Python Modules

- A module allows you to logically organize your Python code.
  - Grouping related codes into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that you can bind and reference.
  - Simply, a module is a file consisting of Python code.
- A module can define functions, classes and variables.
- A module can also include runnable code.

The *import* Statement

- You can use any Python source file as a module by executing an import statement in some other Python source file.
  The *import* has the following syntax:

```
import module1[, module2[,... moduleN]
```

- When the interpreter encounters an import statement, it imports the module if the module is present in the search path.
- A search path is a list of directories that the interpreter searches before importing a module.

```
In [3]: import sys

In [4]: sys.modules.keys()
```

```
Out[4]: dict_keys(['sys', 'builtins', '_frozen_importlib', '_imp', '_thread', '_warnings', '_weakref', 'zipimport',
'_frozen_importlib_external', '_io', 'marshal', 'nt', 'winreg', 'encodings', 'codecs', '_codecs', 'encodings.aliases',
'encodings.utf_8', '_signal', '__main__', 'encodings.latin_1', 'io', 'abc', '_abc', '_bootlocale', '_locale',
```

Example

- To import the module *samply_module.py*, you need to put the following command at the top of the script:

temp.py | samply_module.py | untitled1.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Apr 18 16:19:33 2020
4
5  @author: user
6  """
7
8
9  def sample_func():
10     print('Hello!')
```

temp.py | samply_module.py | untitled1.py*

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Apr 18 16:21:50 202
4
5  @author: user
6  """
7
8
9  import samply_module
10 if __name__ == '__main__':
11     samply_module.sample_func()
```

```
In [12]: runfile('C:/Users/user/untitled1.py', wdir='C:/Users/user')
Reloaded modules: samply_module
Hello!
```

Example

- xmath.py

```python
def max(a, b):
    return a if a > b else b
def min(a, b):
    return a if a < b else b

def sum(*numbers): # numbers 接受可變長度引數
    total = 0
    for number in numbers:
        total += number
    return total

pi = 3.141592653589793
e = 2.718281828459045
```

```
# import xmath
3.14159265359
10
15
# import xmath as math
2.71828182846
# from xmath import min
5
```

```python
import xmath
print '# import xmath'
print xmath.pi
print xmath.max(10, 5)
print xmath.sum(1, 2, 3, 4, 5)

print '# import xmath as math'
import xmath as math # 為 xmath 模組取別名為 math
print math.e

print '# from xmath import min'
from xmath import min  # 將 min 複製至目前模組，不建議 from modu import *，易造
print min(10, 5)
```

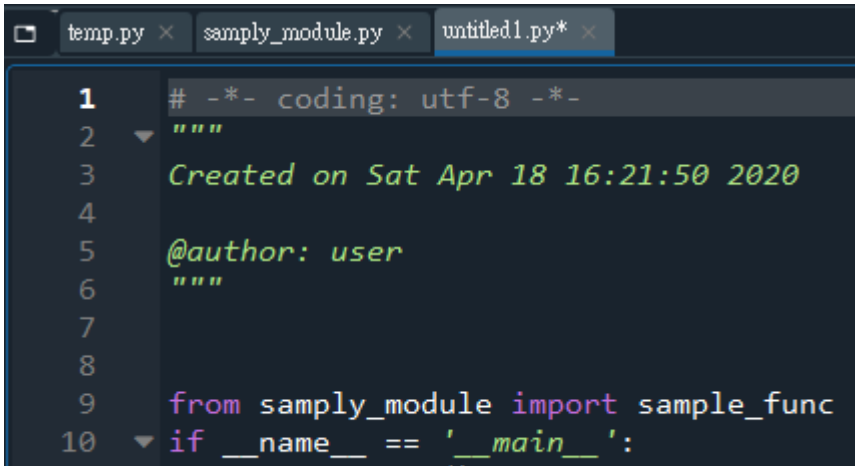
The *from...import* Statement

- Python's *from* statement lets you import specific attributes from a module into the current namespace.
- The *from...import* has the following syntax:

```
from modname import name1[, name2[, ... nameN]]
```

- For example, to import the function fibonacci from the module fib, use the following statement:

```
from fib import fibonacci
```



The *from...import* * Statement:

- It is also possible to import all names from a module into the current namespace by using the following import statement:

```
from modname import *
```

