Introduction to Python
Part 2

v0.4

Research Computing Services
Information Services & Technology

BOSTON UNIVERSITY

Tutorial Outline – Part 2

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

Lists

- A Python list is a general purpose 1-dimensional container for variables.
  - i.e. it is a row, column, or vector of things
- Lots of things in Python act like lists or use list-style notation.
- Variables in a list can be of any type at any location, including other lists.
- Lists can change in size: elements can be added or removed

Making a list and checking it twice…

- Make a list with [ ] brackets.
- Append with the *append()* function
- Create a list with some initial elements
- Create a list with N repeated elements

```
list_1 = []

list_1.append(1)
list_1.append('A string!')
list_1.append([])


list_2 = [4, 5, -23.0+4.1j, 'cat']


list_3 = 10 * [42]
```

Try these out yourself!
Edit the file in Spyder and run it.
Add some print() calls to see the lists.

List functions

- Try *dir(list_1)*
- Like strings, lists have a number of built-in functions
- Let’s try out a few…
- Also try the len() function to see how many things are in the list: *len(list_1)*

```
'append',
'clear',
'copy',
'count',
'extend',
'index',
'insert',
'pop',
'remove',
'reverse',
'sort']
```

BOSTON UNIVERSITY

List Indexing

- Elements in a list are accessed by an index number.

- Index #'s start at 0.

- List: `x=['a', 'b', 'c', 'd' ,'e']`

- First element: `x[0]` → `'a'`
- Nth element: `x[2]` → `'c'`
- Last element: `x[-1]` → `'e'`
- Next-to-last: `x[-2]` → `'d'`

List Slicing

```
x=['a', 'b', 'c', 'd' ,'e']
x[0:1] → ['a']
x[0:2] → ['a','b']
x[-3:] → ['c', 'd', 'e']
# Third from the end to the end
x[2:5:2] → ['c', 'e']
```

- Slice syntax: `x[start:end:step]`
  - The start value is inclusive, the end value is exclusive.
  - Start is optional and defaults to 0.
  - Step is optional and defaults to 1.
  - Leaving out the end value means “go to the end”
  - Slicing always returns a **new list copied from the existing list**

BOSTON UNIVERSITY

List assignments and deletions

- Lists can have their elements overwritten or deleted (with the *del)* command.

```
x=['a', 'b', 'c', 'd' ,'e']

x[0] = -3.14 → x is now [-3.14, 'b', 'c', 'd', 'e']

del x[-1] →  x is now [-3.14, 'b', 'c', 'd']
```

BOSTON UNIVERSITY

DIY Lists

- Go to the menu File→New File
- Enter your list commands there
- Give the file a name when you save it
- Use print() to print out results

- In the Spyder editor try the following things:
- Assign some lists to some variables.   a = [1,2,3]   b = 3*['xyz']
  - Try an empty list, repeated elements, initial set of elements
- Add two lists: a + b   What happens?
- Try list indexing, deletion, functions from *dir(my_list)*
- Try assigning the result of a list slice to a new variable

BOSTON UNIVERSITY

More on Lists and Variables

- Open the sample file *list_variables.py* but don't run it yet!

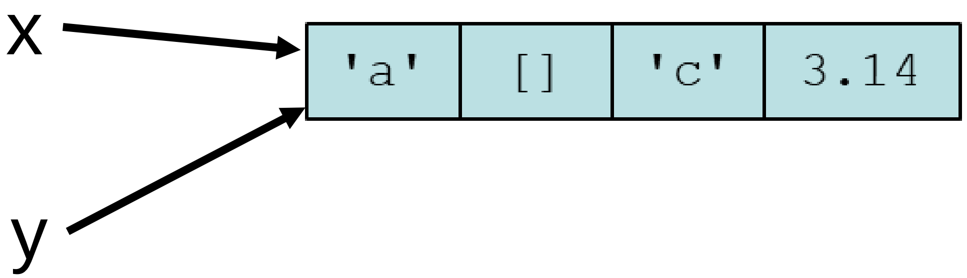
- What do you think will be printed?

```
x = ['a',[],'c',3.14]

y = x

# id() returns a unique identifier for a variable
print('x: %s    addr of x: %s' % (x,id(x)))
print('y: %s    addr of y: %s' % (y,id(y)))


x[0] = -100

print('x: %s' % x)
print('y: %s' % y)
```

