Introduction to Python
Part 2

v0.3

Brian Gregor
Research Computing Services
Information Services & Technology

BOSTON UNIVERSITY

Tutorial Outline – Part 2

- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

Tuples

- Tuples are lists whose elements can't be changed.
  - Like strings they are immutable
- Indexing (including slice notation) is the same as with lists.

```
# a tuple
a = 10,20,30
# a tuple with optional parentheses
b = (10,20,30)
# a list
c = [10,20,30]
# ...turned into a tuple
d = tuple(c)

# and a tuple turned into a list
e = list(d)
```

Return multiple values from a function

- Tuples are more useful than they might seem at first glance.
- They can be easily used to return multiple values from a function.
- Python syntax can automatically unpack a tuple return value.

```
def min_max(x):
    ''' Return the maximum and minimum
        values of x '''
    minval = min(x)
    maxval = max(x)
    # a tuple return...
    return minval,maxval


a = [10,4,-2,32.1,11]

val = min_max(a)
min_a = val[0]
max_a = val[1]

# Or, easier...
min_a, max_a = min_max(a)
```

Dictionaries

- Dictionaries are another basic Python data type that are tremendously useful.

- Create a dictionary with a pair of curly braces:

```
x = {}
```

- Dictionaries store *values* and are indexed with *keys*

- Create a dictionary with some initial values:

```
x = {'a_key':55, 100:'a_value', 4.1:[5,6,7]}
```

BOSTON UNIVERSITY

Dictionaries

- Values can be any Python thing
- Keys can be primitive types (numbers), strings, tuples, and some custom data types
  - Basically, any data type that is **immutable**
- Lists and dictionaries cannot be keys but they can stored as values.
- Index dictionaries via keys:

```
x['a_key'] → 55
x[100] → 'a_value'
```

Try Out Dictionaries

- Create a dictionary in the Python console or Spyder editor.

```
x = {}
x[3] = -3.3
x[10.2] = []

print(x)
```

- Add some values to it just by using a new key as an index. Can you overwrite a value?

- Try *x.keys()* and *x.values()*

- Try: `del x[valid_key]` → deletes a key/value pair from the dictionary.

Tutorial Outline – Part 2

- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

Modules

- Python modules, aka libraries or packages, add functionality to the core Python language.
- The Python Standard Library provides a very wide assortment of functions and data structures.
  - Check out their Brief Tour for a quick intro.
- Distributions like Anaconda provides dozens or hundreds more
- You can write your own libraries or install your own.

PyPI

- The Python Package Index is a central repository for Python software.
  - Mostly but not always written in Python.
- A tool, *pip*, can be used to install packages from it into your Python setup.
  - Anaconda provides a similar tool called *conda*
- Number of projects (as of May 2018): **140,310**
- You should always do your due diligence when using software from a place like PyPI. Make sure it does what you think it's doing!

Python Modules on the SCC

- Python modules should not be confused with the SCC *module* command.
- For the SCC there are instructions on how to install Python software for your account or project.
- Many SCC modules provide Python packages as well.
  - Example: tensorflow, pyopencl, others.
- Need help on the SCC? Send us an email: help@scv.bu.edu

BOSTON UNIVERSITY

Importing modules

- The *import* command is used to load a module.
- The name of the module is prepended to function names and data structures in the module.
  - The preserves the module *namespace*
- This allows different modules to have the same function names – when loaded the module name keeps them separate.

```
import math

z=math.sin(0.1)

print(z)

dir(math)

help(math.ceil)
```

Try these out!

BOSTON UNIVERSITY

Fun with *import*

- The *import* command can strip away the module name:

```
from math import *
```

- Or it can import just a single function:

```
from math import cos
```

- Or rename on the import:

```
from math import sin as exact_sin
```

Fun with *import*

- The *import* command can also load your own Python files.
- The Python file to the right can be used in another Python script:

```
# Don't use the .py ending
import myfuncs
x = [1,2,3,4]
y = myfuncs.get_odds(x)
```

myfuncs.py

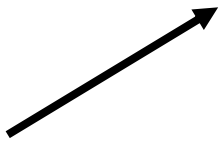
```
def get_odds(lst):
    ''' Gets the odd numbers in a list.

        lst: incoming list of integers
        return: list of odd integers '''
    odds = []
    for elem in lst:
        # Odd if there's a remainder when
        # dividing by 2.
        if elem % 2 != 0:
            odds.append(elem)
    return odds
```