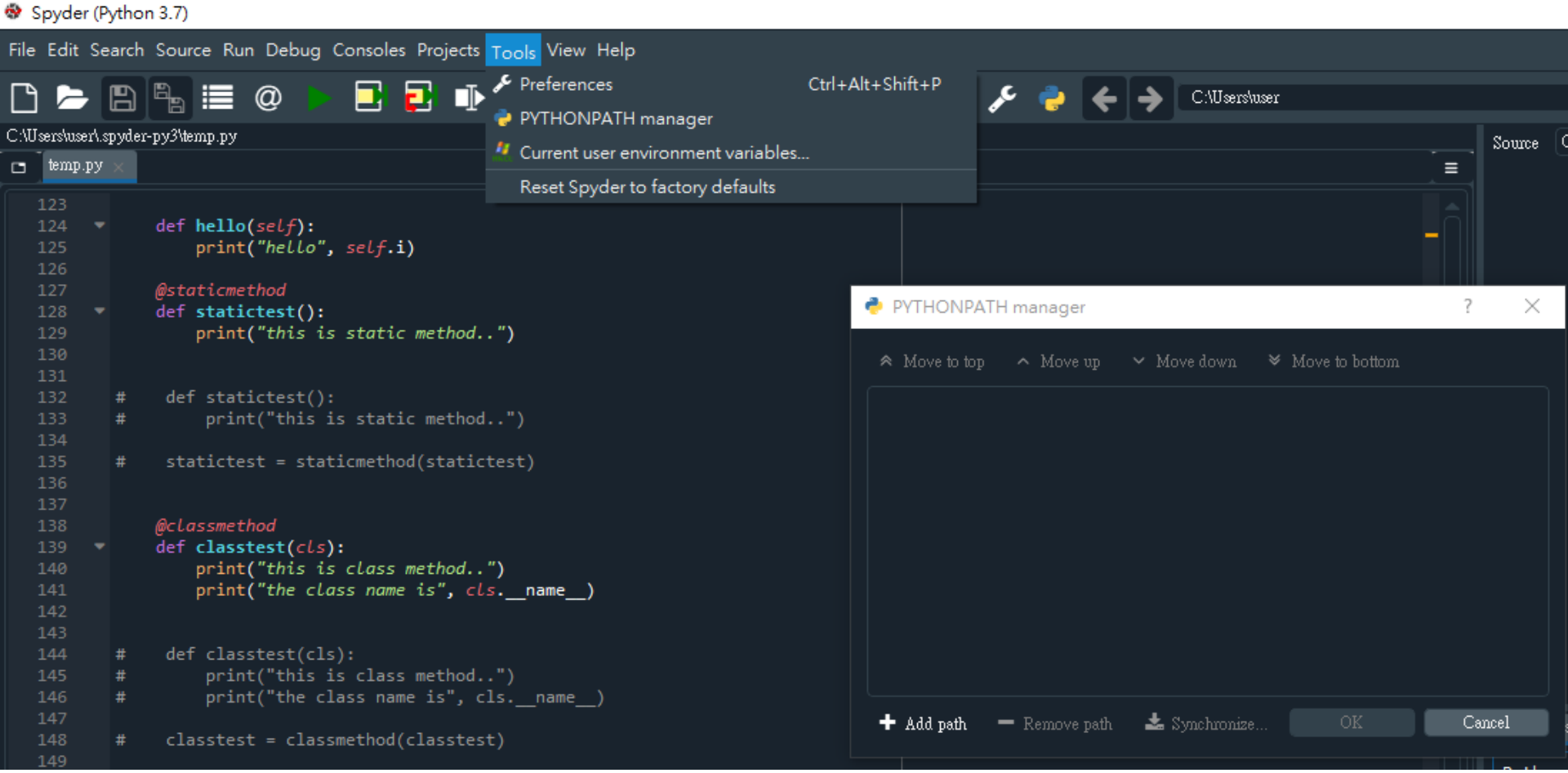
Locating Modules:

- When you import a module, the Python interpreter searches for the module in the following sequences:
  - The current directory.
  - If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.
  - If all else fails, Python checks the default path.
    - On UNIX, this default path is normally /usr/local/lib/python/.