Variables and Memory Locations

- Variables refer to a value stored in memory.
- `y = x` does **not** mean “make a copy of the list x and assign it to y” it means “make a copy of the memory location in x and assign it to y”
- x is **not the list** it’s just a reference to it.
- This is how all objects in Python are handled.

```python
x = ['a',[],'c',3.14]

y = x
```



Copying Lists

```
z=x[:]
z[0] = 'frog'
print('x: %s     addr of x: %s' % (x,id(x)))
print('z: %s     addr of z: %s' % (z,id(z)))
```

- How to copy (2 ways…there are more!):
  - `y = x[:]` or `y=list(x)`
- In *list_variables.py* uncomment the code at the bottom and run it.

BOSTON UNIVERSITY

While Loops

- While loops have a condition and a code block.
  - the indentation indicates what's in the while loop.
  - The loop runs until the condition is false.
- The *break* keyword will stop a while loop running.

- In the Spyder edit enter in some loops like these. Save and run them one at a time. What happens with the 1st loop?

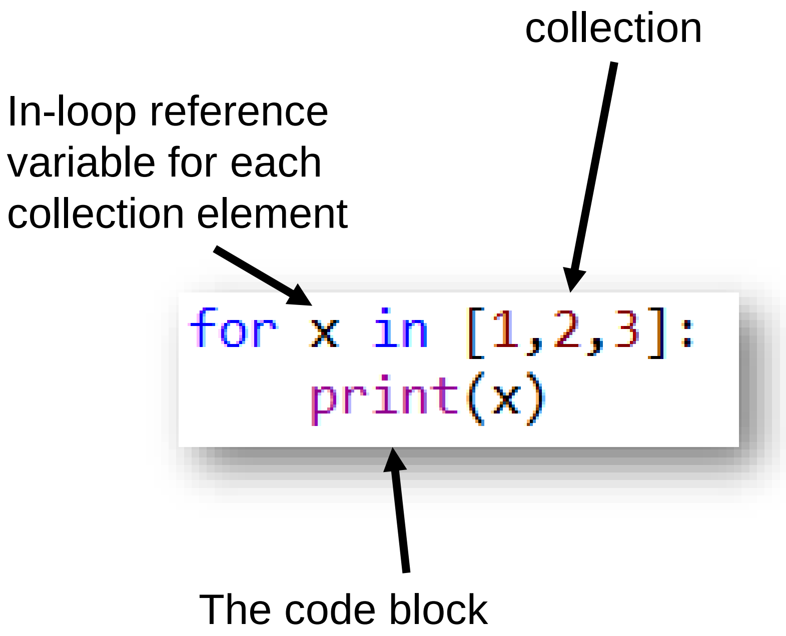
```
while True:
    print("looping!")


a=10
while a > 0:
    print(a)
    a -= 1

my_list=['a','b','c','d','e']
i=0
while i < len(my_list):
    print( my_list[i] )
    i += 1
    if i==3:
        break
```

BOSTON UNIVERSITY

For loops

- *for* loops are a little different. They loop through a collection of things.
- The for loop syntax has a collection and a code block.
  - Each element in the collection is accessed in order by a reference variable
  - Each element can be used in the code block.

- The *break* keyword can be used in for loops too.



Processing lists element-by-element

- A for loop is a convenient way to process every element in a list.

- There are several ways:
  - Loop over the list elements
  - Loop over a list of index values and access the list by index
  - Do both at the same time
  - Use a shorthand syntax called a *list comprehension*
- Open the file *looping_lists.py*
- Let’s look at code samples for each of these.

BOSTON UNIVERSITY

The range() function

- The range() function auto-generates sequences of numbers that can be used for indexing into lists.

