

Python language: Functions, modules and objects

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

SciPy.in 2010, Tutorials

Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

Functions

- **def** - keyword to define a function
- Arguments are local to a function
- Functions can return multiple values

Functions: example

```
def signum( r ):  
    """returns 0 if r is zero  
    -1 if r is negative  
    +1 if r is positive"""  
    if r < 0:  
        return -1  
    elif r > 0:  
        return 1  
    else:  
        return 0
```

Note docstrings

What does this function do?

```
def what( n ):  
    if n < 0: n = -n  
    while n > 0:  
        if n % 2 == 1:  
            return False  
        n /= 10  
    return True
```

What does this function do?

```
def what( n ):  
    i = 1  
    while i * i < n:  
        i += 1  
    return i * i == n, i
```

What does this function do?

```
def what ( x, n ) :  
    if n < 0 :  
        n = -n  
        x = 1.0 / x  
  
    z = 1.0  
    while n > 0 :  
        if n % 2 == 1 :  
            z *= x  
        x *= x  
        n /= 2  
    return z
```


Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

Problem 3.1

Write a function to return the gcd of two numbers.

Problem 3.2

Write a program to print all primitive pythagorean triads (a, b, c) where a, b are in the range 1—100

A pythagorean triad (a, b, c) has the property $a^2 + b^2 = c^2$.

By primitive we mean triads that do not ‘depend’ on others. For example, $(4,3,5)$ is a variant of $(3,4,5)$ and hence is not primitive. And $(10,24,26)$ is easily derived from $(5,12,13)$ and is also not primitive.

Problem 3.3

Write a program that generates a list of all four digit numbers that have all their digits even and are perfect squares.

For example, the output should include 6400 but not 8100 (one digit is odd) or 4248 (not a perfect square).

Outline

1 Functions

- Exercises
- **Default arguments**
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

Functions: default arguments

```
In []: greet = 'hello world'
```

```
In []: greet.split()
```

```
Out[]: ['hello', 'world']
```

```
In []: line = 'Rossum, Guido, 54, 46, 55'
```

```
In []: line.split(',')
```

```
Out[]: ['Rossum', ' Guido', ' 54',  
        ' 46', ' 55']
```

Functions: default arguments ...

```
In []: def welcome(greet, name="World"):
      .... print greet, name
```

```
In []: welcome("Hello")
Hello World
```

```
In []: welcome("Hi", "Guido")
Hi Guido
```

Outline

1 Functions

- Exercises
- Default arguments
- **Keyword arguments**
- Built-in functions

2 Modules

3 Objects

Functions: Keyword arguments

We have seen the following

```
legend(['sin(2y)'], loc = 'center')
```

```
plot(y, sin(y), 'g', linewidth = 2)
```

```
annotate('local max', xy = (1.5, 1))
```

```
pie(science.values(),  
    labels = science.keys())
```

Functions: keyword arguments ...

```
In []: def welcome(greet, name="World"):
      .... print greet, name
```

```
In []: welcome("Hello", "James")
Hello James
```

```
In []: welcome("Hi", name="Guido")
Hi Guido
```

```
In []: welcome(name="Guido", greet="Hey")
Hey Guido
```

Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- **Built-in functions**

2 Modules

3 Objects

Before writing a function

- Variety of built-in functions are available
- **`abs, any, all, len, max, min`**
- **`pow, range, sum, type`**
- Refer here: <http://docs.python.org/library/functions.html>

Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

from ... import magic

```
from scipy.integrate import odeint
```

```
from scipy.optimize import fsolve
```

Above statements import a function to our namespace

Running scripts from command line

- Fire up a terminal
- `python four_plot.py`

```
Traceback (most recent call last):  
  File "four_plot.py", line 1, in <module>  
    x = linspace(-5*pi, 5*pi, 500)  
NameError: name 'linspace' is not defined
```

Running scripts from command line

- Fire up a terminal
- `python four_plot.py`

```
Traceback (most recent call last):  
  File "four_plot.py", line 1, in <module>  
    x = linspace(-5*pi, 5*pi, 500)  
NameError: name 'linspace' is not defined
```


Remedy ...

```
from scipy import *
```

Now run `python four_plot.py` again

Now what?