1./usr/lib/python2.7 on Ubuntu
1./usr/lib/python2.7 on Ubuntu

The *PYTHONPATH* Variable:

- The PYTHONPATH is an environment variable, consisting of a list of directories.
- The syntax of PYTHONPATH is the same as that of the shell variable PATH.
- Here is a typical PYTHONPATH from a Windows system:
  - set PYTHONPATH=c:\python27\lib;
- Here is a typical PYTHONPATH from a UNIX system:
  - set PYTHONPATH=/usr/local/lib/python
- Ubuntu
  - /usr/lib/python2.7


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Preferences Ctrl+Alt+Shift+P
PYTHONPATH manager
Current user environment variables...
Reset Spyder to factory defaults
C:\Users\user
C:\Users\user\.spyder-py3\temp.py
temp.py
Source
def hello(self):
print("hello", self.i)
@staticmethod
def statictest():
print("this is static method..")
# def statictest():
# print("this is static method..")
# statictest = staticmethod(statictest)
@classmethod
def classtest(cls):
print("this is class method..")
print("the class name is", cls.__name__)
# def classtest(cls):
# print("this is class method..")
# print("the class name is", cls.__name__)
# classtest = classmethod(classtest)
PYTHONPATH manager
Move to top
Move up
Move down
Move to bottom
Add path
Remove path
Synchronize...
OK
Cancel

Namespaces and Scoping

- Variables are names (identifiers) that map to objects.
- **A *namespace* is a dictionary of variable names (keys) and their corresponding objects (values)**.
- A Python statement can access variables in a *local namespace* and in the *global namespace*.
  - If a local and a global variable have the same name, the local variable ***shadows*** the global variable.
- Each function has its own local namespace.
  - Class methods follow the same scoping rule as ordinary functions.
- Python assumes that any variables assigns a value in a function is local.