Import details

- Python reads and executes a file when the file
  - is opened directly: `python somefile.py`
  - is imported: `import somefile`
- Lines that create variables, call functions, etc. are all executed.
- Here these lines will run when it's imported into another script!

myfuncs.py

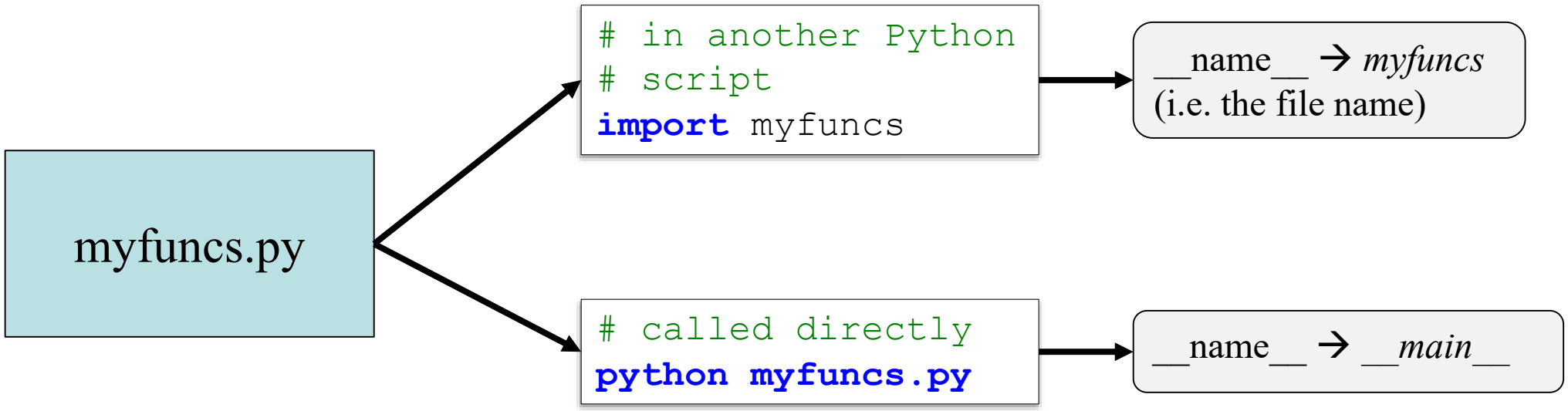
```
def get_odds(lst):
    ''' Gets the odd numbers in a list.

        lst: incoming list of integers
        return: list of odd integers '''
    odds = []
    for elem in lst:
        # Odd if there's a remainder when
        # dividing by 2.
        if elem % 2 != 0:
            odds.append(elem)
    return odds

x = [1,2,3,4]
y = get_odds(x)
print(y)
```



BOSTON UNIVERSITY

The __name__ attribute

- Python stores object information in hidden fields called *attributes*
- Every file has one called __name__ whose value depends on how the file is used.



The __name__ attribute

- __name__ can be used to make a Python scripts usable as a standalone program **and** as imported code.

- Now:
  - `python myfuncs.py` → __name__ has the value of '__main__' and the code in the *if* statement is executed.
  - `import myfuncs` → __name__ is 'myfuncs' and the *if* statement does not run.

myfuncs.py

```
def get_odds(lst):
    ''' Gets the odd numbers in a list.

        lst: incoming list of integers
        return: list of odd integers '''
    odds = []
    for elem in lst:
        # Odd if there's a remainder when
        # dividing by 2.
        if elem % 2 != 0:
            odds.append(elem)
    return odds


if __name__=='__main__':
    x = [1,2,3,4]
    y = get_odds(x)
    print(y)
```

BOSTON UNIVERSITY

Tutorial Outline – Part 2

- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

A brief into to numpy and matplotlib

- numpy is a Python library that provides efficient multidimensional matrix and basic linear algrebra
  - The syntax is very similar to Matlab or Fortran
- matplotlib is a popular plotting library
  - Remarkably similar to Matlab plotting commands!
- A third library, scipy, provides a wide variety of numerical algorithms:
  - Integrations, curve fitting, machine learning, optimization, root finding, etc.
  - Built on top of numpy
- Investing the time in learning these three libraries is worth the effort!!

numpy

- numpy provides data structures written in compiled C code
- Many of its operations are executed in compiled C or Fortran code, not Python.

