Numerical Python

Modules

Announcements

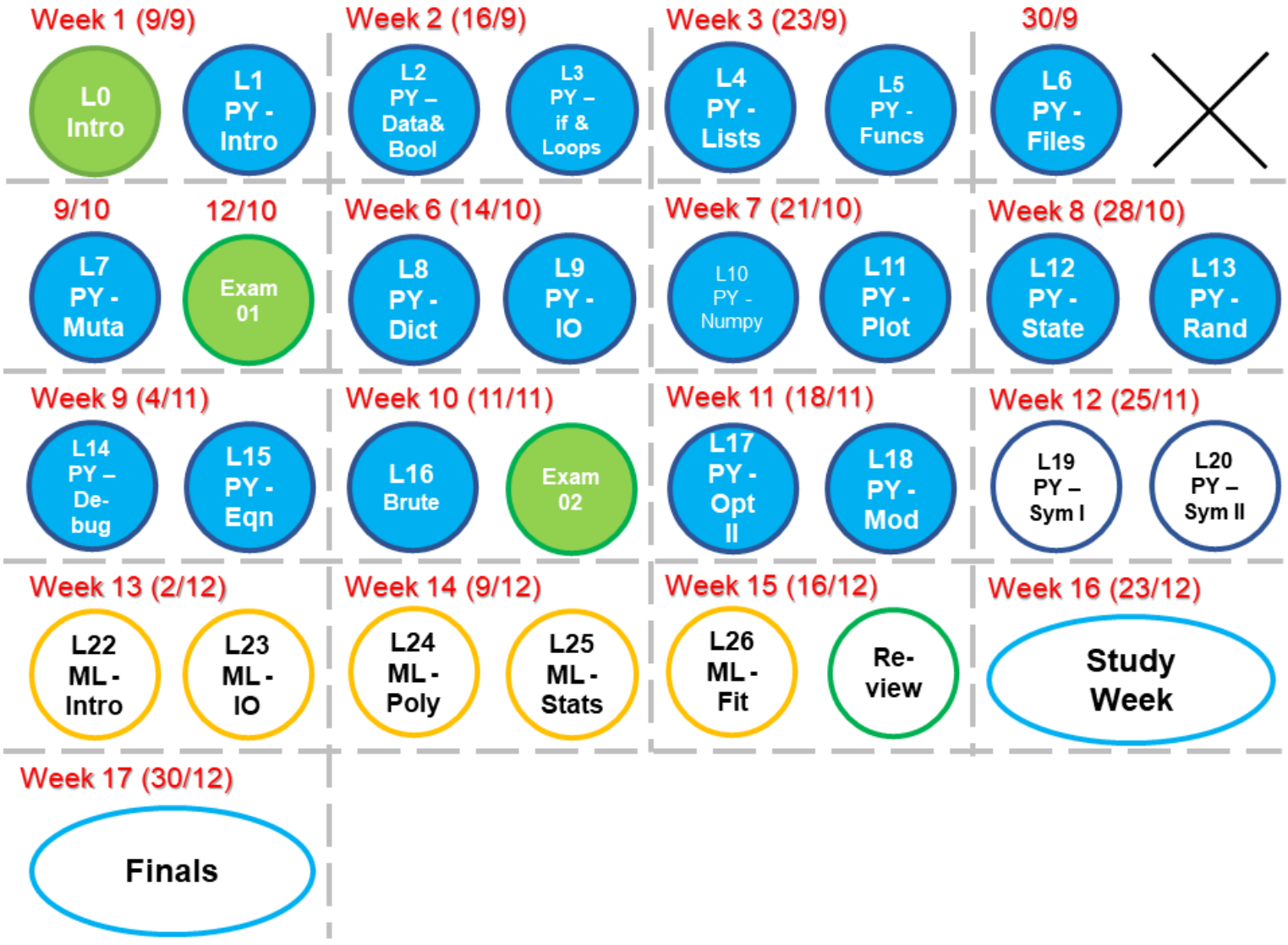
quiz: `quiz18` due on Thurs 21/11

lab: `lab` on Fri 22/11

hw: `hw10` due 27/11

exam: ??? When?? When??

Roadmap


Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Muta
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY - Numpy
L11 PY - Plot
Week 8 (28/10)
L12 PY - State
L13 PY - Rand
Week 9 (4/11)
L14 PY – De-bug
L15 PY - Eqn
Week 10 (11/11)
L16 Brute
Exam 02
Week 11 (18/11)
L17 PY - Opt II
L18 PY - Mod
Week 12 (25/11)
L19 PY – Sym I
L20 PY – Sym II
Week 13 (2/12)
L22 ML - Intro
L23 ML - IO
Week 14 (9/12)
L24 ML - Poly
L25 ML - Stats
Week 15 (16/12)
L26 ML - Fit
Re-view
Week 16 (23/12)
Study Week
Week 17 (30/12)
Finals