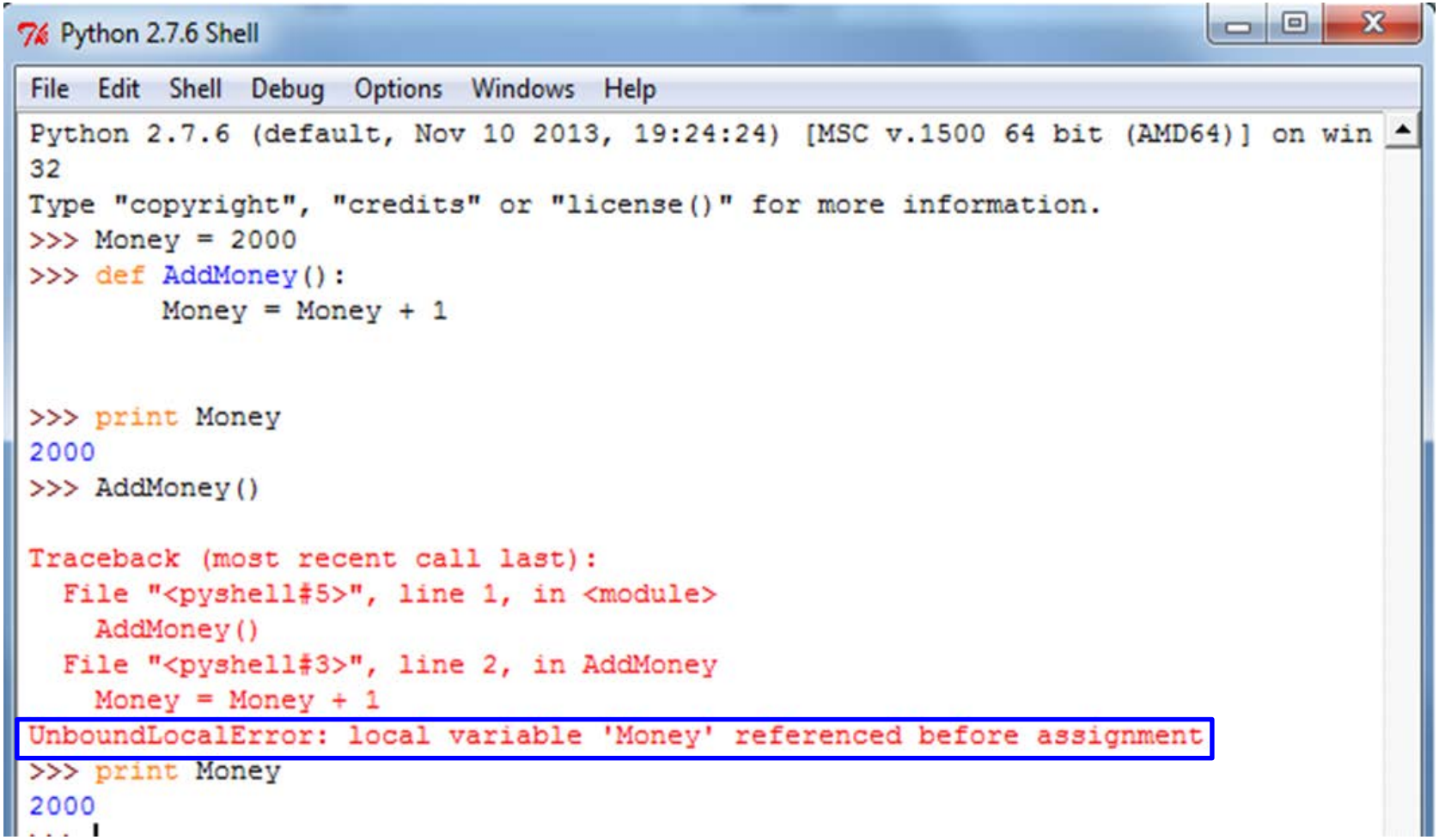
Namespaces and Scoping

- Therefore, in order to assign a value to a global variable within a function, you must first use the *global* statement.
- The statement *global VarName* tells Python that VarName is a global variable.
  - Python stops searching the local namespace for the variable.

```
#!/usr/bin/python

Money = 2000
def AddMoney():
    # Uncomment the following line to fix the code:
    # global Money
    Money = Money + 1

print Money
AddMoney()
print Money
```

```
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    AddMoney()
  File "<pyshell#3>", line 2, in AddMoney
    Money = Money + 1
UnboundLocalError: local variable 'Money' referenced before assignment
```

Results


Python 2.7.6 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.6 (default, Nov 10 2013, 19:24:24) [MSC v.1500 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> Money = 2000
>>> def AddMoney():
Money = Money + 1
>>> print Money
2000
>>> AddMoney()
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
AddMoney()
File "<pyshell#3>", line 2, in AddMoney
Money = Money + 1
UnboundLocalError: local variable 'Money' referenced before assignment
>>> print Money
2000

Example (1)

```
# sample.py
myGlobal = 5

def func1():
    myGlobal = 42

def func2():
    print myGlobal

func1()
func2()
```

5

```
def func1():
    global myGlobal
    myGlobal = 42
```

The global statement is a declaration which holds for the entire current code block.

42

Example (2)

```
x = 10
def outer():
    x = 100          # 這是在 outer() 函式範圍的 x
    def inner():
        nonlocal x
        x = 1000     # 改變的是 outer() 函式的 x
    inner()
    print(x)         # 顯示 1000

outer()
print(x)             # 顯示 10
```

The nonlocal statement causes the listed identifiers to refer to previously bound variables in the nearest enclosing scope excluding globals. (Python 3)

Example (3)

1

```
a = 0

def function1():
    a = 1
    def function2():
        a = 2
        print("function2: ", a)
    function2()
    print("function1: ", a)

function1()
print("global: ", a)
```

```
function2:  2
function1:  1
global:  0
```

2

```
a = 0

def function3():
    a = 1
    def function4():
        nonlocal a
        a = 2
        print("function4: ", a)
    function4()
    print("function3: ", a)

function3()
print("global: ", a)
```

```
function4:  2
function3:  2
global:  0
```

3

```
a = 0

def function5():
    a = 1
    def function6():
        global a
        a = 2
        print("function6: ", a)
    function6()
    print("function5: ", a)

function5()
print("global: ", a)
```

```
function6:  2
function5:  1
global:  2
```

globals() locals(), and var()

- The *globals() locals()* and *var()* functions can be used to return the names in the global and local namespaces depending on the location from where they are called.
- If locals() is called from within a function, it will return all the names that can be accessed **locally** from that function.
- If globals() is called from within a function, it will return all the names that can be accessed **globally** from that function.
- var() returns *either a* dictionary of the current namespace (if called with no argument) or *the* dictionary of the argument.
- The return type of both these functions (i.e., locals and globals)is dictionary.
  - Therefore, names can be extracted using the keys() function.

