Branch: master ▾ Find file Copy path

ma591spring2020 / material / modules.ipynb

 asaibab Add files via upload

29307b5 on Jan 30

1 contributor

 Raw Blame History   

271 lines (270 sloc) 5.84 KB

Modules

Python modules allow us to define multiple functions in a single script file, which we can then use and reuse multiple times. We will learn how to use modules in this lecture; writing modules will be left for a separate lecture.

Python has several in-built modules. Some useful ones are

- `math`, `cmath`, `os`, `pdb` (debugger), `profile` (profiler). There are many third party modules/packages that we will explore (e.g., matplotlib, numpy, scipy).

Each module is a collection of different functions (methods), variables (attributes), and classes. This defines a *namespace* We can access their functionalities in several different ways.

Method 1: `import`

In [1]:
```
import math
print(math.cos(math.pi))              #Can be painful if the
 module name is long
```

```
-1.0
```

Method 2: Import a module using an alias.

In [4]:
```
import math as m
print(m.cos(m.pi))
```

```
-1.0
```

Method 3: Importing contents directly

In [5]:
```
from math import cos, pi
print(cos(pi))
```

```
-1.0
```

In [6]:
```
from cmath import cos, pi
print(cos(pi))
```

```
(-1-0j)
```

This example shows the pitfalls of importing the functions directly. Different namespaces may have similar functions; use the module names (namespaces) to distinguish between these functions.

Method 4: Importing all the contents

This method can only be used at the start of a script and not inside a function. Do not use, if you can avoid it!

In [2]:
```
from math import * # * is a wildcard for all the contents
print(cos(pi))
```

```
-1.0
```

Writing your own modules

Let us create a script called `quad.py'

In [ ]:
```
import math

__all__ = ['quadratic', 'Parabola']

def quadratic(x, a = 1., b = 2., c = 3.):
    return a*x**2. + b*x + c


class Parabola:
    def __init__(self, a0, a1, a2):
        self.a0 = a0
        self.a1 = a1
        self.a2 = a2

    def eval(self, x):
        return self.a0 + self.a1*x + self.a2*x**2.

    def __call__(self, x):
        return self.eval(x)
```

Once this file `quad.py` is in your directory, any of these commands should work.

```
import quad
import quad as qd
from quad import quadratic, Parabola
from quad import *
```

In [2]:
```
import quad
import quad as qd
from quad import quadratic, Parabola
from quad import *
```

In [3]:
```
print(quadratic(1.0))
```

```
6.0
```

Executing a module as a script

At the bottom the script, we can add a few lines of code, like

```
if __name__ == '__main__':
    print(quadratic(1.0))
```

When we execute the script at a terminal using the line

```
python quad.py
```