level patterns.

It was designed for efficient and high-performing database-access.

6. BeautifulSoup

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

```
Terminal — bash — 80×24
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$
```

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 | Community

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:

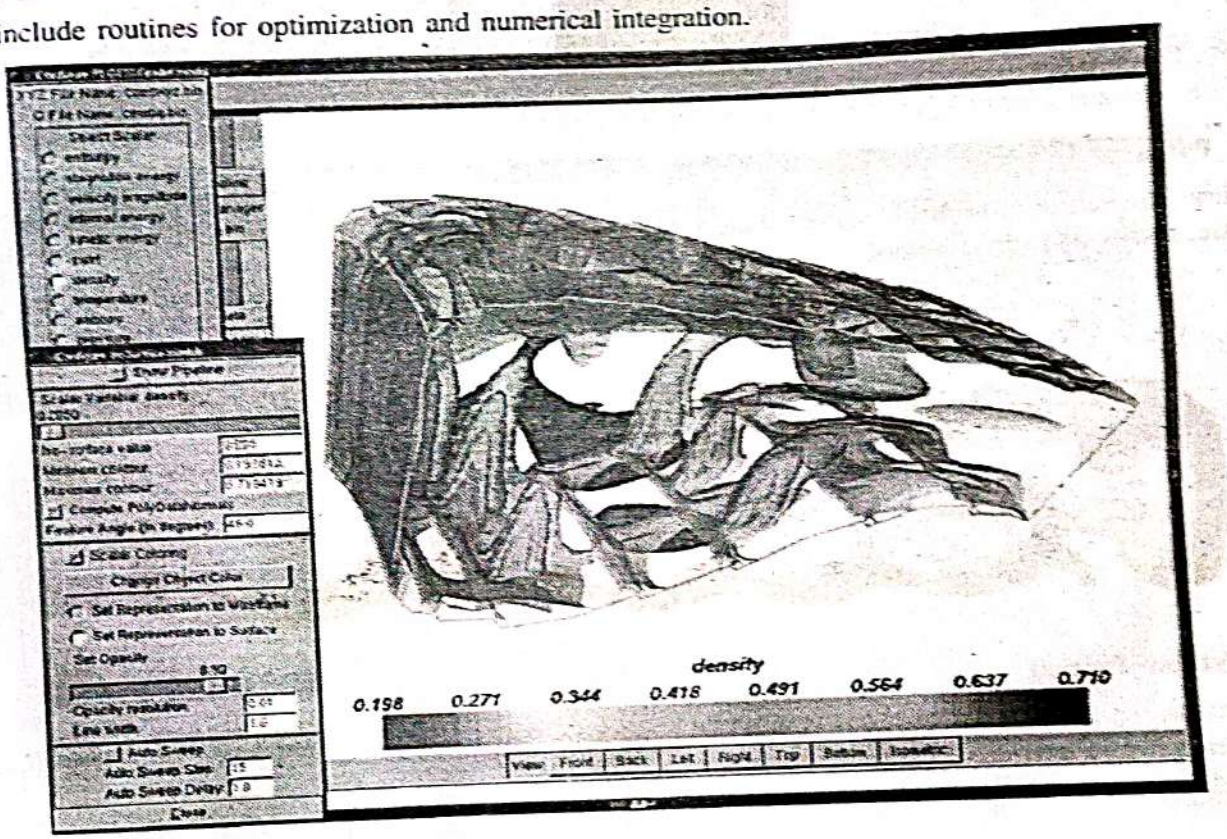
- No external dependencies or installation requirements. For most application and game requirements, pyglet needs nothing else besides Python, simplifying distribution and installation.
- Take advantage of multiple windows and multi-monitor desktops. pyglet allows you to use as many windows as you need, 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!

8. SciPy

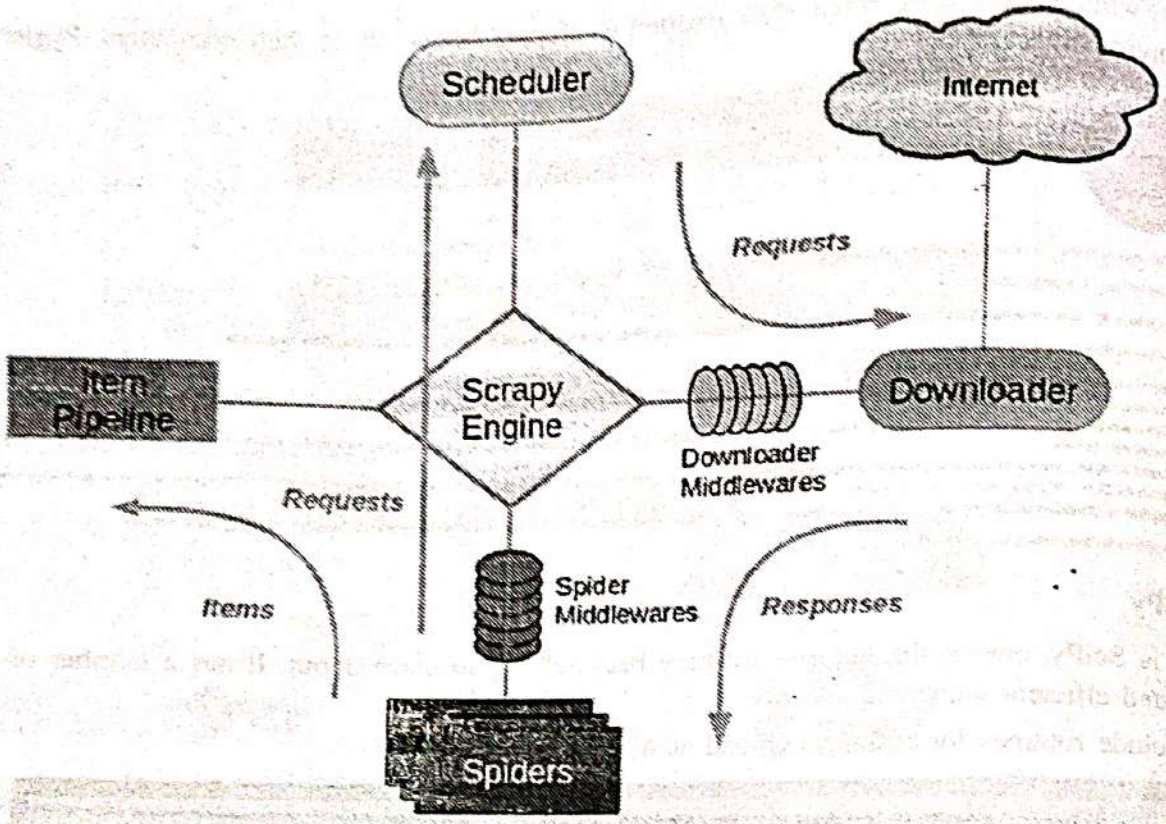
Next up is SciPy, one of the libraries we have been talking so much about. It has a number of user-friendly and efficient numerical routines.

