More Basic Python

FOSSEE

Department of Aerospace Engineering
IIT Bombay

Outline

Outline

hello.py

- Script to print 'hello world' – `hello.py`

```
print "Hello world!"
```

- We have been running scripts from IPython

```
In[]: %run -i hello.py
```

- Now, we run from the shell using python

```
$ python hello.py
```

Modules

- Organize your code
- Collect similar functionality
- Functions, classes, constants, etc.

Modules

- Define variables, functions and classes in a file with a `.py` extension
- This file becomes a module!
- The `import` keyword "loads" a module
- One can also use:
  `from module import name1, name2, name2`
  where `name1` etc. are names in the module, "module"
- `from module import *` — imports everything from module, use only in interactive mode
- File name should be valid variable name

Modules: example

```
# --- foo.py ---
some_var = 1
def fib(n): # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
# EOF
```

Modules: example

```
>>> import foo
>>> foo.fib(10)
1 1 2 3 5 8
>>> foo.some_var
1
```

Python path

- In IPython type the following

```
import sys
sys.path
```

- List of locations where python searches for a module
- `import sys` – searches for file `sys.py` or dir `sys` in all these locations
- So, our own modules can be in any one of the locations
- Current working directory is one of the locations
- Can also set `PYTHONPATH` env var

Another example: GCD script

- Function that computes gcd of two numbers
- Save it as `gcd_script.py`

```
def gcd(a, b):
    while b:
        a, b = b, a%b
    return a
```

- Also add the tests to the file

```
if gcd(40, 12) == 4 and gcd(12, 13) == 1:
    print "Everything OK"
else:
    print "The GCD function is wrong"
```

```
$ python gcd_script.py
```

__name__

```
import gcd_script
```

- The import is successful
- But the test code, gets run
- Add the tests to the following `if` block

```
if __name__ == "__main__":
```

- Now the script runs properly
- As well as the import works; test code not executed
- `__name__` is local to every module and is equal to `__main__` only when the file is run as a script.

Stand-alone scripts

Consider a file `f.py`:

```
#!/usr/bin/env python
"""Module level documentation."""
# First line tells the shell that it should use Python
# to interpret the code in the file.
def f():
    print "f"

# Check if we are running standalone or as module.
# When imported, __name__ will not be '__main__'
if __name__ == '__main__':
    # This is not executed when f.py is imported.
    f()
```

Outline

Motivation

- How do you signal errors to a user?

Exceptions

- Python's way of notifying you of errors
- Several standard exceptions: `SyntaxError`, `IOError` etc.
- Users can also `raise` errors
- Users can create their own exceptions
- Exceptions can be "caught" via `try/except` blocks

Exception: examples

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' obj
```

Exception: examples

```
>>> while True:
...     try:
...         x = int(raw_input("Enter a number: "))
...         break
...     except ValueError:
...         print "Invalid number, try again..."
...
>>> # To raise exceptions
... raise ValueError("your error message")
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
ValueError: your error message
```

Exception: try/finally

```
>>> while True:
...     try:
...         x = open("my_data.txt")
...         lines = x.readlines()
...         # Process the data from the file.
...         value = int(line[0])
...     except ValueError:
...         print "Invalid file!"
...     finally:
...         print "All good!"
...
```