- Check out *numpy_basics.py*

numpy datatypes

- Unlike Python lists, which are generic containers, numpy arrays are typed.
- If you don’t specify a type, numpy will assign one automatically.
- A wide variety of numerical types are available.
- Proper assignment of data types can sometimes have a significant effect on memory usage and performance.

```
import numpy as np
x = np.array([1, 2])
 # Prints "int64"
print(x.dtype)

x = np.array([1.0, 2.0])
# Prints "float64"
print(x.dtype)

x = np.array([1, 2], dtype=np.uint8)
 # Prints "uint8"
print(x.dtype)
```

Numpy operators

- Numpy arrays will do element-wise arithmetic: + / - * **
- Matrix (or vector/matrix, etc.) multiplication needs the .dot() function.
- Numpy has its own sin(), cos(), log(), etc. functions that will operate element-by-element on its arrays.

```
import numpy as np
x = np.array([1, 2])

x = x + 1
print(x)

y=x / 2.5

print(y.dtype)
print(y)

print(y * x)
print('Dot product: %s' % y.dot(x))
```

Try these out!

indexing

- Numpy arrays are indexed much like Python lists
- Slicing and indexing get a little more complicated when using numpy arrays.
- Open *numpy_indexing.py*

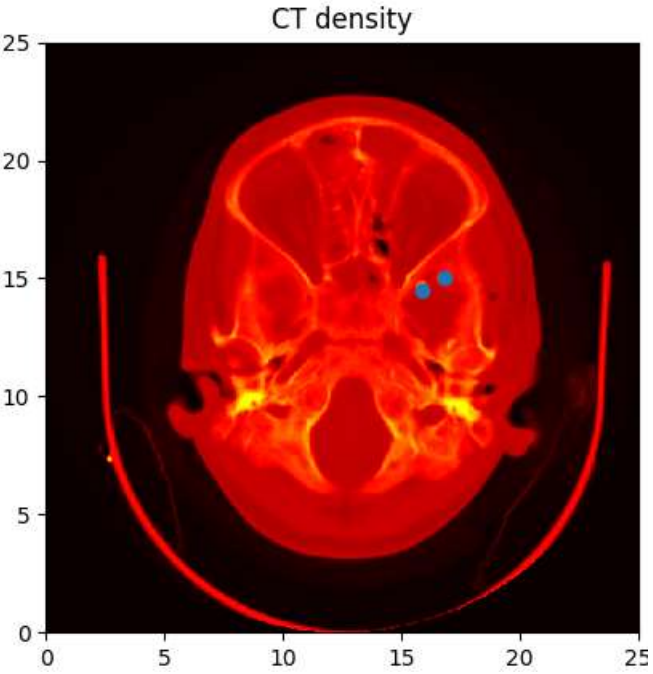
Plotting with matplotlib

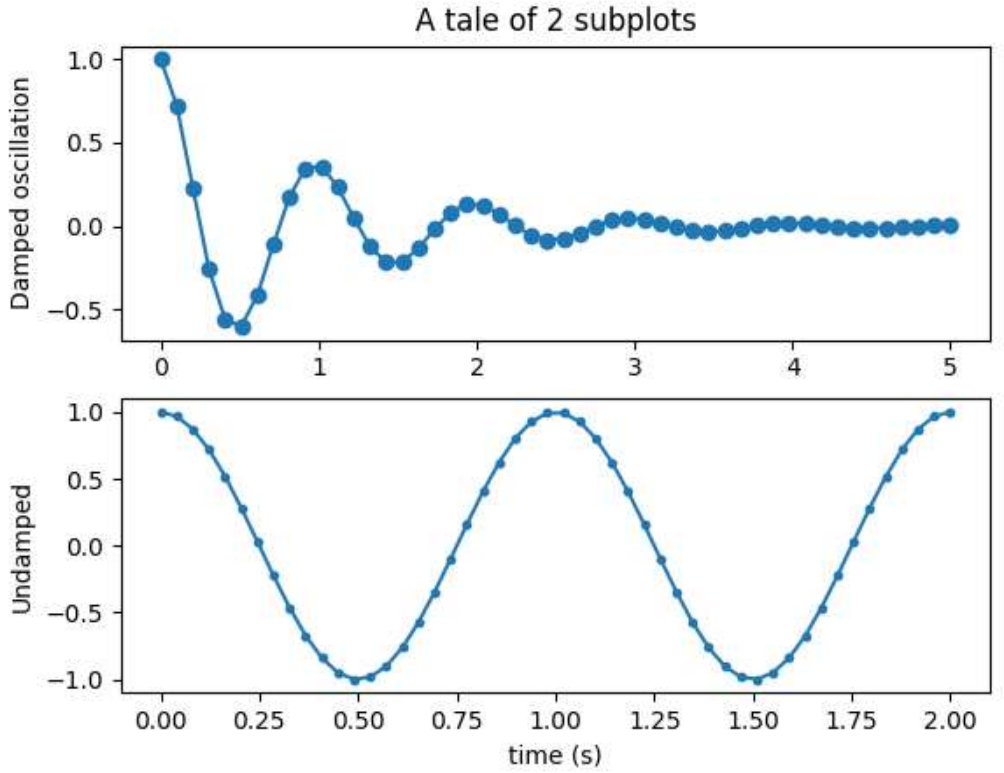