- Syntax: range(*start*, *exclusive end*, *increment)*

- range(0,4) → produces the sequence of numbers 0,1,2,3
- range(-3,15,3) → -3,0,3,6,9,12
- range(4,-3,2) → 4,2,0,-2

- Try this: print(range(4))

Lists With Loops

- Open the file *read_a_file.py*

- This is an example of reading a file into a list. The file is shown to the right, *numbers.txt*

- We want to read the lines in the file into a list of strings (1 string for each line), then extract separate lists of the odd and even numbers.

numbers.txt

```
38,83,37,21,98
50,53,55,37,97
39,7,81,87,82
18,83,66,82,47
56,64,9,39,83
…etc…
```

- Let’s walk through this line-by-line using Spyder

- *read_a_file_low_mem.py* is a modification that uses less memory.

Tutorial Outline – Part 2

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

BOSTON UNIVERSITY

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)
```

BOSTON UNIVERSITY

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'
```

BOSTON UNIVERSITY

Try Out Dictionaries

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

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

print(x)
```

BOSTON UNIVERSITY

Tutorial Outline – Part 2

- Lists
- 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.

BOSTON UNIVERSITY

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 January 2019): **164,947**

- 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!

BOSTON UNIVERSITY

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, pycuda, 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 select functions:

```
from math import cos
from math import cos,sqrt
```

- Or rename on the import:

```
from math import sin as pySin
```

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!

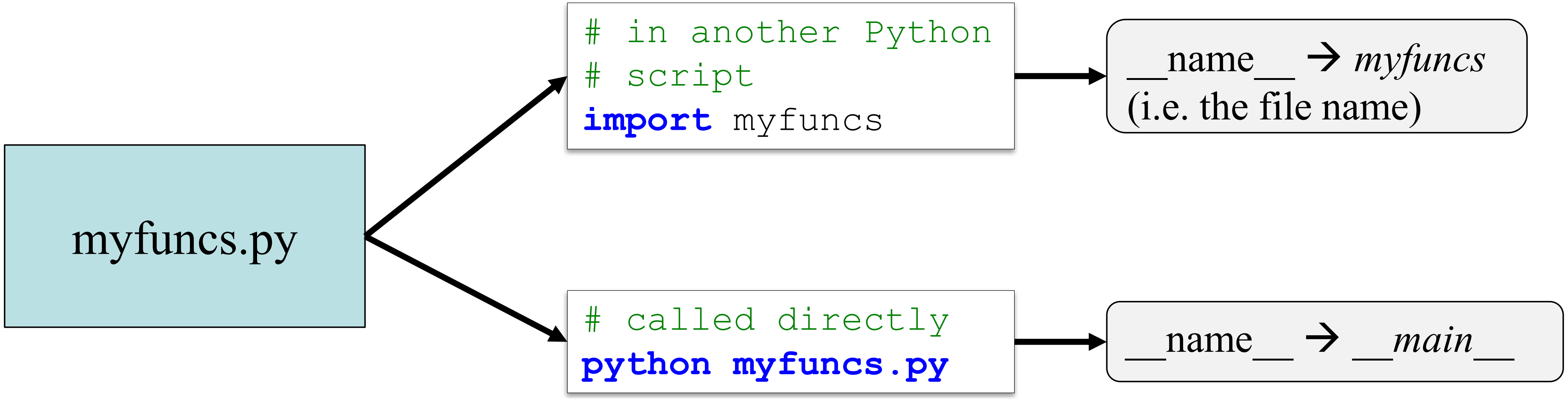
myfuncs.py