Example

```
class A():
    def __init__(self, id):
        self.id = id
        print("Class A locals:\t%s" % locals())
        print("Class A vars:\t%s" % vars())


def B():
    id = 1
    print("Function B locals:\t%s" % locals())
    print("Function B vars:\t%s" % vars())


if __name__ == '__main__':
    a = A(1)

    B()
    print("Module globals:\t%s\n" % globals())
    print("Module locals:\t%s\n" % locals())
    print("Module vars:\t%s\n" % vars())
```

```
Class A locals: {'self': <__main__.A object at 0x000002063392C3C8>, 'id': 1}
Class A vars:   {'self': <__main__.A object at 0x000002063392C3C8>, 'id': 1}
Function B locals:      {'id': 1}
Function B vars:        {'id': 1}
Module globals: {'__name__': '__main__', '__file__': 'C:\\Users\\user\\untitled0.py', '__nonzero__': <function
InteractiveShell.new_main_mod.<locals>.<lambda> at 0x00000206334EE798>, '__builtins__': {'__name__': 'builtins',
'__doc__': "Built-in functions, exceptions, and other objects.\n\nNoteworthy: None is the `nil' object; Ellipsis
represents `...' in slices.", '__package__': '', '__loader__': <class '_frozen_importlib.BuiltinImporter'>, '__sp
ModuleSpec(name='builtins', loader=<class '_frozen_importlib.BuiltinImporter'>), '__build_class__': <built-in fun
__build_class__>, '__import__': <built-in function __import__>, 'abs': <built-in function abs>, 'all': <built-in
function all>, 'any': <built-in function any>, 'ascii': <built-in function ascii>, 'bin': <built-in function bin>
'breakpoint': <built-in function breakpoint>, 'callable': <built-in function callable>, 'chr': <built-in function
'compile': <built-in function compile>, 'delattr': <built-in function delattr>, 'dir': <built-in function dir>,
'divmod': <built-in function divmod>, 'eval': <built-in function eval>, 'exec': <built-in function exec>, 'format
<built-in function format>, 'getattr': <built-in function getattr>, 'globals': <built-in function globals>, 'hasa
<built-in function hasattr>, 'hash': <built-in function hash>, 'hex': <built-in function hex>, 'id': <built-in fu
```

The dir( ) Function

- The dir() built-in function returns a sorted list of strings containing the names defined by a module.
- The list contains the names of all the modules, variables and functions that are defined in a module.
- Here, the special string variable *__name__* is the **module's name**, and *__file__* is the **filename** from which the module was loaded.

```
In [25]: import math

In [26]: content =dir(math)

In [27]: print(content)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'fl
'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma
'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'tau', 'trunc']
```

__name__ == '__main__'

samply_module.py untitled1.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 18 16:19:33 2020

@author: user
"""


def sample_func():
    print('Hello!')
    print("__name__", __name__)


sample_func()
```

```
Hello!
__name__ __main__
```

samply_module.py untitled1.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 18 16:21:50 2020

@author: user
"""


from samply_module import sample_func
if __name__ == '__main__':
    sample_func()
```

```
Hello!
__name__ samply_module
```

Packages in Python

- A package is a hierarchical file directory structure
  - It consists of modules and subpackages and sub-subpackages, and so on.
- Consider a file *Pots.py* available in *Phone* directory
- We have another two files having different functions with the same directory as above:
  - *Phone/Isdn.py* file having function Isdn()
  - *Phone/G3.py* file having function G3()
- Now, create one more file __init__.py in *Phone* directory:
  - Phone/__init__.py
- *Phone* directory includes *Pots.py, Isdn.py, G3.py, and __init__.py.*

Packages in Python

- To make all of your functions available when you've imported Phone, you need to put explicit import statements in __init__.py as follows:
  - from Pots import Pots
  - from Isdn import Isdn
  - from G3 import G3

```
#!/usr/bin/python

# Now import your Phone Package.
import Phone

Phone.Pots()
Phone.Isdn()
Phone.G3()
```

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

Example

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

```
import sound.effects.echo
```

```
sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
```

```
from sound.effects import echo
```

```
echo.echofilter(input, output, delay=0.7, atten=4)
```

```
from sound.effects.echo import echofilter
```

```
echofilter(input, output, delay=0.7, atten=4)
```

Python Image Library - Examples

Python Imaging Library

```
from PIL import Image
global ext
ext = ".jpg"
imageFile = "test.jpg"
im1 = Image.open(imageFile)
im1.show()
```

Original image

- **Python Imaging Library (PIL)**
- http://www.pythonware.com/products/pil/
- **PIL 1.1.7**
- http://effbot.org/downloads/PIL-1.1.7.win32-py2.7.exe