- Matplotlib is probably the most popular Python plotting library
  - Plotly is another good one
- If you are familiar with Matlab plotting then matplotlib is very easy to learn!
- Plots can be made from lists, tuples, numpy arrays, etc.

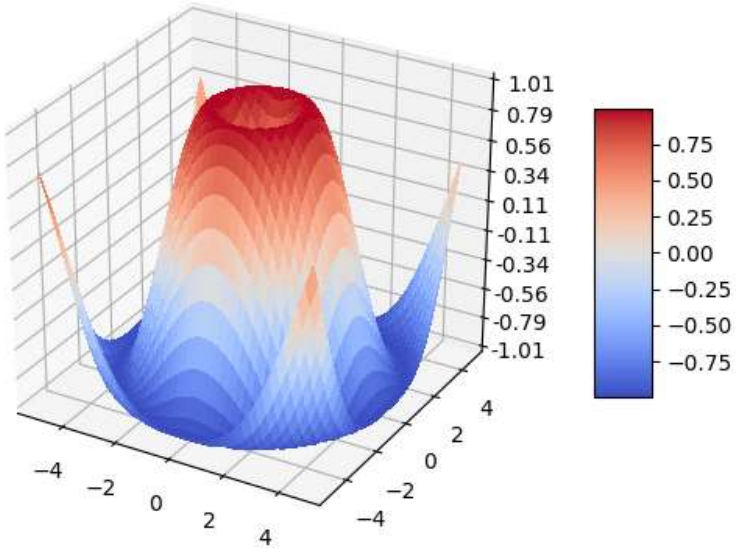
```
import matplotlib.pyplot as plt
plt.plot([5,6,7,8])
plt.show()

import numpy as np
plt.plot(np.arange(5)+3, np.arange(5) / 10.1)
plt.show()
```

Try these out!

BOSTON UNIVERSITY







- Some sample images from matplotlib.org
- A vast array of plot types in 2D and 3D are available in this library.

BOSTON UNIVERSITY

A numpy and matplotlib example

- *numpy_matplotlib_fft.py* is a short example on using numpy and matplotlib together.
- Open *numpy_matplotlib_fft.py*
- Let's walk through this…

Tutorial Outline – Part 2

- Functions
- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

Writing Quality Pythonic Code

- Cultivating good coding habits pays off in many ways:
  - Easier and faster to write
  - Easier and faster to edit, change, and update your code
  - Other people can understand your work

- Python lends itself to readable code
  - It's quite hard to write **completely** obfuscated code in Python.
    - Exploit language features where it makes sense
  - Contrast that with this sample of obfuscated C code.