These include routines for optimization and numerical integration.



9. 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.

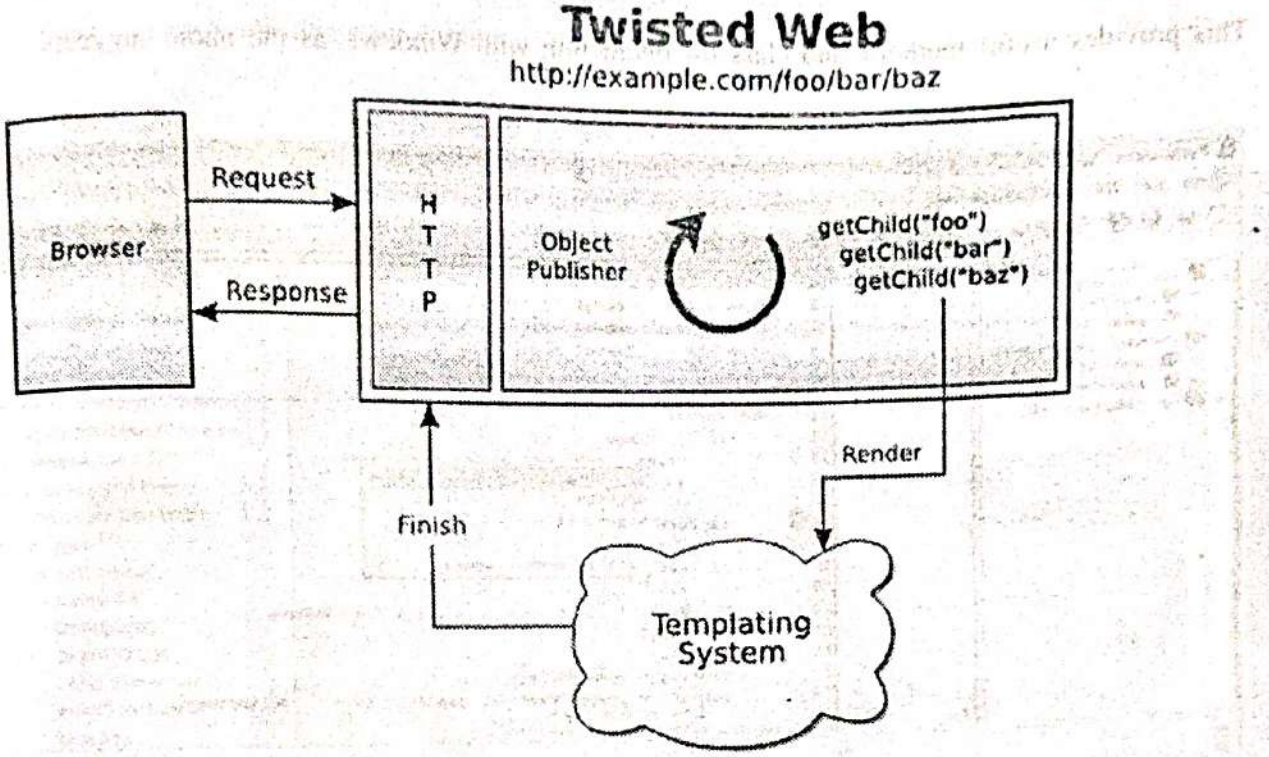


10. PyGame

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

11. Python Twisted

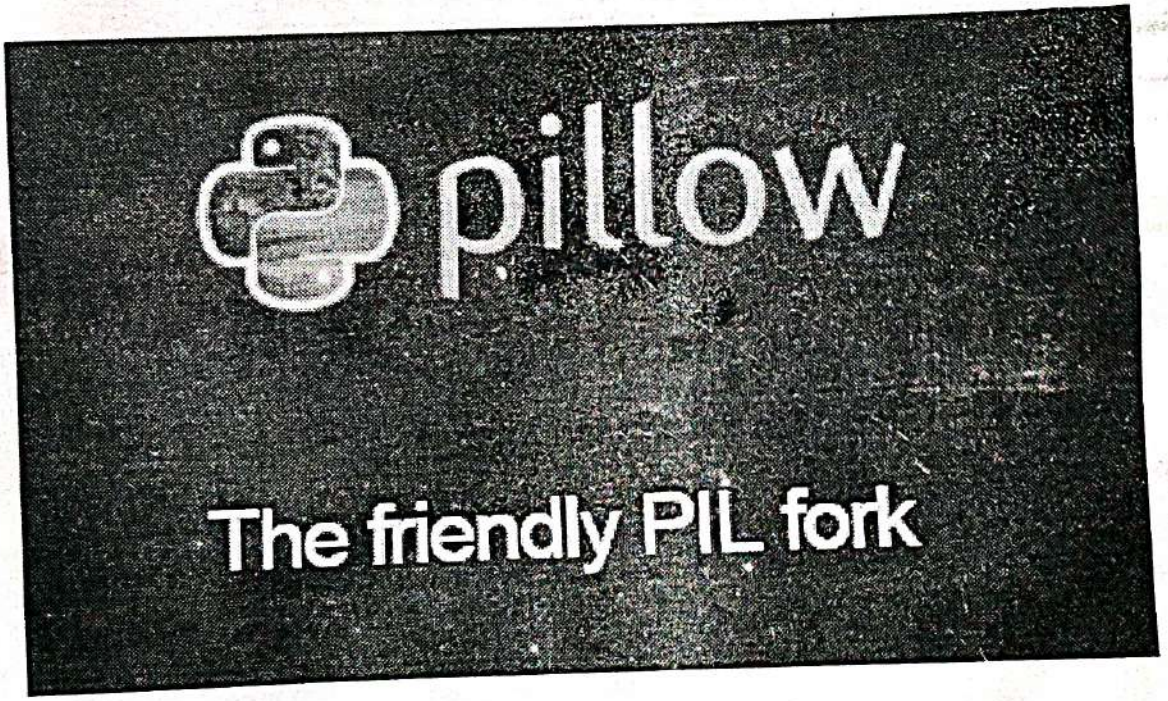
An event-driven networking engine, Twisted is written in Python, and licensed under the open-source MIT license.



