Introduction to Python
Part 2

Brian Gregor
Research Computing Services
Information Services & Technology

BOSTON UNIVERSITY

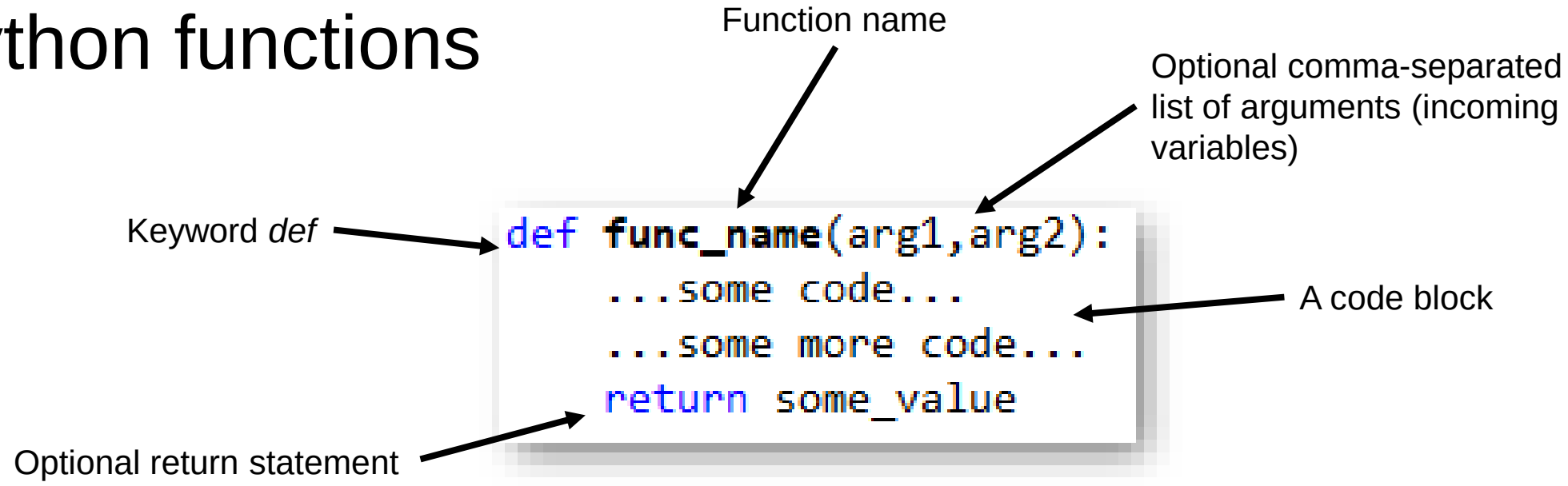
Tutorial Outline – Part 2

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

Functions

- Functions are used to create code that can be used in a program or in other programs.
- The use of functions to logically separate the program into discrete computational steps.
- Programs that make heavy use of function definitions tend to be easier to develop, debug, maintain, and understand.

Python functions



- The return value can be any Python type
- If the return statement is omitted a special *None* value is still returned.
- The arguments are optional but the parentheses are required!
- Functions must be defined before they can be called.

Function Return Values

- A function can return any Python value.
- Function call syntax:

```
A = some_func()       # return a value
Another_func()          # ignore return value or nothing returned
b,c = multiple_vals(x,y,z)   # return multiple values
```

- Open *function_calls.py* for some examples

Function arguments

- Function arguments can be required or optional.
- Optional arguments are given a default value

```
def my_func(a,b,c=10,d=-1):
    …some code…
```

- To call a function with optional arguments:
- Optional arguments can be used in the order they're declared or out of order if their name is used.

```
my_func(x,y,z)          # a=x, b=y, c=z, d=-1
my_func(x,y)            # a=x, b=y, c=10, d=-1
my_func(x,y,d=w,c=z)    # a=x, b=y, c=z, d=w
```

Function arguments

- Remember the list assignment?

