DATA 520
Lecture 8
Modules

Unleashing more code

Modules

A Module is a collection of functions (often algorithms)
and variables (usually constants) in a single file

Modules are usually based on a theme

- String handler, or searcher, or parser
- Statistical functions (general), or Matrix operations, or Machine Learning
- Time and Date functions
- Encryption / Decryption / Compression (ZIP, tar, etc.)
- Python things (version, etc.)
- SYSTEM things (time, OS, OS version, disk space, etc.)

Modules

To use a module, import it.

```
import math  # math things based on a C library
help(math)
```

- take a look

e **(2.718281828459045),** ***pi*** **(π) (3.141592653589793)**

sin, cos, convert radians to degrees, different rounding functions, hypotenuse,

```
math.ceil(x)   This is the smallest integer >= x.
math.floor(x)  This is the largest integer <= x.
inf, nan
```

Modules

```
sqrt(9)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    sqrt(9)
NameError: name 'sqrt' is not defined

# format is <library name>.<function>
math.sqrt(9)
#  be careful!!!!!!
math.pi
3.141592653589793
math.pi = 56
math.pi
56
```

```
# To reset a messed up module:
You must exit the shell and reload!
- reloading does not fix things like math.pi = 56
```

Modules

You can selectively import some functions

- you do not have to import ALL of them

```
# from <module name> <function or variable name>

from math import sqrt, pi  # OR from math import *
# NOW you can use functions this way:
sqrt(9)
```

- they are now in the NAMESPACE (RAM)

NOTE: There are few rules for naming functions

If you use a module that has **`Superfunction()`**,

then import another module with **`Superfunction()`**, what will happen?

- previous one is forgotten (overwritten in RAM)

Modules

Python's built-in functions

```
dir(__builtins__)
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError',
'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis',
'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError',
'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError',
'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError',
'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration',
'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError',
'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError',
'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '_',
'__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__',
'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod',
'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec',
'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex',
'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max',
'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range',
'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super',
'tuple', 'type', 'vars', 'zip']
```

Modules

Defining your own modules

save as temperature.py

then import:

```
import temperature
```

A module is loaded only once

```
# temperature.py
print ('temperature module loaded and ready!')

def convert_to_celsius(fahrenheit):
    """ (number) -> float

    Return the number of Celsius degrees equivalent to fahrenheit
degrees.

    >>> convert_to_celsius(75)
    23.88888888888889
    """
    return (fahrenheit - 32.0) * 5.0 / 9.0

def above_freezing(celsius):
    """ (number) -> bool

    Return True iff temperature celsius is above freezing.

    >>> above_freezing(5.2)
    True
    >>> above_freezing(-2)
    False
    """

    return celsius > 0
```

Modules

```
# temperature.py - code
print ('temperature module loaded and ready!')

def convert_to_celsius(fahrenheit):
    """ (number) -> float

    Return the number of Celsius degrees equivalent to fahrenheit
degrees.

    >>> convert_to_celsius(75)
    23.88888888888889
    """
    return (fahrenheit - 32.0) * 5.0 / 9.0

def above_freezing(celsius):
    """ (number) -> bool

    Return True iff temperature celsius is above freezing.

    >>> above_freezing(5.2)
    True
    >>> above_freezing(-2)
    False
    """

    return celsius > 0
```

Modules

Where are the modules found?

```
import sys

help(sys)

sys.path  # python locations for files
```

```
['C:/Users/sousley/AppData/Local/Programs/Python/Python36-32',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32\\Lib\\idlelib',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32\\python36.zip',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32\\DLLs',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32\\lib',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32',
'C:\\Users\\sousley\\AppData\\Local\\Programs\\Python\\Python36-32\\lib\\site-packages']
```

```
os.getcwd() # to find default current working directory:
```

```
'C:\\Users\\Steve9\\AppData\\Local\\Programs\\Python\\Python36-32'
```

```
sys.platform
```

```
'win32'
```

```
sys.float_info


# see what is loaded into the namespace

dir()
```

```
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'above_freezing', 'convert_to_celsius', 'sys', 'temperature', 'x']
```

Modules

Create and save a file called testmodule.py with:

```
# testmodule.py
print ('the test module has started')

>>> import testmodule
the test module has started
>>> import testmodule
[nothing. nada. Nichts.]

# add this line to testmodule.py,
print("__name__ is", __name__)
# restart shell
>>> import testmodule
the test module has started
__name__ is testmodule
```