12. Pillow

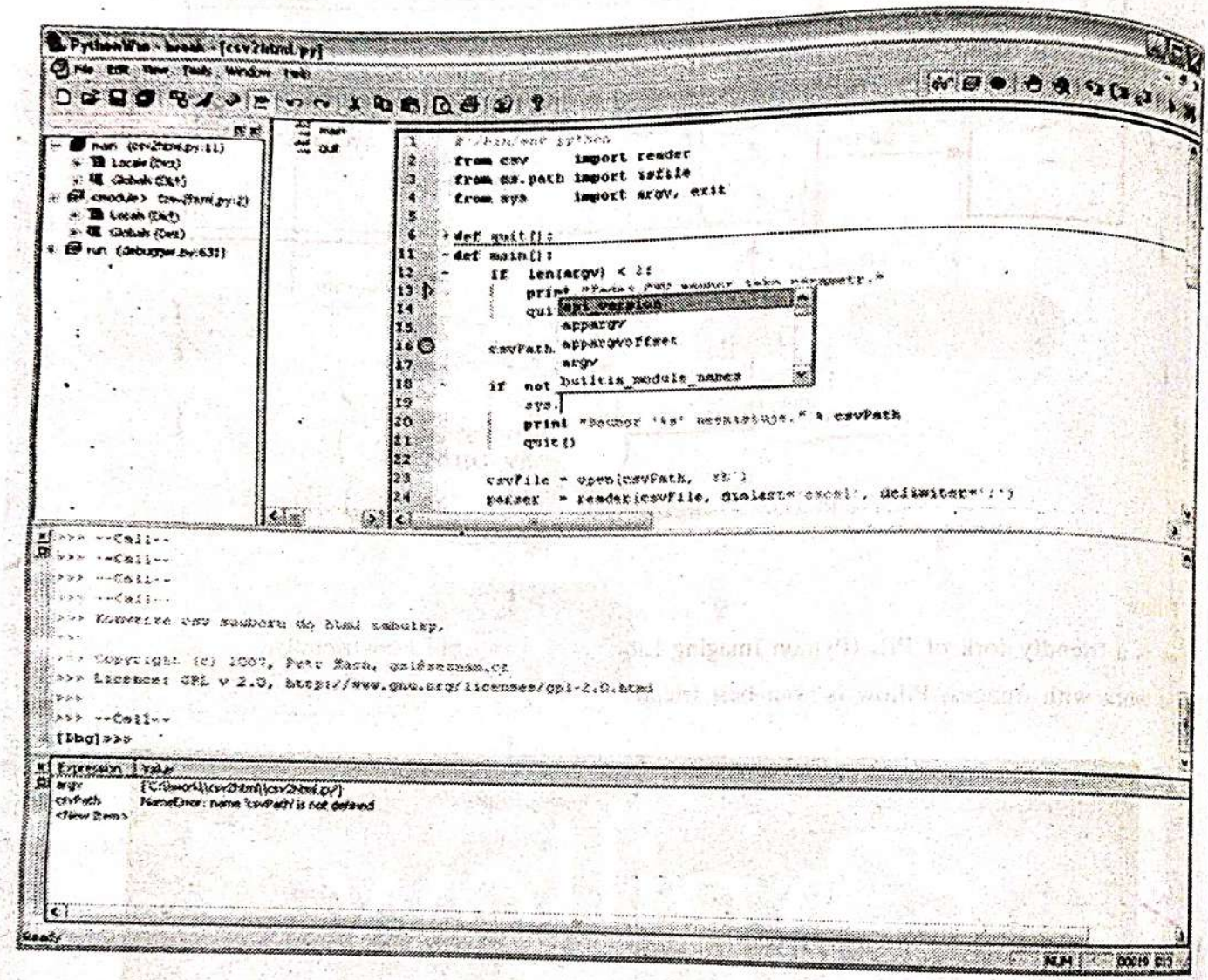
Pillow is a friendly fork of PIL (Python Imaging Library), but is more user-friendly.

If you work with images, Pillow is your best friend.