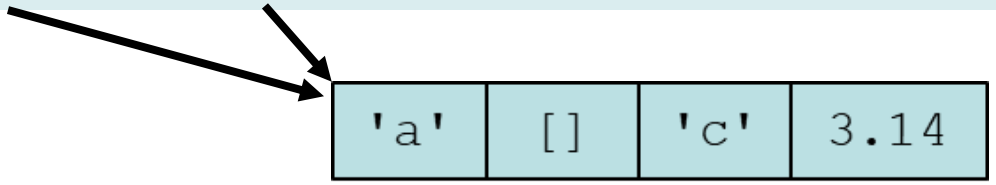
```
x = ['a', [], 'c', 3.14]
y=x  # y points to the same list as x
```

- This applies in function calls too.

```
def my_func(a_list):
    # modifies the list in the calling routine!
    a_list.append(1)
```

- Then call it:

```
my_func(x)     # x and a_list inside the function are the same list!
```



Garbage collection

- Variables defined in a function (or in any code block) no longer have any “live” references to them once the function returns.
- These variables become *garbage*, and *garbage collection* operates to remove them from the computer’s memory, freeing up the memory to be re-used.
- There is no need to explicitly destroy or release most variables.
  - Some complex data types provide .close(), .clean(), etc. type functions. Use these where available.
  - Simple data types (int, string,lists) will be taken care of automatically.

When does garbage collection occur?

- That's hard to say. It happens when Python thinks it should.
- For the great majority of programs this is not an issue.
- Programs using very large quantities of memory or allocating large chunks of memory in repeated function calls can run into trouble.

```
def my_func(N):
    # make a large list
    tmp = [1]*N
    # get its sum
    sum_tmp = sum(tmp)
    return sum_tmp


# What happens to the list created for tmp?
# It gets garbage collected.
# when?  ????

# Call my_func with a large N repeatedly
sums = []
for i in range(1000):
    sums.append(my_func(100000))
```

Memory usage in functions

- If possible, pre-allocate a list or other large data structures and re-use it.
  - i.e. allocate outside the function call and send it as a function argument. Remember function arguments are just references so they don't copy large data structures.

```
my_lst = 1000*[0]
for i in range(1000):
    my_lst[i] = some_func()

# that's better than:
my_lst = []
for i in range(1000):
    my_lst.append(some_func())
```

- There are numerous tools to *profile* a program, which means exam the CPU and memory usage of different parts of the program.

- Need to control memory cleanup? Python provides a library for this.