- BMP
- EPS
- GIF
- JPEG
- PNG
- TIFF
- PDF

conda list

```
In [2]: conda list
# packages in environment at C:\Users\user\anaconda3:
#
# Name                    Version                   Build  Channel
_anaconda_depends         2020.02                  py37_0

Note: you may need to restart the kernel to use updated packages.
_ipyw_jlab_nb_ext_conf    0.1.0                    py37_0
alabaster                 0.7.12                   py37_0
anaconda                  custom                   py37_1
anaconda-client           1.7.2                    py37_0
anaconda-navigator        1.9.12                   py37_0
anaconda-project          0.8.4                      py_0
argh                      0.26.2                   py37_0
asn1crypto                1.3.0                    py37_0
astroid                   2.3.3                    py37_0
astropy                   4.0.1.post1      py37he774522_0
atomicwrites              1.3.0                    py37_1
attrs                     19.3.0                     py_0
autopep8                  1.4.4                      py_0
babel                     2.8.0                      py_0
backcall                  0.1.0                    py37_0
backports                 1.0                        py_2
```

conda **install** package_name

```
ata.json): ...working... done
with initial frozen solve. Retrying with flexible solve.
): ...working... done
with initial frozen solve. Retrying with flexible solve.

o use updated packages.
PackagesNotFoundError: The following packages are not available from current channels:

  - image

Current channels:

  - https://repo.anaconda.com/pkgs/main/win-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/win-64
  - https://repo.anaconda.com/pkgs/r/noarch
  - https://repo.anaconda.com/pkgs/msys2/win-64
  - https://repo.anaconda.com/pkgs/msys2/noarch

To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org

and use the search bar at the top of the page.
```

conda upgrade --all

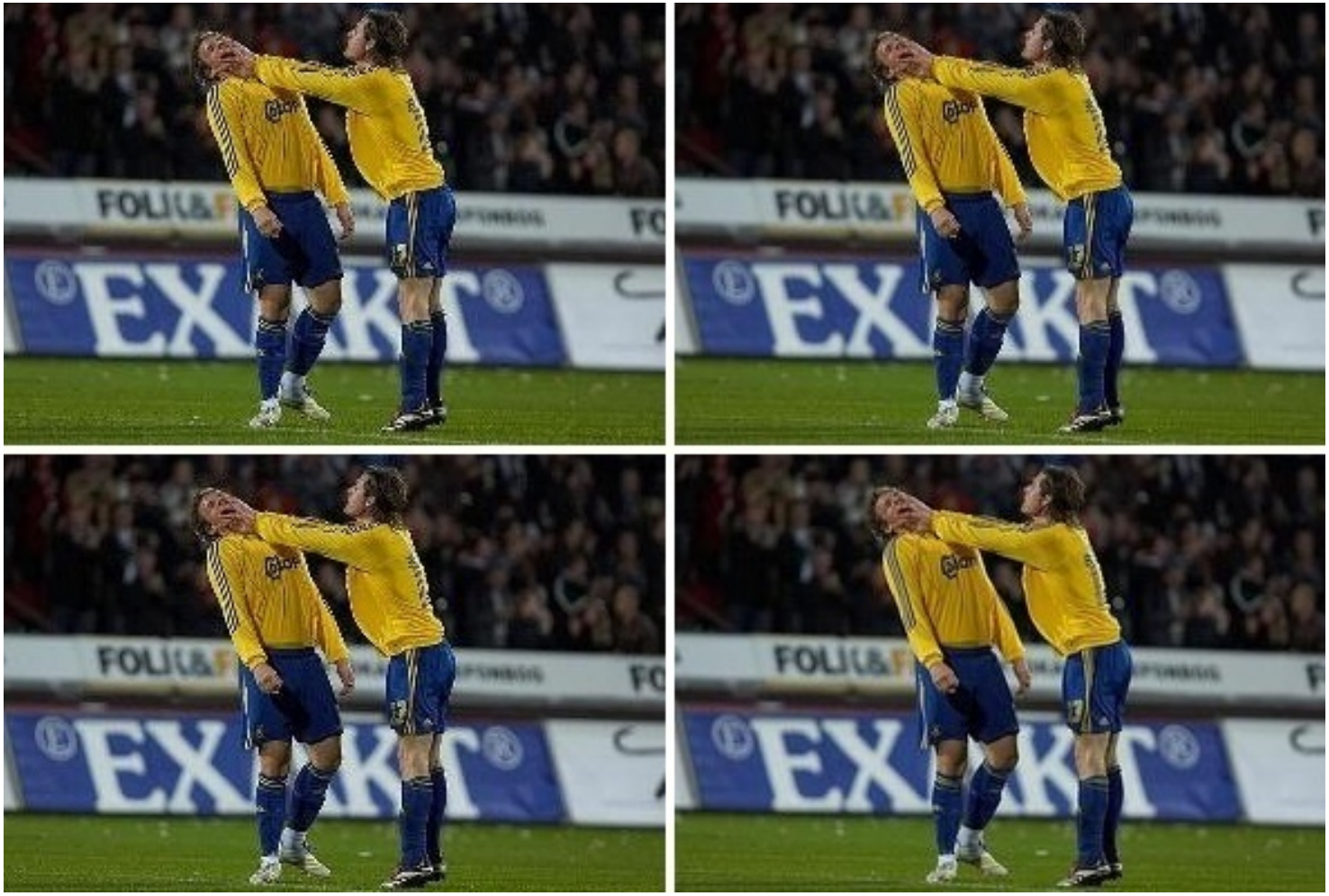
```
In [1]: conda upgrade --all
Collecting package metadata (current_repodata.json): ...working... done
Note: you may need to restart the kernel to use updated packages.
Solving environment: ...working... done

## Package Plan ##

  environment location: C:\Users\user\anaconda3


The following packages will be downloaded:


    package                    |            build
    ---------------------------|-----------------
    dask-2.15.0                |             py_0          14 KB
    dask-core-2.15.0           |             py_0         575 KB
    distributed-2.15.0         |          py37_0         997 KB
    ------------------------------------------------------------
                                           Total:         1.5 MB

The following packages will be UPDATED:

  dask                                       2.14.0-py_0 --> 2.15.0-py_0
  dask-core                                  2.14.0-py_0 --> 2.15.0-py_0
  distributed                             2.14.0-py37_0 --> 2.15.0-py37_0



Downloading and Extracting Packages

dask-2.15.0           | 14 KB     |            |   0%
dask-2.15.0           | 14 KB     | ########## | 100%

distributed-2.15.0    | 997 KB    |            |   0%
distributed-2.15.0    | 997 KB    | ########8  |  79%
distributed-2.15.0    | 997 KB    | ########## | 100%

dask-core-2.15.0      | 575 KB    |            |   0%
dask-core-2.15.0      | 575 KB    | ########## | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
```