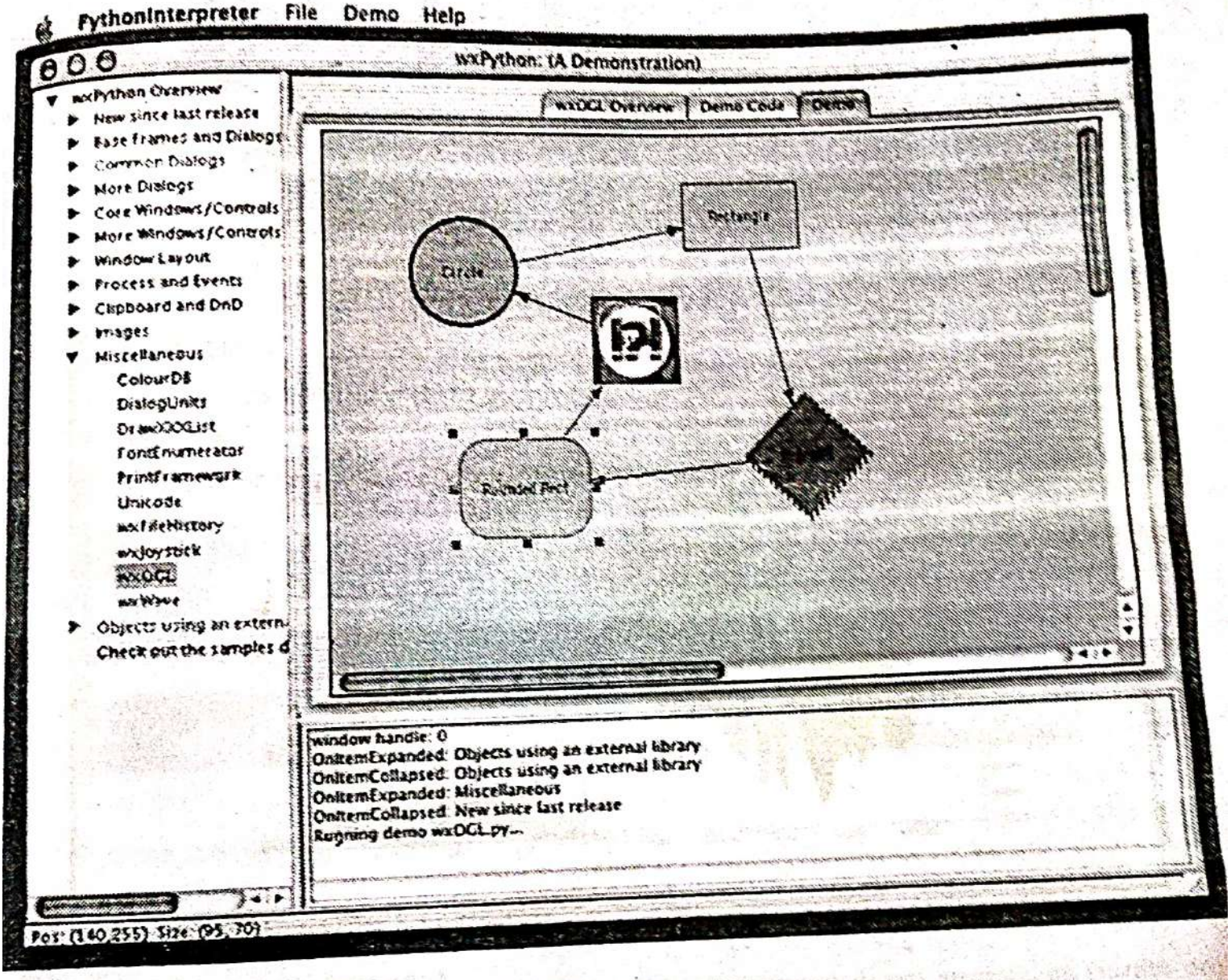
13. pywin32

This provides useful methods and class for interaction with Windows, as the name suggests.