BOSTON UNIVERSITY

Tutorial Outline – Part 2

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

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

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

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
```

Tutorial Outline – Part 2

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

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*

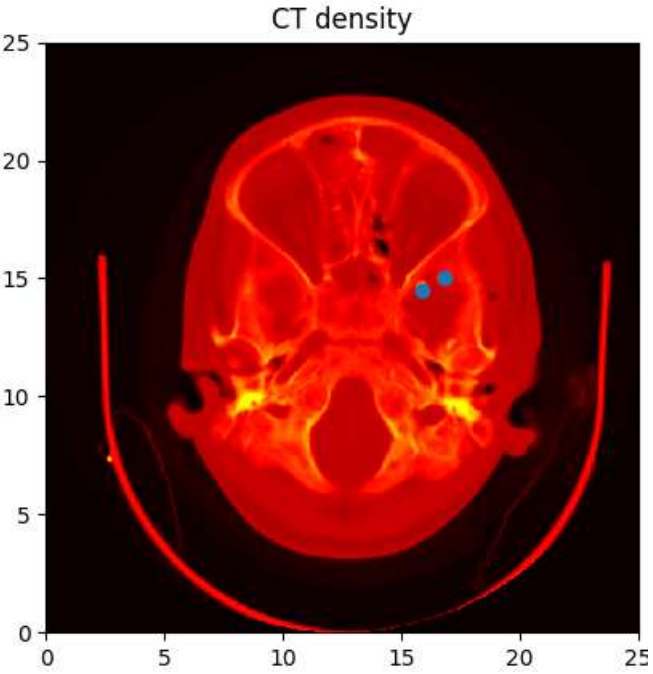
Plotting with matplotlib

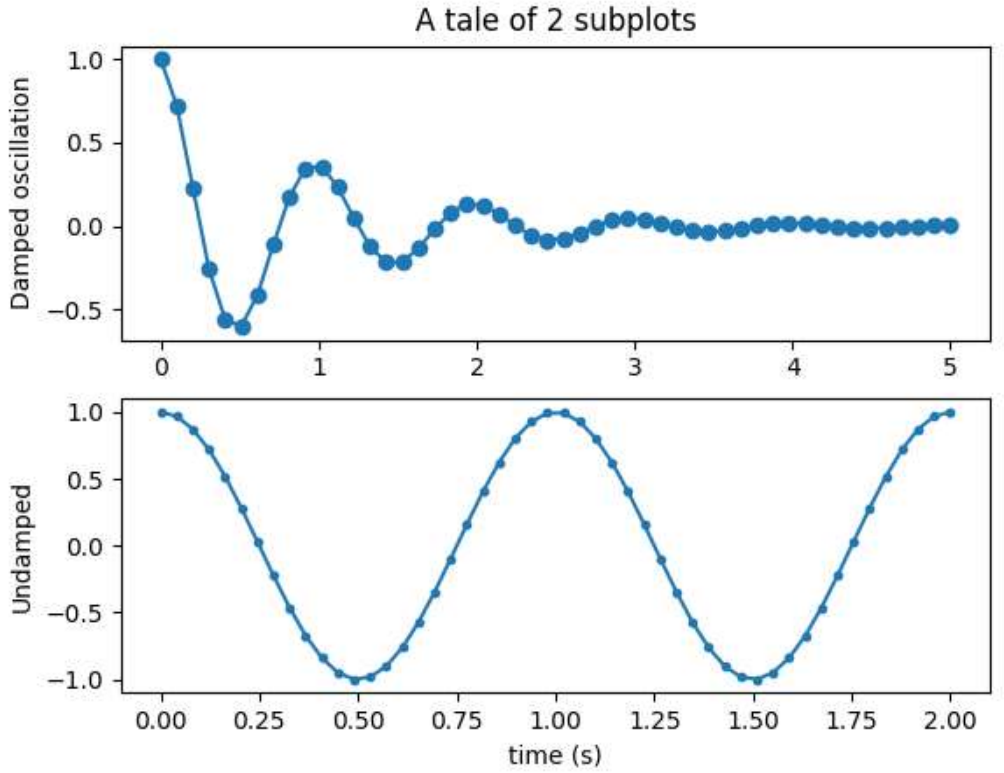
- Matplotlib is probably the most popular Python plotting library
  - Plotly is another good one
- If you are familiar with Matlab plotting then matplotlib is very easy to learn!
- Plots can be made from lists, tuples, numpy arrays, etc.

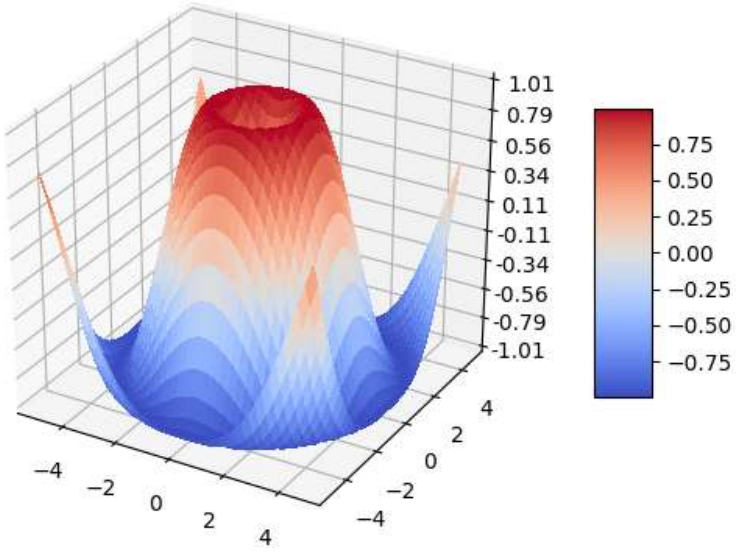
```
import matplotlib.pyplot as plt
plt.plot([5,6,7,8])
plt.show()

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

- Let’s walk through 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
- Debugging

Classes and objects

- Python is a fully object oriented programming (OOP) language.
- In OOP a *class* is a data structure that combines data with functions that operate on that data.
- An *object* is a variable whose type is a *class*
  - Also called an *instance* of a class
- Classes provide a lot of power to help organize a program and can improve your ability to re-use your own code.

Classes

- The data types we've used so far are classes!
- Make a list: a = [ ]
- See what functions a list defines internally: dir(a)
- Functions internal to a class are called *methods*.
- Variables stored in a class (like the length of a list) are called *members*
- When writing your own they 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)
```

Try this out!

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,…args…):
    # 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.
  - Less coding, fewer bugs, easier to maintain
- However…it's outside the scope of this tutorial.

```
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.