Modules

Next, add this at the bottom of testmodule.py:

```
if __name__ == "__main__":
    # called from itself
    print("I am the main program.")
else:
    # (Called from the Shell)
    print("Another module is importing me.")

# restart shell, then type:
>>> import testmodule
the test module has started
__name__ is testmodule
Another module is importing me.
```

Modules

Next, run the same program from the editor window (F5)

```
RESTART: C:/Users/Steve9/AppData/Local/Programs/Python/Python36-32/Lib/testmodule.py
the test module has started
__name__ is __main__
I am the main program.
```

Modules

Next, then add this to top of of temperature.py:

```
import testmodule
```

then add this at the bottom:

```
# run only first time
# if __name__ == '__main__':
    fahrenheit = float(input('Enter the temperature in degrees Fahrenheit: '))
    celsius = convert_to_celsius(fahrenheit)
    if above_freezing(celsius):
        print('It is ABOVE freezing.')
    else:
        print('It is below freezing.')
```

Run (F5) What happens?

Now uncomment the `if __name__ == '__main__':` line above and run

Modules

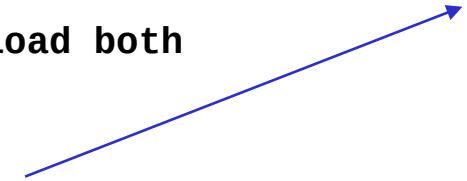
There is also a module to test modules!

- it tests the examples in the docstrings

```
>>> import doctest
>>> help(doctest)
>>> import temperature # necessary
>>> doctest.testmod(temperature)
TestResults(failed=0, attempted=3)

# mess up code, move right paren and save:
return (fahrenheit - 32.0 * 5.0 / 9.0)

# restarts namespace, so reload both
import doctest
import temperature
doctest.testmod(temperature)
```



```
**********************************************************************
File "C:/Users/sousley/AppData/Local/Programs/Python/Python36-
32\temperature.py", line 8, in temperature.convert_to_celsius
Failed example:
    convert_to_celsius(75)
Expected:
    23.88888888888889
Got:
    57.22222222222222
**********************************************************************
1 items had failures:
   1 of   1 in temperature.convert_to_celsius
***Test Failed*** 1 failures.
TestResults(failed=1, attempted=3)
```

verbose doctest

```
doctest.testmod(temperature, verbose=True)
Trying:
    above_freezing(5.2)
Expecting:
    True
ok
Trying:
    above_freezing(-2)
Expecting:
    False
ok
Trying:
    convert_to_celsius(75)
Expecting:
    23.88888888888889
ok
1 items had no tests:
    temperature
2 items passed all tests:
   2 tests in temperature.above_freezing
   1 tests in temperature.convert_to_celsius
3 tests in 3 items.
3 passed and 0 failed.
Test passed.
TestResults(failed=0, attempted=3)
```

Modules

We can use libraries for graphics too - turtle graphics

in the file editor, open :

```
C:\Users\sousley\AppData\Local\Programs\Python\Python36-32\Lib\turtledemo\__main__.py
```

and we will run some demos

Modules

Online: google search or

https://www.catswhocode.com/blog/python-50-modules-for-all-needs

https://pythontips.com/2013/07/30/20-python-libraries-you-cant-live-without/

With so many modules, how can we install them? with the help of a module!

```
# sys.path # find location of python.exe on Lab computers, should be c:\..\36-32\
# at command prompt on your own PC:(must point to python.exe), so cd or include file location:
C:\Users\sousley\AppData\Local\Programs\Python\Python36-32\python -m pip install -U pip setuptools
# import matplotlib
C:\Users\sousley\AppData\Local\Programs\Python\Python36-32\python -m pip install matplotlib
# import mpmath
C:\Users\sousley\AppData\Local\Programs\Python\Python36-32\python -m pip install mpmath

# mpmath: high precision math - extended digits (up to 1,000!)
>>> from mpmath import mp
>>> mp.dps = 100
>>> print(mp.quad(lambda x: mp.exp(-x**2), [-mp.inf, mp.inf]) ** 2)
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706
```

Homework 6 due before class Monday

Find an online Python module that does something you are interested in

- Mandelbrot plots!

- other visualization

- timing program execution

- measuring memory use

- ????

Email me an idea (first come, first served) to register it asap

(1^{st} ,2^{nd} , 3^{rd} choice will help you)

I will approve if not already taken

I will post updated lists on Blackboard