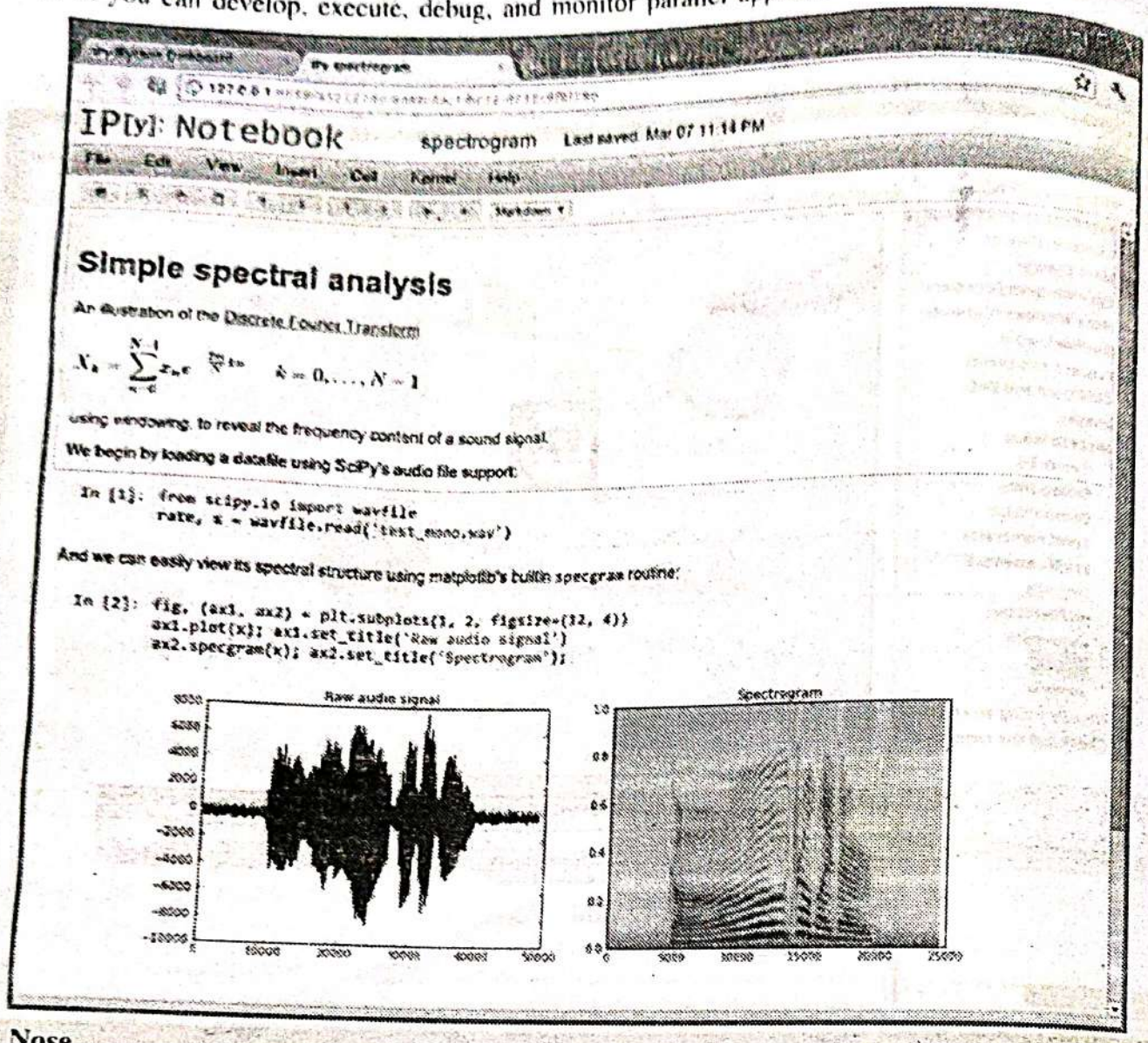
14. wxPython

It is a wrapper around wxWidgets for Python.



15. iPython

iPython Python Library has an architecture that facilitates parallel and distributed computing. With it, you can develop, execute, debug, and monitor parallel applications.



16. Nose

Nose delivers an alternate test discovery and running process for unittest. This intends to mimic py.test's behavior as much as it can.

nose

is nicer testing for python

nose extends unittest to make testing easier.

17. Flask

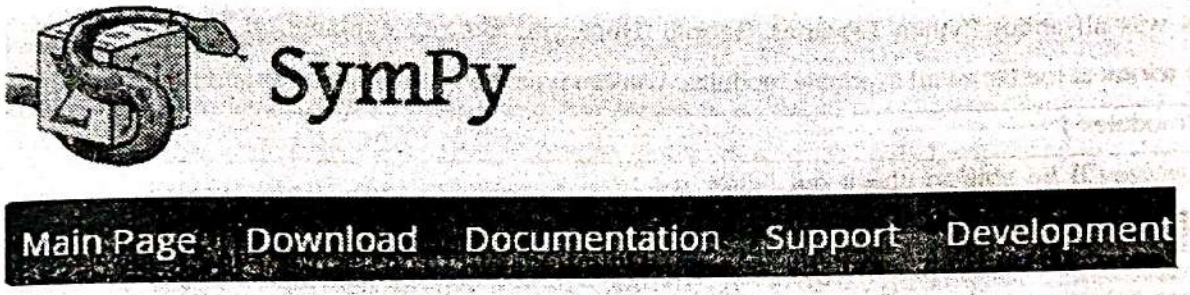
A web framework, Flask is built with a small core and many extensions.

18. SymPy

It is an open-source library for symbolic math.

With very simple and comprehensible code that is easily extensible, SymPy is a full-fledged Computer Algebra System (CAS).

It is written in Python, and hence does not need external libraries.




```
Python console for SymPy 0.7.6 (Python 2.7.5)

These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
```

19. Fabric

Along with being a library, Fabric is a command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.

With it, you can execute local or remote shell commands, upload/download files, and even prompt running user for input, or abort execution.

Fabric

Pythonic remote execution

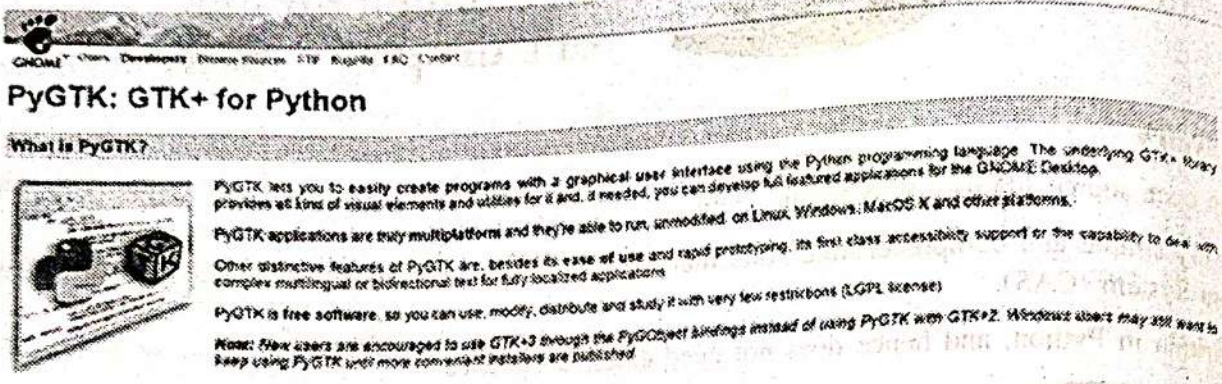
Welcome to Fabric!

Fabric is a Python (2.5-2.7) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.

It provides a basic suite of operations for executing local or remote shell commands (normally or via sudo) and uploading/downloading files, as well as auxiliary functionality such as prompting the running user for input, or aborting execution.

20. PyGTK

PyGTK lets you easily create programs with a GUI (Graphical User Interface) with Python.



GNOME Users Developers Documentation Support Bugzilla FAQ Contact

PyGTK: GTK+ for Python

What is PyGTK?

PyGTK lets you to easily create programs with a graphical user interface using the Python programming language. The underlying GTK+ library provides all kind of visual elements and utilities for it and, if needed, you can develop full featured applications for the GNOME Desktop.

PyGTK applications are truly multiplatforms and they're able to run, unmodified, on Linux, Windows, MacOS X and other platforms.

Other distinctive features of PyGTK are, besides its ease of use and rapid prototyping, its first class accessibility support or the capability to deal with complex multilingual or bidirectional text for fully localized applications.

PyGTK is free software, so you can use, modify, distribute and study it with very few restrictions (LGPL license).

Note: New users are encouraged to use GTK+3 through the PyGObject bindings instead of using PyGTK with GTK+2. Windows users may still want to keep using PyGTK until more convenient installers are published.

So, this was all about Python Libraries Tutorial. Hope you like our explanation.