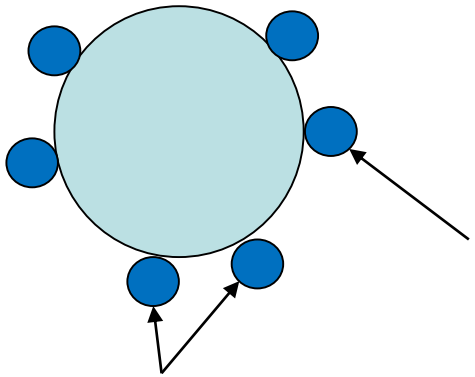
An example with classes

- Open *faces_by_tables.py*

- This is a program that produces images of students assigned to groups at tables in a classroom.
- The output is an image per table with the student's faces and names in groups by rows.
- A naturally object-oriented situation.
- Each class tracks on the info it needs to define itself.

Classroom: a classroom has some Tables.

Table: a Table contains one or more Groups.



Student: Students have names, BUIDs, and a picture.

Group: A group has a name and one or more students.

Writing Good 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.
  - Contrast that with this sample of obfuscated C code.
  - But some attention should still be paid…

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

Think of others

- You should think of (at least!) three other people when working on your program.
  - **EVEN** for little 'one-off' scripts. Many such programs take on a life of their own.
- Person 1: Yourself, tomorrow.
  - Your code may seem blindingly obvious to you late at night after several espressos, but it may be less so the next afternoon.
- Person 2: Yourself, in six months.
  - Same reasoning as the previous, x1000.
- Person 3: Your replacement in your lab.
  - Ever inherit a program from a previous researcher? What would you like to have handed to you to work on?

Comment your code

- Remember those (min.) 3 people to think about?

ADD PLENTY OF COMMENTS TO YOUR CODE.

A habit of heavily commenting your code will make you popular and appreciated in your research group.

BOSTON UNIVERSITY

Work with functions and classes

- Break up your program into logical chunks of functionality.
- Place those chunks into functions.
- Related functions and data can go into a class.
- Maximum length of a function: ideally no more than one screen's worth of code.

What's in a __name__?

- Add the __name__ convention to your program
- Once your functionality is in functions, craft a "main" section that will be run when Python reads your script.
- When a file is read by Python a hidden attribute called __name__ is set to the value of __main__
- If imported into another Python program the __name__ is set to the name of the file.
- This separates the implementation of the program its execution.

```
def some_func(x):
    # .. do something ...
    return 1

def read_file(fname):
    # read a file
    # do something
    return some_val

# This is "my_prog.py"
# When run with:
#     python my_prog.py
# the following if statement will run:
if __name__=='__main__':
    filename = "myfile.txt"
    data = read_file(filename)
    result = some_func(x)
```

BOSTON UNIVERSITY

Compare some Python scripts

- Open up three files and let's look at them.
- Just look at *bad_code.py* to start.
  - *bad_code.py*
  - *good_code.py*
  - *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

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.

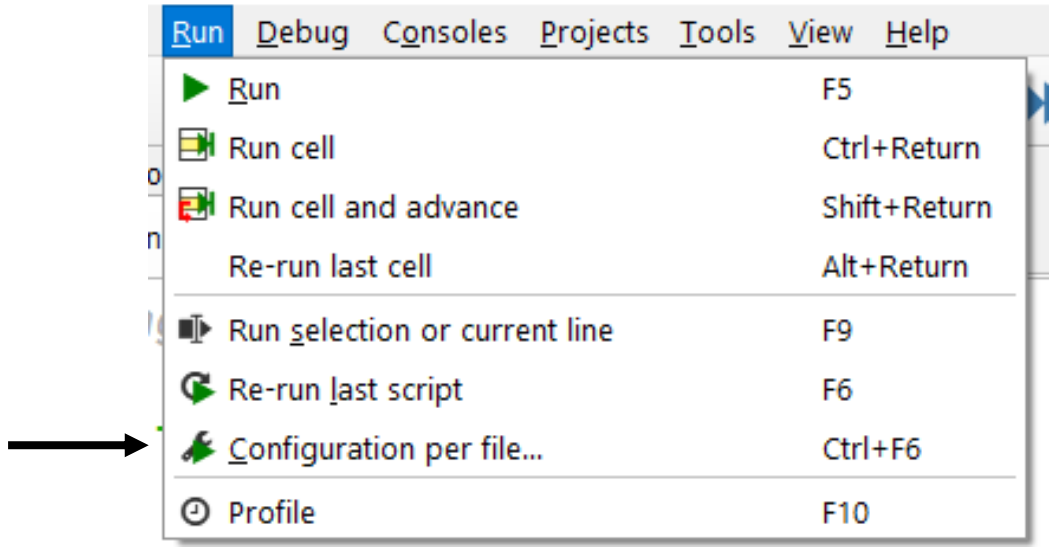
- An IDE (like Spyder) will help you fill in longer names so there's no extra typing anyway!

