

Python Shell vs. Modules

Terminal Screenshot:

```
Last login: Fri Aug 28 12:34:11 on ttys001
(wmwhite@WmLeh)-> python
Python 2.7.10 (default, Sep 9 2014, 15:04:36)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (Clang-600.0.39)]
on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 1+2
>>> x = 3*x
>>> x
0
>>>
```

Text Editor Screenshot:

module.py

```
1 # module.py
2 # Walker M. White (wmw2)
3 # June 28, 2012
4
5 """This is a simple module.
6 It shows how modules work."""
7
8 x = 1+2
9 x = 3*x
10 print x
```

- Launch in command line
- Type each line separately
- Python executes as you type

- Write in a text editor
 - We use Komodo Edit
 - But anything will work
- Run module with import

Using a Module

Module Contents

module.py

Single line comment
(not executed)

""" This is a simple module.
It shows how modules work """

Docstring (note the Triple Quotes)
Acts as a multiple-line comment
Useful for [code documentation](#)

x = 1+2

Commands
Executed on import

x

Not a command.
import ignores this

Using a Module

Module Contents

```
# module.py
"""
This is a simple module.
It shows how modules work"""

x = 1+2
x = 3*x
x
```

Module data must be prefixed by module name

Prints docstring and module contents

Python Shell

```
>>> import module
>>> x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>> module.x
9
>>> help(module)
```

We Write Programs to Do Things

- Functions are the key doers

Function Call

- Command to do the function

greet('Walker')

argument to assign to n

Function Definition

- Defines what function does

def greet(n):

print 'Hello '+n+'!'

declaration of parameter n

Function Body (indented)

Parameter: variable that is listed within the parentheses of a method header.

Argument: a value to assign to the method parameter when it is called

Anatomy of a Function Definition

name parameters

def greet(n):

Function Header

"""Prints a greeting to the name n

Docstring Specification

Parameter n: name to greet

Precondition: n is a string"""

print 'Hello '+n+'!'

Statements to execute when called

print 'How are you?'

The vertical line indicates indentation

Use vertical lines when you write Python on exams so we can see indentation

Procedures vs. Fruitful Functions

Procedures

- Functions that do something
- Call them as a statement
- Example: greet('Walker')

Fruitful Functions

- Functions that give a value
- Call them in an expression
- Example: x = round(2.56,1)

Historical Aside

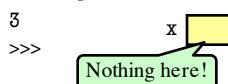
- Historically “function” = “fruitful function”
- But now we use “function” to refer to both

The `return` Statement

- Fruitful functions require a `return statement`
- Format:** `return <expression>`
 - Provides value when call is used in an expression
 - Also stops executing the function!
 - Any statements after a `return` are ignored
- Example:** temperature converter function

```
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0
```

Print vs. Return

Print	Return
<ul style="list-style-type: none">Displays a value on screen<ul style="list-style-type: none">Used primarily for <code>testing</code>Not useful for calculations	<ul style="list-style-type: none">Defines a function's value<ul style="list-style-type: none">Important for <code>calculations</code>But does not display anything
<pre>def print_plus(n): print (n+1) >>> x = plus_one(2) 3 >>></pre> 	<pre>def return_plus(n): return (n+1) >>> x = plus_one(2) >>></pre> 

Functions and Modules

- Purpose of modules is **function definitions**
 - Function definitions are written in module file
 - Import the module to call the functions
- Your Python workflow (right now) is
 - Write a function in a module (a .py file)
 - Open up the command shell
 - Move to the directory with this file
 - Start Python (type `python`)
 - Import the module
 - Try out the function

9/3/15

Defining Functions

9

Aside: Constants

- Modules often have variables outside a function
 - We call these global variables
 - Accessible once you import the module
- Global variables should be **constants**
 - Variables that never, ever change
 - Mnemonic representation of important value
 - Example:** `math.pi`, `math.e` in `math`
- In this class, constant names are **capitalized**!
 - So we can tell them apart from non-constants

Module Example: Temperature Converter

```
# temp.py
"""Conversion functions between fahrenheit and centigrade"""

# Functions
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0

def to_fahrenheit(x):
    """Returns: x converted to fahrenheit"""
    return 9*x/5.0+32

# Constants
FREEZING_C = 0.0 # temp. water freezes
```

Style Guideline:
Two blank lines between function definitions

Example from Previous Lecture

```
def second_in_list(s):
    """Returns: second item in comma-separated list
Parameter s: The list of items
Precondition: s is a string of items separated by commas.
The final result does not have any whitespace on edges"""

startcomma = s.index(',')
tail = s[startcomma+1:]
endcomma = tail.index(',')
item = tail[:endcomma].strip()
return item
```

See commalist.py