Resize

```python
from PIL import Image
global ext
ext = ".jpg"
imageFile = "test.jpg"
im1 = Image.open(imageFile)



def imgResize(im):
    div = 2
    width = int(im.size[0] / div)
    height = int(im.size[1] / div)
    im2 = im.resize((width, height), Image.NEAREST) # use nearest neighbour
    im3 = im.resize((width, height), Image.BILINEAR) # linear interpolation in a 2x2 environment
    im4 = im.resize((width, height), Image.BICUBIC) # cubic spline interpolation in a 4x4 environme
    im5 = im.resize((width, height), Image.ANTIALIAS) # best down-sizing filter

    im2.save("NEAREST" + ext)
    im3.save("BILINEAR" + ext)
    im4.save("BICUBIC" + ext)
    im5.save("ANTIALIAS" + ext)
imgResize(im1)
```

NEAREST：最近濾波。從輸入影象中選取最近的畫素作為輸出畫素。它忽略了所有其他的畫素。
BILINEAR：雙線性濾波。在輸入影象的2**x2**矩陣上進行線性插值。
BICUBIC：雙立方濾波。在輸入影象的4**x4**矩陣上進行立方插值。
ANTIALIAS：平滑濾波。這是**PIL** 1.1.3版本中新的濾波器。對所有可以影響輸出畫素的輸入畫素進行高質量的重取樣濾波，以計算輸出畫素

Resize



Crop

```
def imgCrop(im):
    box = (50, 50, 200, 300)
    region = im.crop(box)
    region.save("CROPPED" + ext)

imgCrop(im1)
```

Transpose

```
def imgTranspose(im):
  box = (50, 50, 200, 300)
  region = im.crop(box)
  region =region.transpose(Image.ROTATE_180)
  im.paste(region, box)
  im.save("TRANSPOSE"+ext)

imgTranspose(im1)
```