To have a look at the list of all available modules, you can type the following command in the python shell :

```
help('modules')
```

And now you'll be able to check out below list :

```
Python 3.8 (64-bit)
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> help('modules')

Please wait a moment while I gather a list of all available modules...

__future__          _tkinter            getpass             sched
_abc                _tracemalloc        gettext             secrets
_ast                _warnings           glob                select
_asyncio            _weakref            gzip                selectors
_bisect             _weakrefset         hashlib             setuptools
_blake2             _winapi             heapq               shelve
_bootlocale         _xxsubinterpreters  hmac                shlex
_bz2                abc                 html                shutil
_codecs             aifc                http                signal
_codecs_cn          antigravity         idlelib             site
_codecs_hk          argparse            imaplib             smtpd
_codecs_iso2022     array               imghdr              smtplib
_codecs_jp          ast                 imp                 sndhdr
_codecs_kr          asynchat            importlib           socket
_codecs_tw          asyncio             inspect             socketserver
_collections        asyncore            io                  sqlite3
_collections_abc    atexit              ipaddress           sre_compile
_compat_pickle      audioop             itertools           sre_constants
_compression        base64              json                sre_parse
_contextvars        bdb                 keyword             ssl
_csv                binascii            lib2to3             stat
_ctypes             binhex              linecache           statistics
_ctypes_test        bisect              locale              string
_datetime           builtins            logging             stringprep
```

Note : This is not the complete list of the modules.

Built-in Functions

Input or output - input(), print()

input ()

Python provides us with a function that allows us to ask a user to enter some data and returns a reference to the data in the form of a string. The function is called input.

Python's input function takes a single parameter that is a string. For example, you might call input as follows:

```
aName = input('Please enter your name: ')
```

Now whatever the user types after the prompt will be stored in the aName variable. Using the input function, we can easily write instructions that will prompt the user to enter data and then incorporate that data into further processing. For example, in the following two statements, the first asks the user for their name and the second prints the result of some simple processing based on the string that is provided.

Example :

```
aName = input("Please enter your name ")
print("Your name in all capitals is",aName.upper(),"and has length", len(aName))
```

Output :

```
Please enter your name Jitendra
Your name in all capitals is JITENDRA and has length 8
```

It is important to note that the value returned from the input function will be a string representing the exact characters that were entered after the prompt. If you want this string interpreted as another type, you must provide the type conversion explicitly. In the statements below, the string that is entered by the user is converted to a float so that it can be used in further arithmetic processing.

```
sradius = input("Please enter the radius of the circle ")
radius = float(sradius)
diameter = 2 * radius
print(radius)
print(diameter)
```

Output :

```
Please enter the radius of the circle 2
2.0
4.0
```

print()

The print function provides a very simple way to output values from a Python program. print takes zero or more parameters and displays them using a single blank as the default separator. It is possible to change the separator character by setting the sep argument. In addition, each print ends with a newline character by default. This behavior can be changed by setting the end argument. These variations are shown in the following session :

```
print("Hello")
print("Hello","World")
print("Hello","World", sep="***")
print("Hello","World", end="***")
```

Output :

```
Hello
Hello World
Hello***World
Hello World***
```

Python provides us with an alternative called **formatted strings**. A formatted string is a template in which words or spaces that will remain constant are combined with placeholders for variables that will be inserted into the string. For example, the statement

```
print(aName, "is", age, "years old.")
```

contains the words is and years old, but the name and the age will change depending on the variable values at the time of execution. Using a formatted string, we write the previous statement as

```
print("%s is %d years old." % (aName, age))
```

This simple example illustrates a new string expression. The % operator is a string operator called the **format operator**. The left side of the expression holds the template or format string, and the right side holds a collection of values that will be substituted into the format string. Note that the number of values in the collection on the right side corresponds with the number of % characters in the format string. Values are taken—in order, left to right—from the collection and inserted into the format string.

The format string may contain one or more conversion specifications. In the example above, the %s specifies a string, while the %d specifies an integer. Following table summarizes all of the various type specifications.

Character	Output Format
d, i	Integer
u	Unsigned integer
f	Floating point as m.ddddd
e	Floating point as m.dddddde+/-xx
E	Floating point as m.dddddE+/-xx
g	Use %e for exponents less than "4"4 or greater than +5+5, otherwise use %f
c	Single character
s	String, or any Python data object that can be converted to a string by using the str function.
%	Insert a literal % character

Format modifiers may be used to left-justify or right-justifiy the value with a specified field width. Modifiers can also be used to specify the field width along with a number of digits after the decimal point. Following table explains these format modifiers

Modifier	Example	Description
number	%20d	Put the value in a field width of 20
-	%-20d	Put the value in a field 20 characters wide, left-justified
+	%+20d	Put the value in a field 20 characters wide, right-justified
0	%020d	Put the value in a field 20 characters wide, fill in with leading zeros.
.	%20.2f	Put the value in a field 20 characters wide with 2 characters to the right of the decimal point.
(name)	%(name)d	Get the value from the supplied dictionary using name as the key.

Example :

```
price = 24
item = "banana"
print("The %s costs %d cents"%(item,price))
print("The %+10s costs %5.2f cents"%(item,price))
print("The %+10s costs %10.2f cents"%(item,price))
itemdict = {"item":"banana","cost":24}
print("The %(item)s costs %(cost)7.1f cents"%itemdict)
```

Output :

```
The banana costs 24 cents
The     banana costs 24.00 cents
The     banana costs      24.00 cents
The banana costs    24.0 cents
```

In addition to format strings that use format characters and format modifiers, Python strings also include a format method that can be used in conjunction with a new Formatter class to implement complex string formatting. More about these features can be found in the Python library reference manual.

Mathematical Functions - abs(), divmod(), max(), min(), pow(), sum()

Numbers and Numeric Representation

These functions are used to represent numbers in different forms. The methods are like below "

Function & Description
ceil(x)Return the Ceiling value. It is the smallest integer, greater or equal to the number x.
copysign(x, y)It returns the number x and copy the sign of y to x.
fabs(x)Returns the absolute value of x.
factorial(x)Returns factorial of x. where x e" 0
floor(x)Return the Floor value. It is the largest integer, less or equal to the number x.
fsum(iterable)Find sum of the elements in an iterable object
gcd(x, y)Returns the Greatest Common Divisor of x and y

isfinite(x)Checks whether x is neither an infinity nor nan.
isinf(x)Checks whether x is infinity
isnan(x)Checks whether x is not a number.
remainder(x, y)Find remainder after dividing x by y.