```
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

- Lists
- 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)
```

BOSTON UNIVERSITY

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!

BOSTON UNIVERSITY

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*

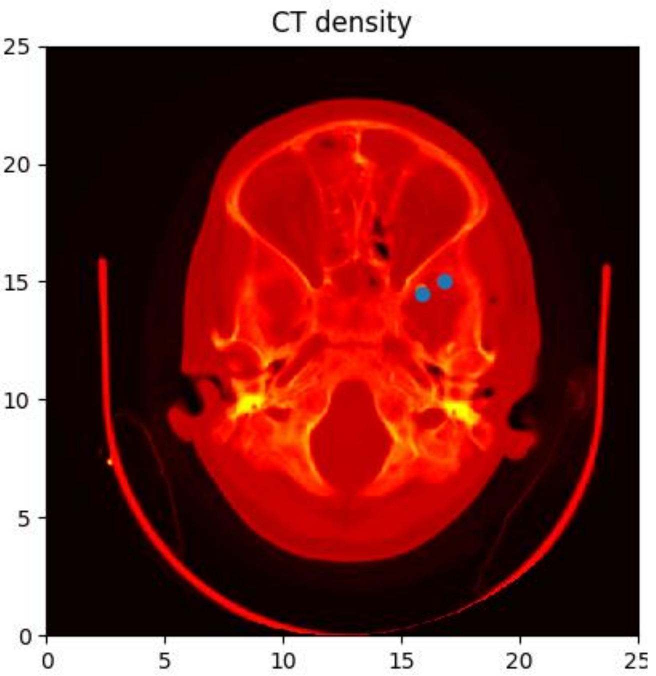
Plotting with matplotlib

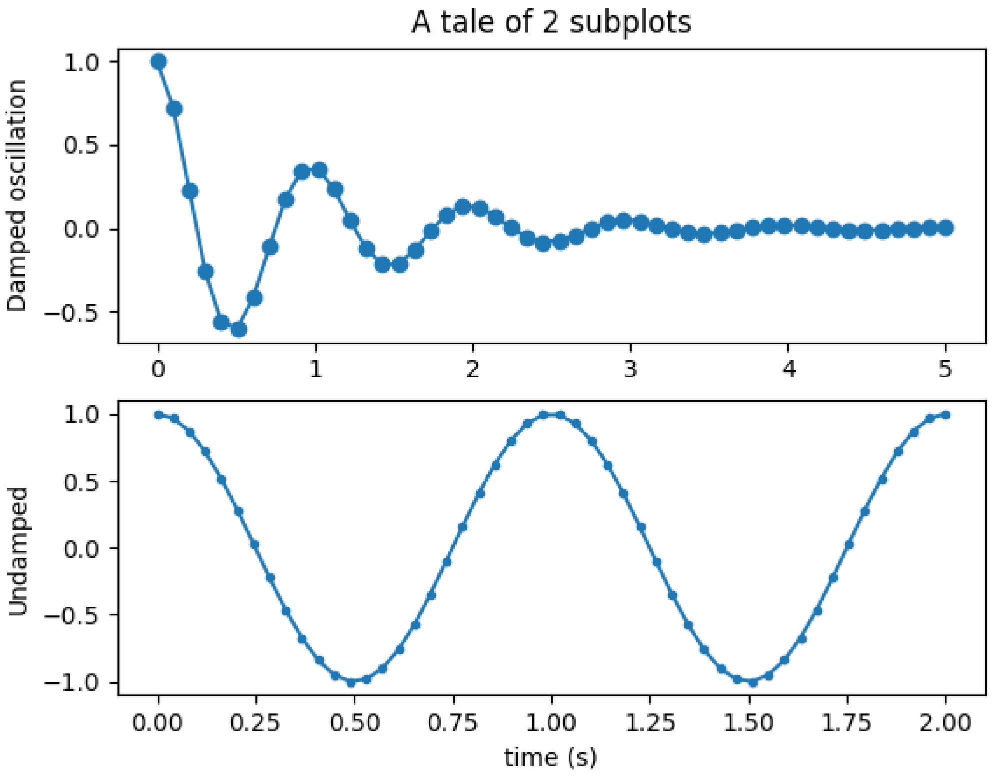
- Matplotlib is probably the most popular Python plotting library
  - Plotly is another good one
- If you are familiar with Matlab plotting then matplotlib is very easy to learn!
- Plots can be made from lists, tuples, numpy arrays, etc.

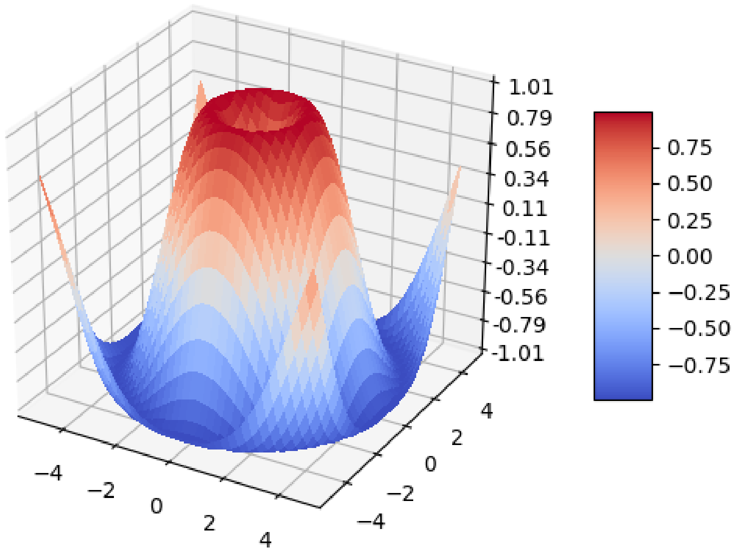
```
import matplotlib.pyplot as plt
plt.plot([5,6,7,8])
plt.show()