- Here we'll go over some suggestions on how to setup a Python script, make it readable, reusable, and testable.

Compare some Python scripts

- Open up three files and let's look at them.

- A file that does…something…
  - *bad_code.py*
- Same code, re-organized:
  - *good_code.py*
- Same code, debugged, with testing code:
  - *good_code_testing.py*

Command line arguments

- Try to avoid hard-coding file paths, problem size ranges, etc. into your program.
- They can be specified at the command line.
- Look at the *argparse* module, part of the Python Standard Library.

```
import argparse

parser = argparse.ArgumentParser(description='Process some integers.')
parser.add_argument('integers', metavar='N', type=int, nargs='+',
                    help='an integer for the accumulator')
parser.add_argument('--sum', dest='accumulate', action='store_const',
                    const=sum, default=max,
                    help='sum the integers (default: find the max)')

args = parser.parse_args()
print(args.accumulate(args.integers))
```

```
$ python prog.py -h
usage: prog.py [-h] [--sum] N [N ...]

Process some integers.

positional arguments:
 N           an integer for the accumulator

optional arguments:
 -h, --help  show this help message and exit
 --sum       sum the integers (default: find the max)
```

BOSTON UNIVERSITY

Tutorial Outline – Part 2

- Functions
- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

Classes (writing your own)

- The data types we've used so far are classes!
- Make a list: a = [ ]
- See what functions a list defines internally: dir(a)
- Your own classes can be as simple or complex as you need.

Class syntax

- A class is defined with the keyword *class*, a classname, and a code block.
- Methods always take an extra argument, *self*, and are called with the *self* prefix inside the class.
- Members (i.e. variables) in the class can be added at any time even outside of the class definition.
  - Members are called internally with the *self* prefix.

```
class MyClass:
    # define a class member
    var1 = 1
    # and a class method
    def func1(self,x):
        return self.var1 + x

# make an object
mc = MyClass()
print(mc)
print(mc.var1)
# Call the method
tmp = mc.func1(10)
# what's the value?
print(tmp)

# Add a member to the class
mc.var2 = ['another','member']
print(mc.var2)
```

Initializer

- When an object is instantiated from a class, a special function called the initializer is called to set up the object.
- Syntax:

```
def __init__(self,arg1,…etc…):
    # initialize a member
    self.x = arg1
    # etc
```

- The members are typically created here, files are opened, etc.

A class by example…

- Open the file *read_a_file_classes.py*
- This is a re-write of the earlier code that reads numbers from a file.
- The functionality is pushed into a custom class, *OddEvenNums*.
- Let's walk through and compare to the other solutions.

Other special methods

- To have a class work with print(), implement the *__str__()* method.
- To make a class sortable in a list, implement the “less than” method, *__lt__()*
- To make a class usable as a key in a dictionary, implement the *__hash__()* method.
- For a complete list see the official docs.

Class inheritance

- Classes can *inherit* from other classes.
  - The one being inherited from is called the *parent* or *super* class
  - The one doing the inheriting is called the *child* or *sub* class.
- Sub-classes get all of their parent's members and methods and can add their own.
- This is a very useful feature that really pays off in more complex code.

```
class Shape:
    ''' An empty base class'''
    pass

class Rectangle(Shape):
    ''' Inherits from Shape '''
    def __init__(self,length,width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

class Square(Rectangle):
    ''' A simpler rectangle '''
    def __init__(self,length):
        # Use the Rectangle initializer
        super().__init__(length,length)


rt = Rectangle(10.5,4)
sq = Square(4)

rt.area()  # returns 42.0
sq.area()  # returns 16
```

When to use your own class

- A class works best when you've done some planning and design work before starting your program.
- Simple programs can be written via classes although they will function just like a function-based program.
- Classes can be easier to re-use in other programs compared with a set of functions.

Tutorial Outline – Part 2

- Functions
- Tuples and dictionaries
- Modules
- numpy and matplotlib modules
- Script setup
- Classes
- Development notes

Function, class, and variable naming

- There's no word or character limit for names.
- It's ok to use descriptive names for things.