Power and Logarithmic Functions

These functions are used to calculate different power related and logarithmic related tasks.

Function & Description
pow(x, y) Return the x to the power y value.
sqrt(x) Finds the square root of x
exp(x) Finds xe, where e = 2.718281
log(x[, base]) Returns the Log of x, where base is given. The default base is e
log2(x) Returns the Log of x, where base is 2
log10(x) Returns the Log of x, where base is 10

Trigonometric & Angular Conversion Functions

These functions are used to calculate different trigonometric operations.

Function & Description
sin(x) Return the sine of x in radians
cos(x) Return the cosine of x in radians
tan(x) Return the tangent of x in radians
asin(x) This is the inverse operation of the sine, there are acos, atan also.
degrees(x) Convert angle x from radian to degrees
radians(x) Convert angle x from degrees to radian

abs()

Python abs() method returns the absolute value of the given number or list of numbers.

Syntax

```
abs(n)
```

Parameters

The **n** is the required parameter, and it is the **number.** It can be a float or a complex number.

The abs() method returns the absolute value of the given number.

1. For integers – The absolute integer value is returned.
2. For floating numbers – floating absolute value is returned.
3. For complex numbers – the magnitude of the number is returned.

Example :

```
x = abs(-19.21)
y = abs(19 + 21j)
print(x)
print(y)
```

Output :

```
19.21
28.319604517012593
```

divmod()

Division and Modulo are two different but related operations as one returns the quotient and another returns the remainder. But Python divmod() integrates these both operations within a function and returns quotient and remainder as a pair in a tuple.

Syntax

```
divmod(a,b)
```

- **a** (required) – numerator (non-complex)
- **b** (required) – denominator (non-complex)

What does Python divmod() returns?

- divmod() function returns a tuple which contains a pair of the quotient and the remainder like (quotient, remainder).
  - For integers, the return value is the same as (a // b, a % b).
  - For floating point numbers the return value is (q, a % b), where q is usually math.floor(a / b) which is the whole part of the quotient.

Example 1 :

```
print(divmod(10,7))
print(divmod(13.5,7.8))
```

Output :

```
(1, 3)
(1.0, 5.7)
```

As you can see in above example divmod() combines two division operators. It performs an integral division(//) and a modulo division(%) and returns a tuple containing the result of both integral division and modulo division.

Example 2 :

```
x = 5
y = 3
result = divmod(x,y)
#Printing both elements
print(result[0])
print(result[1])
```

Output :

```
(1, 3)
(1.0, 5.7)
Output:
1
2
```

max()

The max() function returns the largest item in an iterable. It can also be used to find the largest item between two or more parameters.

Syntax :

max(iterable, *iterables, key, default)

max() Parameters

- **iterable** - an iterable such as list, tuple, set, dictionary, etc.
- ***iterables (optional)** - any number of iterables; can be more than one
- **key (optional)** - key function where the iterables are passed and comparison is performed based on its return value
- **default (optional)** - default value if the given iterable is empty

Example 1 : find the maximum number

```
numbers = [9, 34, 11, -4, 27]
# find the maximum number
max_number = max(numbers)
print(max_number)
```

Output :

```
34
```

Example 2 : find the largest string

```
languages = ["Python", "C Programming", "Java", "JavaScript"]
largest_string = max(languages);
print("The largest string is:", largest_string)
```

Output

```
The largest string is : Python
```

min()

The min() function returns the smallest item in an iterable. It can also be used to find the smallest item between two or more parameters.

Syntax

```
min(iterable, *iterables, key, default)
```

min() Parameters

- **iterable** - an iterable such as list, tuple, set, dictionary, etc.
- ***iterables (optional)** - any number of iterables; can be more than one
- **key (optional)** - key function where the iterables are passed and comparison is performed based on its return value
- **default (optional)** - default value if the given iterable is empty

Example 1 : find the smallest number

```
numbers = [9, 34, 11, -4, 27]
# find the smallest number
min_number = min(numbers)
print(min_number)
```

Output :

```
-4
```

Example 2 : The smallest string in a list

```
languages = ["Python", "C Programming", "Java", "JavaScript"]
smallest_string = min(languages);
print("The smallest string is:", smallest_string)
```

Output

```
The smallest string is: C Programming
```

pow()

Python pow() function computes a**b. The pow() function first converts its arguments into float and then computes the power.

Syntax

```
pow(a, b, c)
```

Parameter	Description
a	A number, the base
b	A number, the exponent
c	Optional. A number, the modulus

1. If only two arguments are provided, then a to the power of b is returned. In this case, the arguments can be integers, floats, and complex numbers. The two-argument form of pow(a, b) is equivalent to using the power operator: a**b.
2. If three arguments are provided, then a to the power b, modulo c is returned. It's computed more efficiently than using pow(a, b) % c.
3. If c is present, a and b must be of integer types, and b must be non-negative.

Example :

```
p1 = pow(4, 3)
p2 = pow(4, 3, 10)
p3=pow(11, 2.0)
p4=pow(21.0, 2)
print(p1)
print(p2)
print(p3)
print(p4)
```

Output :

```
64
4
121.0
441.0
```

sum()

The sum() function adds the items of an iterable and returns the sum.

Syntax

The syntax of the sum() function is:

- sum(iterable, start)

The sum() function adds start and items of the given iterable from left to right.

sum() Parameters

- **iterable** - iterable (list, tuple, dict, etc). The items of the iterable should be numbers.
- **start** (optional) - this value is added to the sum of items of the iterable. The default value of start is 0 (if omitted)

Example

```
marks = [65, 71, 68, 74, 61]
# find sum of all marks
total_marks = sum(marks)
print(total_marks)
```

Output :

```
339
```

Module

Common Standard Library Modules:

sys:

The sys module contains system-specific functionality. The sys module has a version_info tuple that gives us information about the version.