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.

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

- Lists
- 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.

BOSTON UNIVERSITY

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

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

Writing Your Own Classes

```python
class Student:
    def __init__(self, name, buid, gpa):
        self.name = name
        self.buid = buid
        self.gpa = gpa

    def has_4_0(self):
        return self.gpa==4.0

me = Student("RCS Instructor","U0000000",2.9)
print(me.has_4_0())
```

- Your own classes can be as simple or as complex as you need.
- Define your own Python classes to:
  - Bundle together logically related pieces of data
  - Write functions that work on specific types of data
  - Improve code re-use
  - Organize your code to more closely resemble the problem it is solving.

BOSTON UNIVERSITY

When to use your own class

- A class works best when you’ve done some planning and design work before starting your program.

- This is a topic that is best tackled after you’re comfortable with solving programming problems with Python.

- Some tutorials on using Python classes:

  W3Schools: https://www.w3schools.com/python/python_classes.asp

  Python tutorial: https://docs.python.org/3.6/tutorial/classes.html

BOSTON UNIVERSITY

Tutorial Outline – Part 2

- Lists
- 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.
- An IDE (like Spyder) will help you fill in longer names so there’s no extra typing anyway.
- Give your functions and variables names that reflect their meaning.
  - Once a program is finished it’s easy to forget what does what where