- BE OBVIOUS. It helps you and others use and understand your code.

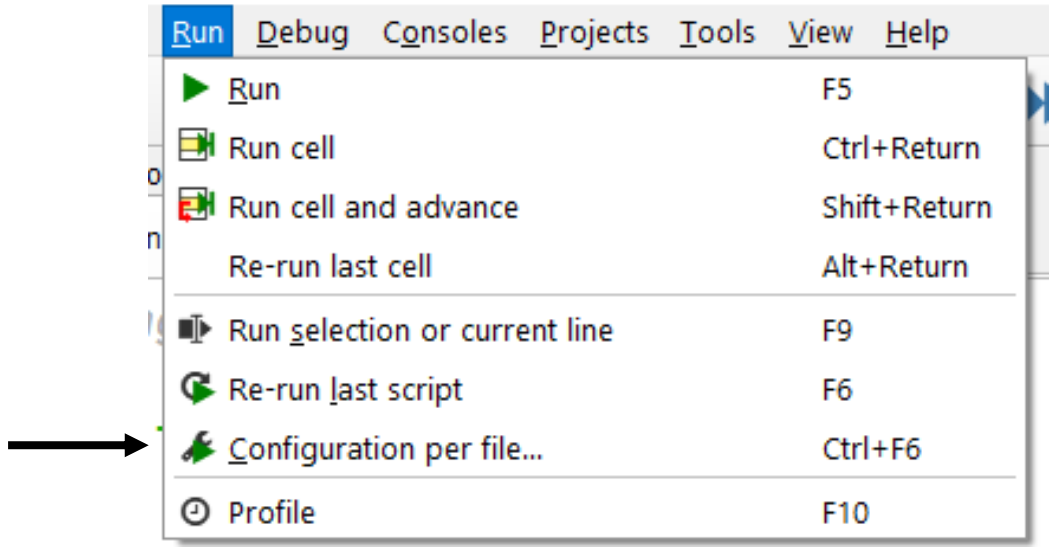
- An IDE (like Spyder) will help you fill in longer names so there's no extra typing anyway!