- Image.FLIP_LEFT_RIGHT (左右翻轉)
- Image.FLIP_TOP_DOWN (上下翻轉)
- Image.ROTATE_90 (旋轉90度)
- Image.ROTATE_180 (旋轉180度)
- Image.ROTATE_270 (旋轉270度)

Band Merge

```
def bandMerge(im):
    r, g, b = im.split()
    im = Image.merge("RGB", (g,g,g))
    im.save("MERGE" + ext)

bandMerge(im1)
```

```
from PIL import Image
if __name__ == '__main__':
    im = Image.open('test.jpg')
    r,g,b = im.split()
    b.show()
    imx = Image.merge("RGB", (g, b, r))
    imx.show()
```

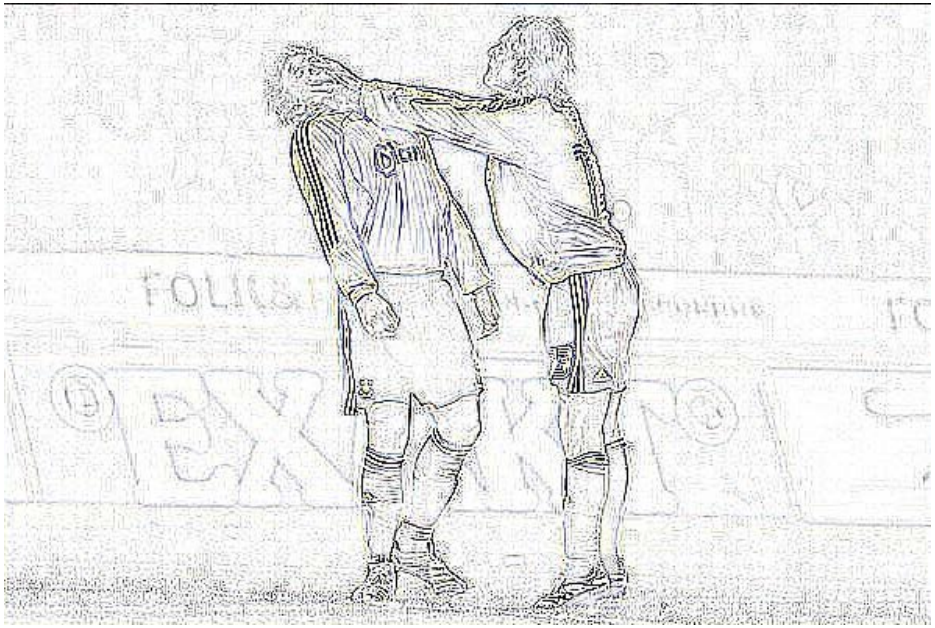
Blur

```
from PIL import ImageFilter
def filterBlur(im):
    im1 = im.filter(ImageFilter.BLUR)
    im1.save("BLUR" + ext)

filterBlur(im1)
```

加入濾鏡

Find contours

```
from PIL import ImageFilter
def filterContour(im):
    im1 = im.filter(ImageFilter.CONTOUR)
    im1.save("CONTOUR" + ext)

filterContour(im1)
```



Find edges

```
from PIL import ImageFilter
def filterFindEdges(im):
    im1 = im.filter(ImageFilter.FIND_EDGES)
    im1.save("EDGES" + ext)

filterFindEdges(im1)
```

ImageFilter.BLUR ：模糊濾鏡
ImageFilter.CONTOUR ：只顯示輪廓
ImageFilter.EDGE_ENHANCE ：邊界加強
ImageFilter.EDGE_ENHANCE_MORE ：邊界加強(閥值更大)
ImageFilter.EMBOSS ：浮雕濾鏡
ImageFilter.FIND_EDGES ：邊界濾鏡
ImageFilter.SMOOTH ：平滑濾鏡
ImageFilter.SMOOTH_MORE ：平滑濾鏡(閥值更大)
ImageFilter.SHARPEN ：銳化濾鏡

```python
from PIL import Image,ImageDraw,ImageFont
if __name__ == '__main__':
    im = Image.open('test.jpg')
    dr_im = ImageDraw.Draw(im)
    w,h = im.size
    myFont = ImageFont.truetype('timesbd.ttf',80)
    dr_im.text([0.1 * w,0.8 * h], u"Python", fill = (255,0,0), font=myFon
    im.show()
```

Python

```python
from PIL import Image
def deffun(c):
    return c*2

if __name__ == '__main__':
    im = Image.open("test.jpg")
    im = Image.eval(im,deffun)
    im.show()
```