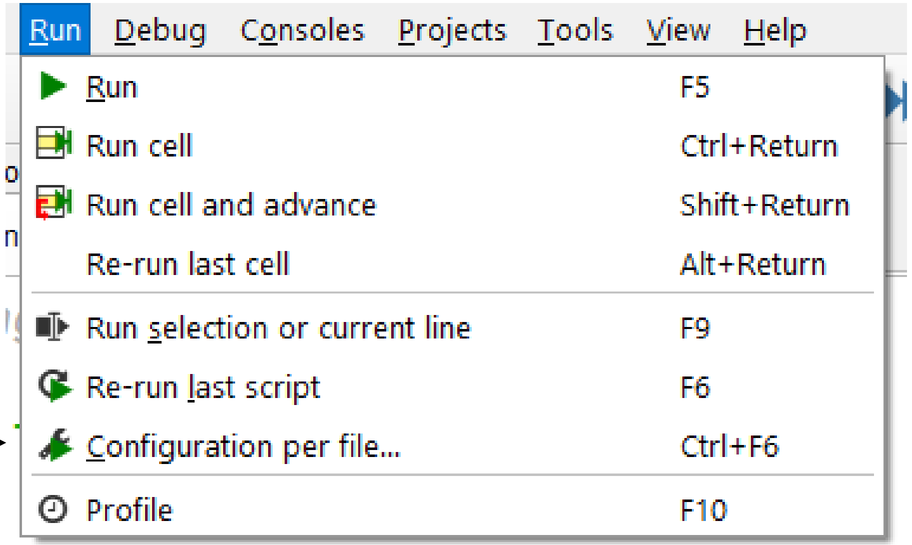
An example development process

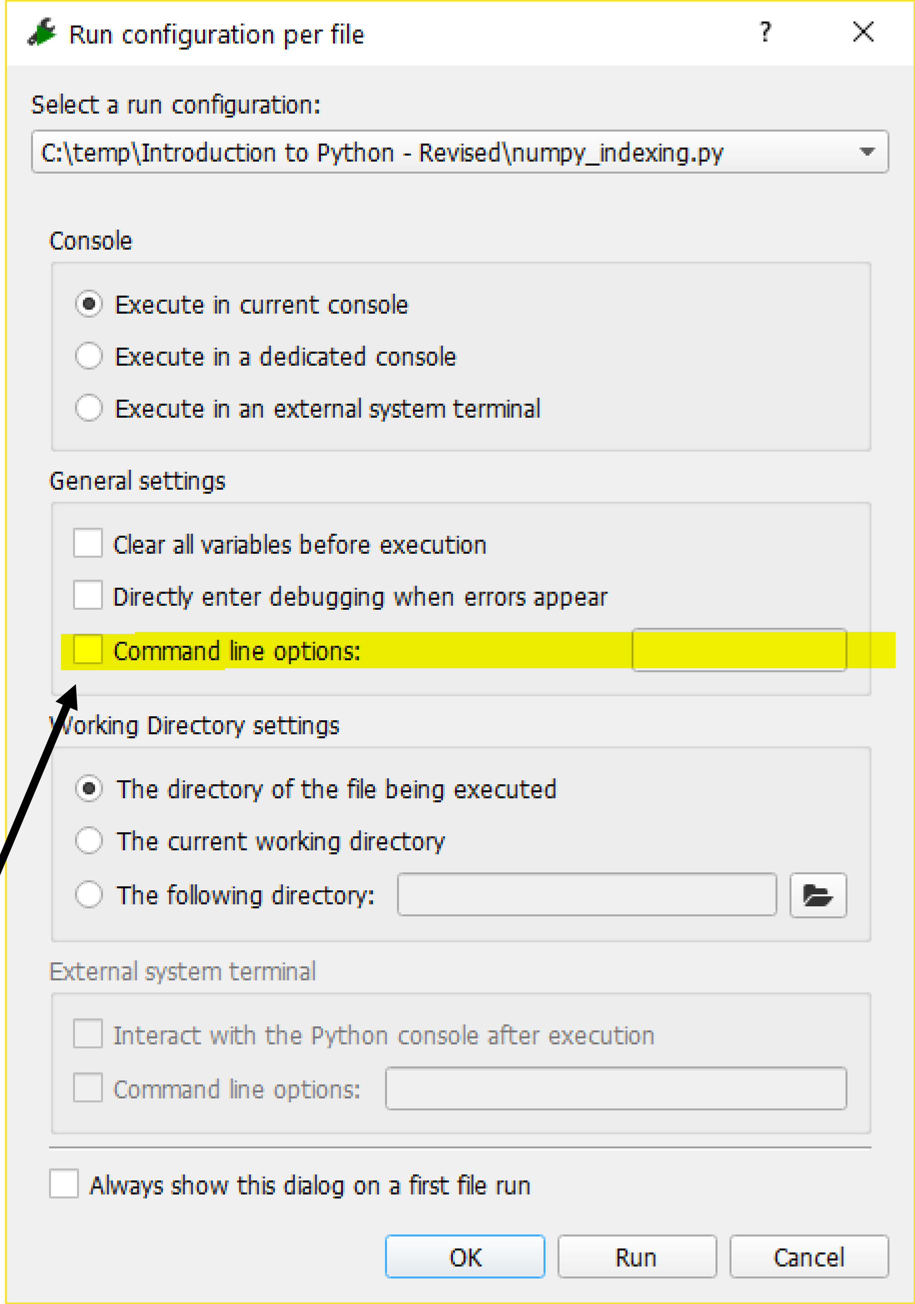
- Work to develop your program.
  - Do some flowcharts, work out algorithms, and so on.
  - Write some Python to try out a few ideas.
  - Get organized.
- Write a “1st draft” version that gets most of what’s needed done.
- Move hard-coded values into the *if __name__==‘__main__’* section of your code.
- Once the code is testing well add command line arguments and remove hard-coded values
- Finally (e.g. to run as an SCC batch job) test run from the command line.

BOSTON UNIVERSITY

Spyder command line arguments

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





- Enter command line arguments

Python from the command line

- To run Python from the command line:

```
bgregor@scc2:~
[bgregor@scc2 ~]$ module load python/3.6.2
[bgregor@scc2 ~]$ python my_script.py -N 30 -L 25 -o outfile.txt
```

- 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