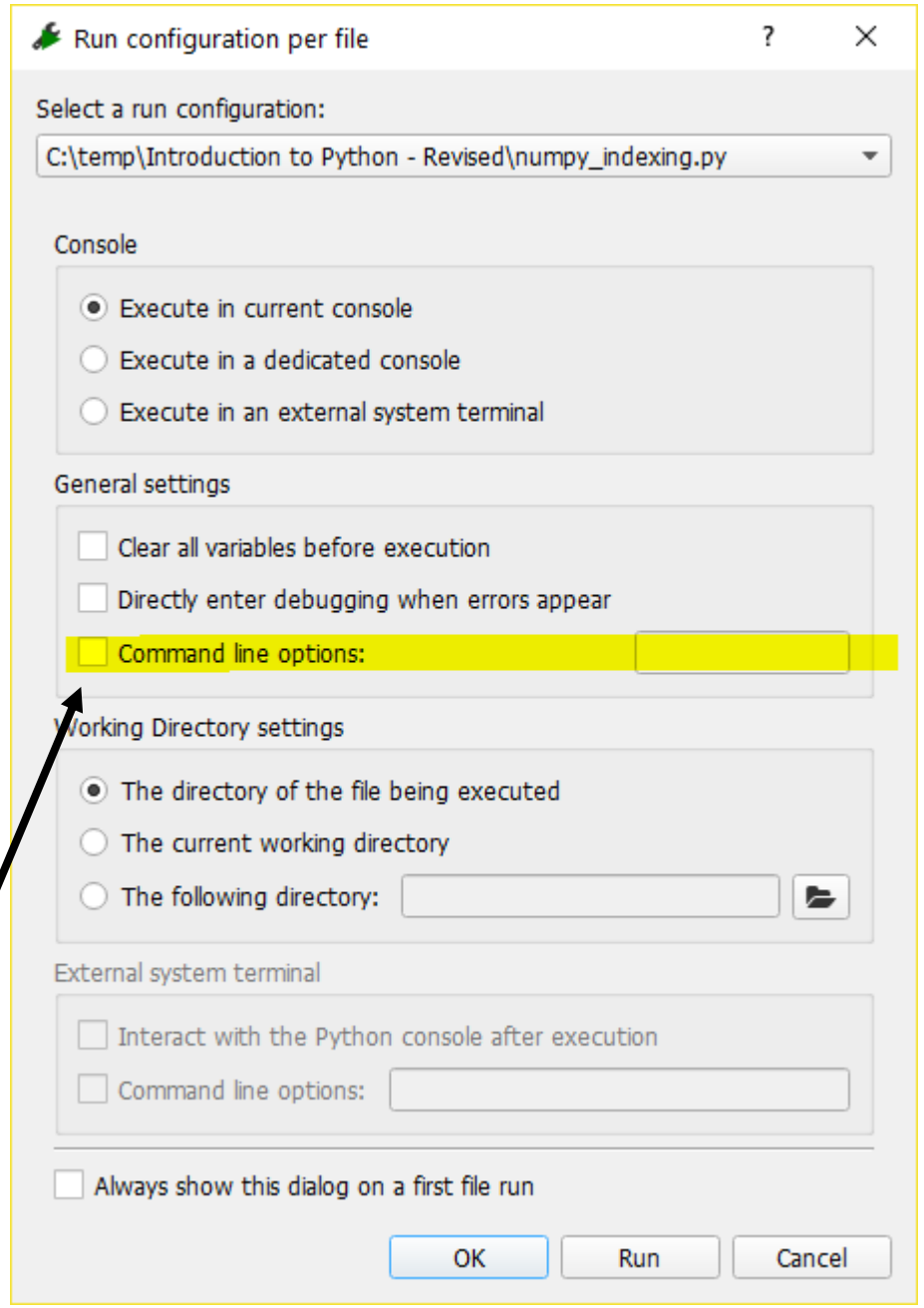
Python from the command line

- A possible development process:
  - Work to develop your program.
  - Put hard-coded values into the *if __name__=='__main__'* section of your code.
  - Once things are underway add command line arguments and remove hard-coded values
  - Modify the Spyder (or Jupyter or other IDE) launch command to use command line arguments.
  - Finally (e.g. to run as an SCC batch job) test run from the command line.

Spyder command line arguments

- Click on the Run menu and choose *Configuration per file*



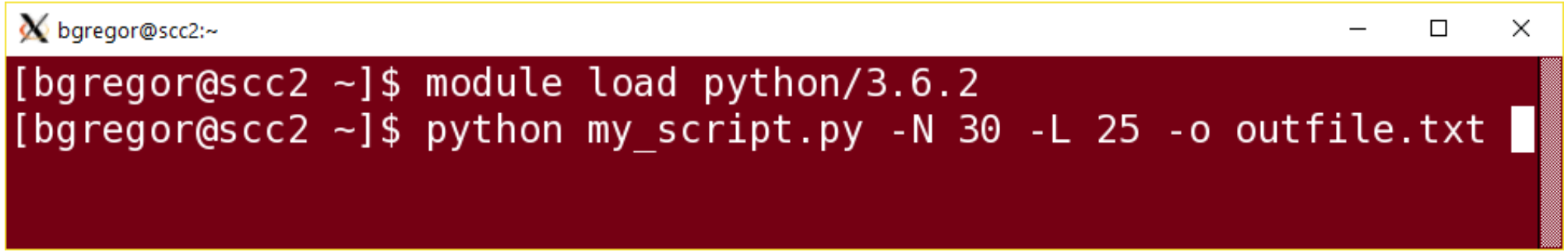


- Enter command line arguments

BOSTON UNIVERSITY

Python from the command line

- To run Python from the command line:



- Just type *python* followed by the script name followed by script arguments.

Where to get help…

- The official Python Tutorial
- Automate the Boring Stuff with Python
  - Focuses more on doing useful things with Python, not focused on scientific computing
- Full Speed Python tutorial
- Contact Research Computing: help@scv.bu.edu

BOSTON UNIVERSITY