Example :

```
>>> import sys
>>>sys.version_info
```

Output :

```
sys.version_info(major=3, minor=8, micro=3, releaselevel='final', serial=0)
>>>
```

Date and Time :

The datetime module provides us with the objects which we can use to store information about date and time:

- datetime.date is used to create dates that are not associated with a time.
- datetime.time is used to have time independent of the date.
- datetime.datetime is used for objects which have both date and time.
- datetime.timedelta is used for objects to store differences in dates or datetime.
- datetime.timezone is used for objects that represent time zones as UTC.

We can have queries of these objects for a particular component like the year, month, hour or minute, perform arithmetic functions on them, extract printable string versions from them if we want to display them

Example :

```
import datetime
now = datetime.datetime.today()
print(now.year)
print(now.hour)
print(now.minute)
```

```
print(now.weekday())
print(now.strftime("%a, %d, %B, %Y"))
long_ago = datetime.datetime(1993, 3, 14, 12,30, 58)
print(long_ago)
print(long_ago< now)
difference = now - long_ago
print(type(difference))
print(difference)
```

Output :

```
2021
20
53
5
Sat, 02, October, 2021
1993-03-14 12:30:58
True
<class 'datetime.timedelta'>
10429 days, 8:22:42.532747
```

Math Module

The math module is a collection of mathematical functions. The Python math module offers you the ability to perform common and useful mathematical calculations within your application.Since the math module comes packaged with the Python release, you don't have to install it separately.

Constants of the math Module

The Python math module offers a variety of predefined constants. Having access to these constants provides several advantages. For one, you don't have to manually hardcode them into your application, which saves you a lot of time. Plus, they provide consistency throughout your code. The module includes several famous mathematical constants and important values:

Constants	Description
pi	returns 3.141592
E	returns 0.718282
nan	Not a number
inf	infinite

Example :

```
import math
print( "Constants in Python")
```

```
print(" PI value : " , math.pi)
print(" E value : " , math.e)
print(" nan value : " , math.nan)
print(" E value : " , math.inf)
```

Output :

```
Constants in Python
PI value :  3.141592653589793
E value :  2.718281828459045
nan value :  nan
E value :  inf
```

Example :

```
import math
#These are constant attributes, not functions
math.pi
math.e
#round a float up or down
print (math.ceil(3.3))
print (math.floor(3.3))

#natural algorithm
print (math.log(5))

#logrithm with base 10
print (math.log(5,10))
print (math.log10(5))

# Square root
print (math.sqrt(10))

# trignometric function
print (math.sin(math.pi/2))
print (math.cos(0))
```

Output :

```
4
3
1.6094379124341003
0.6989700043360187
0.6989700043360189
3.1622776601683795
1.0
1.0
```

Random Module

This module is used to generate pseudo-random numbers (sequence of numbers) and also do some more things depending on randomness. The main function of this module generates a random float between 0 and 1 and most of the other functions are derived from it.

Example :

```
import random
# random float from 0 to 1 (excluding 1)
print(random.random())

pets = ['cat','dog','fish']
# choose random.element from the sequnece
print(random.choice(pets))

# shuffle a list (in place)
print(random.shuffle(pets))
print(pets)

# random integer from 1 to 10 (inclusive)
print(random.randint(1, 10))
```

Output :

```
0.532840047288826
cat
None
['cat', 'fish', 'dog']
7
```

Statistics Module

Python statistics module provides the functions to mathematical statistics of numeric data. There are some popular statistical functions defined in this module.

mean() function

The mean() function is used to calculate the arithmetic mean of the numbers in the list.

Syntax

```
statistics.mean(data)
```

Example :

```
import statistics
# list of positive integer numbers
datasets = [5, 2, 7, 4, 2, 6, 8]
x = statistics.mean(datasets)
# Printing the mean
print("Mean is :", x)
```

Output :

```
Mean is : 4.857142857142857
```

median() function

The median() function is used to return the middle value of the numeric data in the list.

Syntax

```
statistics. median(data)
```

Example :

```
import statistics
datasets = [4, -5, 6, 6, 9, 4, 5, -2]
# Printing median of the
# random data-set
print("Median of data-set is : % s "
    % (statistics.median(datasets)))
```

Output :

```
Median of data-set is : 4.5
```

mode() function

The mode() function returns the most common data that occurs in the list.

Syntax

```
statistics. mode(data)
```

Example :

```
import statistics
# declaring a simple data-set consisting of real valued positive integers.
dataset =[2, 4, 7, 7, 2, 2, 3, 6, 6, 8]
# Printing out the mode of given data-set
print("Calculated Mode % s" % (statistics.mode(dataset)))
```

Output :

```
Calculated Mode 2
```

stdev() function

The stdev() function is used to calculate the standard deviation on a given sample which is available in the form of the list.

Syntax

```
statistics. stdev(data)
```

Example :

```
import statistics
# creating a simple data - set
sample = [7, 8, 9, 10, 11]
# Prints standard deviation
print("Standard Deviation of sample is % s "
        % (statistics.stdev(sample)))
```

Output :

```
Standard Deviation of sample is 1.5811388300841898
```

median_low()

The median_low function is used to return the low median of numeric data in the list.

Syntax

```
statistics. median_low(data)
```

Example :

```
import statistics
# simple list of a set of integers
set1 = [4, 6, 2, 5, 7, 7]
# Note: low median will always be a member of the data-set.
# Print low median of the data-set
print("Low median of data-set is % s "
    % (statistics.median_low(set1)))
```

Output :

```
Low median of the data-set is 5
```

median_high()

The median_high function is used to return the high median of numeric data in the list.

Syntax

```
statistics. median_high(data)
```

Example :

```
import statistics
# list of set of the integers
dataset = [2, 1, 7, 6, 1, 9]
print("High median of data-set is %s "
      % (statistics.median_high(dataset)))
```

Output :

```
High median of the data-set is 6
```