Notation Used for teaching materials:

Python code/ keywords: green text
example:

```
# write a 3-line file.
my_out_file = open('file_a.txt', 'w')
my_out_file.write('This is line 1.\n')
my_out_file.write('Line 2 follows line 1.\n')
```

Python code executed in Python shell: green text preceeded by ">>>"
example:

```
>>>import myfirstpackage.module1
```

Commands executed in (bash) shell: blue text preceeded by "$"
example:

```
$ex31.py test
```

File name text enclosed in "<" and ">"
example:
<ex31.py>

Suggestion for more research/leraning : orange text preceeded by "R!" (you should always do your own research anyway regardless!)
example:

```
R! Do more research on methods related to file object,
e.g write, close, etc
```

Class Activities: To be run/conducted in class: red text preceeded by "CLASS ACTIVITY:"
example:

```
CLASS ACTIVITY:
```

Function

A device that groups a set of statements so they can be run more than once in a program.

Built-in functions (requires no import)

```
# write a 3-line file.
my_out_file = open('file_a.txt', 'w')
my_out_file.write('This is line 1.\n')
my_out_file.write('Line 2 follows line 1.\n')
my_out_file.write('Another line makes 3 lines.\n')
my_out_file.close()

# read the file
my_file = open('file_a.txt')
all_lines = my_file.read()
my_file.close()
print all_lines
```

R! Do more research on methods related to file object, e.g write, close, etc

Defining a function:

```
>>>def add_print(a, b):
>>>  print a+b

>>>add_print(2,5)

>>>def add_return(a, b):
>>>  return a+b

>>>add_return(2,5)



>>>def swap_var(a, b):
>>>  print a, b
>>>  return b, a

>>>swap_var(5, 10)
>>>swap_var(10, 'this is a string')

>>>def min_var(a, b=0):
```

```
>>>  print a, b
>>>  return min(a, b)

>>>min_var(10)
>>>min_var(10, 20)
```

Modules

The highest-level program organization unit that packages program code and data for reuse.

import
Let a client (importer) fetch a module as a whole
from
Allows clients to fetch particular names from a module
reload
Provides a way to reload a module's code without stopping Python

A Python program may consist of a top-level file and module files used by the top level file.

<b.py>

```
def func():
    print __name__
    return 0

if __name__ == '__main__':
  func()
```

(*) we will cover this "if" block later in <ex32.py> below

<a.py>

```
import b
b.func()
```

#note that a module is referred to by its file name(without the "py" extension)
`# import b as mymodule` will allow you to refer to it as "mymodule" or any name you specify

CLASS ACTIVITY: Run the example above (5 min)

Using standard libraries (example: “sys”)

<ex31.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#this line is to be completed
import sys

print sys.path
print

if len(sys.argv) > 1:
    if sys.argv[1] == 'cns':
        print "Hello World!"
    elif sys.argv[1] == 'sfc':
        print "Hello SFC!"
    else:
        print "Hello",sys.argv[1]
```

(*)argv is the argument used when executing python file, in the example above (of course you can have multiple arguments that will be referred to as argv[n])

```
$ex31.py test
$ex31.py cns
$ex.py sfc
```

An example of a module file

<ex32.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def multiply(x,y):
    '''Multiply two numbers'''
    return x*y

def intersect(s_a, s_b):
    res = []
    for x in s_a:
        if x in s_b:
            res.append(x)
    return res
```

```
if __name__ == "__main__":
    print multiply(2,2.5)
    a_list = [1,2,3]
    b_set = {2,3,4}
    print intersect(a_list, b_set)
```

Using your module (<ex33.py> imports ex32 module)
To call a function in a module you(usually) call it this way(after importing it) :

<module name>.<function name>

<ex33.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import ex32

print ex32.multiply(5,2.5)
this_list = [1,2,3,4]
this_set = {2,3,4,5}
print ex32.intersect(this_list, this_set)
```

<ex34.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from ex32 import multiply

print multiply(5,2.5)
this_list = [1,2,3,4]
this_set = {2,3,4,5}
print intersect(this_list, this_set)
```

<ex35.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from ex32 import *

print multiply(5,2.5)
this_list = [1,2,3,4]
```

```
this_set = {2,3,4,5}
print intersect(this_list, this_set)
```

CLASS ACTIVITY: Run the example above(8 min)

Package:

a directory of Python code (containing modules)
The package directory must contain a special file (an empty file, in this example)named __init__.py so it can be imported correctly/recognised by python compiler as a package

```
$mkdir myfirstpackage
$cd myfirstpackage
$touch __init__.py
```

Creating modules for the package that you created above ("myfirspackage")

<module1.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def function1(param1):
  return 10*param1

if __name__ == '__main__':
  print function1('mytest')
```

<another_module.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def another_function(param1):
  return 5*param1

if __name__ == '__main__':
  print another_function ('mytest')
```

Execute the below from python shell , from the directory one level up of "myfirstpackage"

```
>>>import myfirstpackage.module1
>>>myfirstpackage.module1.function1('a string')
```

You can have hierarchical structure for module by creating a sub directories (this will be covered in exercise later)

Modules/packages can be found by your python by setting PYTHONPATH environment variables

CLASS ACTIVITY: Run the example above-8 min (create files as required-<module1.py> and <another_module.py> will be used in the exercise)

Also try adding this line in <ex32.py>
print "you have imported"+ __name__
Run ex32.py on its own, then import ex32.py as a module.
Observe if there is any differnce on what is being printed out by that line

Additional Materials:

CLASS ACTIVITY: You *could* try these first before doing the exercise

dir()

To check what functions and modules available under a package from python prompt:

```
>>>dir(myfirstpackage)
```

This dir() command also works on modules

Reminder/refresher
Iterating (loop) in dictionary(lesson 2)
`iteritems,iterkeys,itervalues` **(python 2 only)**
https://docs.python.org/2/library/stdtypes.html?highlight=iteritems#dict.iteritems

Examples:

```
for k,v in d.iteritems():
for k in d.iterkeys():
for k,v in d.itervalues():
```

the new syntax is without "iter": items,keys,values

Exercise code (incomplete)
(instruction in a separate slide)
Calorie and Expense Tracker

<calexp.py>

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
menu/meal   cal     price
-------------------------
beef        500     300
chicken     250     180
fish        200     300
veggies     100     100
rice        200     120
ramen       800     300

activity    cal/30min
---------------------
class       150
rest        50
walk        250
run         500
bike        400
"""

def intake(consumption):
    '''Returns the total calorie intake based on the
consumptions in one day.'''
    calories = 0

    return calories

def expenses(consumption):
    '''Returns the total expenses based on the consumptions
    in one day.'''
    paid = 0

    return paid

def release(activities):
    '''Returns the total calorie release based on the
activities in one day.'''
    calories = 0
```

```
return calories
```