Objectives

A. Install and utilize new packages
B. Import functions from a user-defined module
C. Create a user-defined module using conventions like `__name__`
D. See a partial list of further topics to explore

Optimization Redux

Figure of Merit

A Equation or function that generates a number for comparsion

Modules

Modules

Many times you need to use a module (package) which has not yet been installed on the machine you are using (inside Python):

```
>>> import seaborn
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent cal
<ipython-input-1-acfe433e7feb> in <module>()
----> 1 import seaborn

ModuleNotFoundError: No module named 'seaborn'
```

Modules - check

Check what packages you have in your python (outside Python in prompt)

```
pip list
```

If your python package is from Anaconda, you can also use (outside Python in prompt):

```
conda list
```

Modules - install

Typically, new packages can be installed using `pip`.
`pip` is invoked *on the prompt/command line/Anaconda prompt*, not within a Python session.
Assume you want to install a package call ”seaborn”,(outside Python)

```
pip install seaborn
```

If your python package is from Anaconda, it is preferred to use (outside Python):

```
conda install seaborn
```

Modules - upgrade

Sometimes packages need to be updated because new features are available:
(outside Python)

```
pip install --upgrade XXX
```

if your python package is from Anaconda, it is preferred to use (outside Python):

```
conda upgrade XXX
```

`XXX` - name of package
Anaconda will check if there is a newer version to upgrade. Then choose y or n to proceed with the upgrading ifa newer version is found

Modules - delete

For some reasons, you decide to delete some packages (outside Python):

```
pip uninstall jupyter
```

if your python package is from Anaconda, it is preferred to use (outside Python):

```
conda remove XXXX
```

where XXXX is the package name

Roll Your Own Module

Why you want to create your own

Why you want to create your own

Reuse old code many times without copy and paste many times!!!

Why you want to create your own

Reuse old code many times without copy and paste many times!!!

Convenience and portability

Why you want to create your own

Reuse old code many times without copy and paste many times!!!

Convenience and portability

Show off!!!

Creating a Module

Our example: compound interest using the Actual/360 convention:

$$A = A0\left(1 + \frac{p}{360 \cdot 100}\right)^{n}$$

A = Future value
A0 = Present value
p = Annual interest rate
n = number of days

We want this interest.py program to be able to calculate these variables.

Creating a Module

We also want to use the interest.py as a module:

```
import interest

A0 = 1
p  = 5
n  = 720

A  = interest.future_value( A0,p,n )

print( '$%.2f has compounded to $%.2f
                        after 2 years'%( A0,A ) )
```

Creating a Module

Easiest way is to use a normal editor (like spyder or others) to create the `interest.py`.
Recommended only functions inside a Module.

```
def future_value( A0, p, n ):
    '''
    Calculate money after compounding A0 money
    for n days at p percent interest using
    the Actual/360 convention.
    '''
    return A0 * ( 1 + p / ( 360.0 * 100 ) ) ** n

def present_value( A, p, n ):
    ....

def days( A, A0, p ):
    ....
```

Creating a Module

Optional:
Other than the functions themselves, good to include test functions to test your functions in `interest.py` :

```
def test_present_value():
    A0,p,n = 2.0,5,730
    A_expected = 2.213398
    A_computed = future_value( A0,p,n )
    import numpy as np
    return np.isclose( A_expected,A_computed )
```

Start these functions name with

```
test_
```

if you want to use the function `pytest`

Testing a Module

Optional:

At the command line (outside Python in the same directory as interest.py):

```
pytest interest.py #pytest may not be installed
```

Install if needed.

For Jupyter: we will need to install a customized code but it has limited functionality.

See *https://github.com/akaihola/ipython_pytest*

Using a Module, method 1

So now, you can do `import interest`

Using a Module, method 2

Whenever the Python interpreter reads a source file, it does two things:

1. it sets a few special variables like __name__, and then
2. it executes all of the code found in the file.

What if you want to run this file as a program? (outside python)
» python interest.py

Using a Module, method 2

Add to interest.py

```
if __name__ == '__main__':
    test_future_value()
    test_present_value()interest
    ...
    print( "okay" )
```

__name__ is equal to the module name when imported from another program. Here *__name__* = '__interest__'
__name__ equals the string '__main__' if the module file is run as a program.

Locating this Module

Unless the module file resides in the same folder (Easiest), we need to tell Python where to find our module.

Python looks for modules in the folders contained in the list sys.path. Use:

```
(inside Python)
import sys, pprint
pprint.pprint(sys.path)
```

to locate which folders Python look into for modules.

Place the module file in one of the folders in sys.path (Second Easiest)

More Python

What else?

A. Next 2 lectures: SymPy—symbolic algebra

B. Pandas—Python for Data Analysis
C. Scikit-Learn—machine learning
D. Classes (object-oriented programming)
E. Bokeh—interactive plots, like web graphics