Traceback (most recent call last):

File "four_plot.py", line 1, in <module>

x = plot(x, x, 'b')

NameError: name 'plot' is not defined

Remedy ...

```
from pylab import *
```

Now run `python four_plot.py` again!!

Modules

- The `import` keyword “loads” a module
- One can also use:

```
In []: from scipy import *
```

```
In []: from scipy import linspace
```

- What is the difference?
- Use the former only in interactive mode

IPython namespace

```
In [4]: whos  
Interactive namespace is empty.
```

IPython namespace

```
In [5]: from numpy import *
```

```
In [6]: whos
```

Package hierarchies

```
from scipy.integrate import odeint
```

```
from scipy.optimize import fsolve
```

from...import - conventional way!

```
from scipy import linspace, pi, sin
from pylab import plot, legend, annotate
from pylab import xlim, ylim
```

```
x = linspace(-5*pi, 5*pi, 500)
plot(x, x, 'b')
plot(x, -x, 'b')
plot(x, sin(x), 'g', linewidth=2)
plot(x, x*sin(x), 'r', linewidth=3)
legend(['x', '-x', 'sin(x)', 'xsin(x)'])
annotate('origin', xy = (0, 0))
xlim(-5*pi, 5*pi)
ylim(-5*pi, 5*pi)
```


from...import - conventional way!

```
import scipy
import pylab
```

```
x = scipy.linspace(-5*scipy.pi, 5*scipy.pi, 500)
pylab.plot(x, x, 'b')
pylab.plot(x, -x, 'b')
pylab.plot(x, scipy.sin(x), 'g', linewidth=2)
pylab.plot(x, x*scipy.sin(x), 'r', linewidth=3)
pylab.legend(['x', '-x', 'sin(x)', 'xsin(x)'])
pylab.annotate('origin', xy = (0, 0))
pylab.xlim(-5*scipy.pi, 5*scipy.pi)
pylab.ylim(-5*scipy.pi, 5*scipy.pi)
```

import - the community convention

```
import numpy as np  
np.linspace(-5 * np.pi, 5 * np.pi, 500)
```

```
import scipy as sp  
sp.linspace(-5 * sp.pi, 5 * sp.pi, 500)
```

Modules: Standard library

- Very powerful, “Batteries included”
- Some standard modules:
 - Math: **math**, **random**
 - Internet access: **urllib2**, **smtplib**
 - System, Command line arguments: **sys**
 - Operating system interface: **os**
 - Regular expressions: **re**
 - Compression: **gzip**, **zipfile**, and **tarfile**
 - And a whole lot more!
- Check out the Python Library reference:
<http://docs.python.org/library/>

Modules of special interest

pylab Easy, interactive, 2D plotting

scipy arrays, statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing, genetic algorithms, ODE solvers, special functions, and more

mayavi Easy, interactive, 3D plotting

From which module?

This plot function in pylab is cool, from where do I import it to include it in `my_nifty_module.py`?

From which module?

```
In [15]: plot.__module__
```

```
Out[15]: 'matplotlib.pyplot'
```

```
from matplotlib.pyplot import plot
```

Outline

1 Functions

- Exercises
- Default arguments
- Keyword arguments
- Built-in functions

2 Modules

3 Objects

Everything is an Object!

- `int`
- `float`
- `str`
- `list`
- `tuple`
- `string`
- `dictionary`
- `function`
- User defined class is also an object!

Using Objects

- Creating Objects

- Initialization

```
In []: a = str()
```

```
In []: b = "Hello World"
```

- Object Manipulation

- Object methods
 - "." operator

```
In []: "Hello World".split()
```

```
Out[]: ['Hello', 'World']
```

Objects provide consistency

```
for element in (1, 2, 3):  
    print element  
for key in {'one':1, 'two':2}:  
    print key  
for char in "123":  
    print char  
for line in open("myfile.txt"):  
    print line  
for line in urllib2.urlopen('http://site.com'):  
    print line
```

What did we learn?

- Functions: Definition
- Functions: Docstrings
- Functions: Default and Keyword arguments
- Modules
- Objects