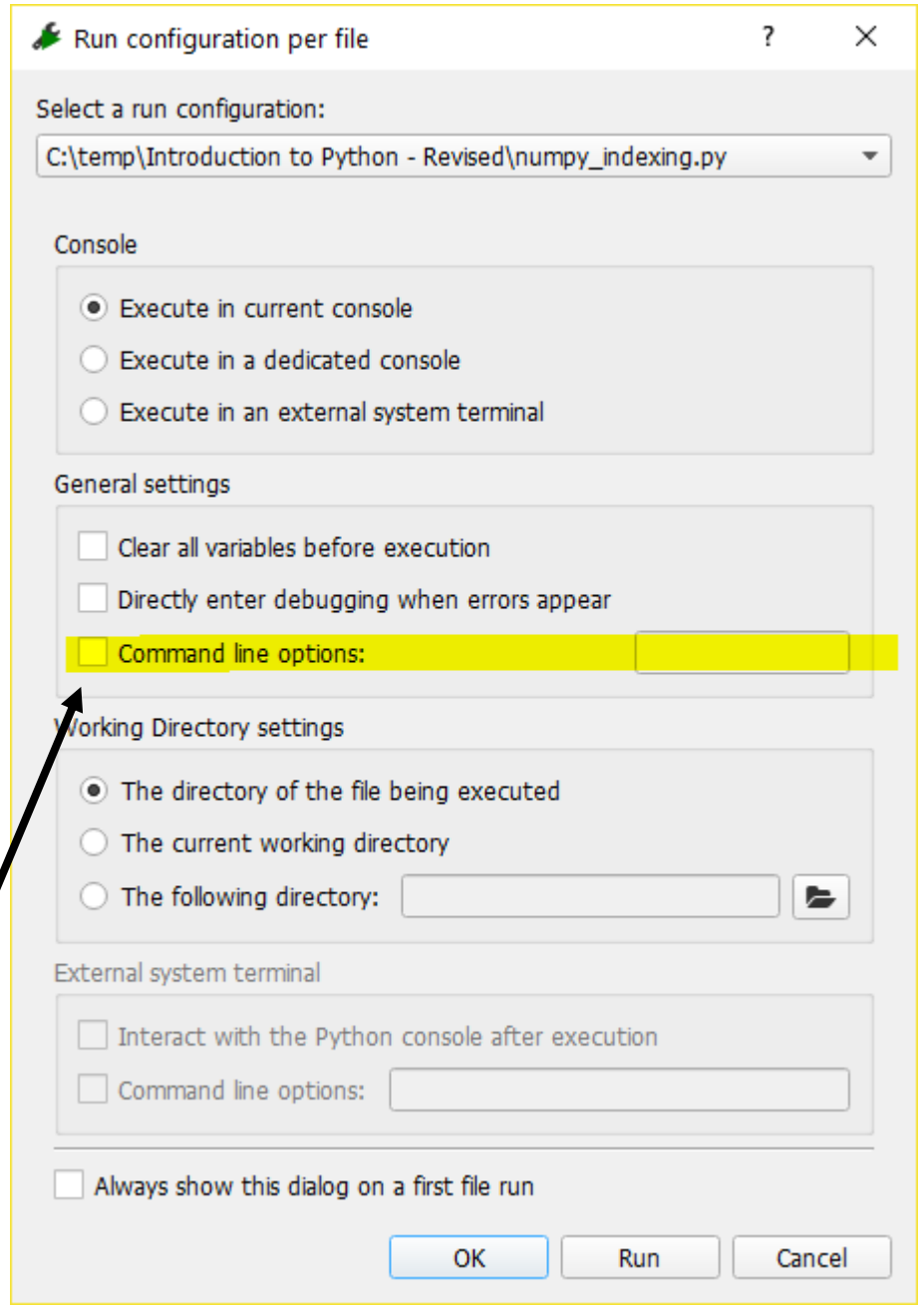
Python from the command line

- A possible development process:
  - Work on 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 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*



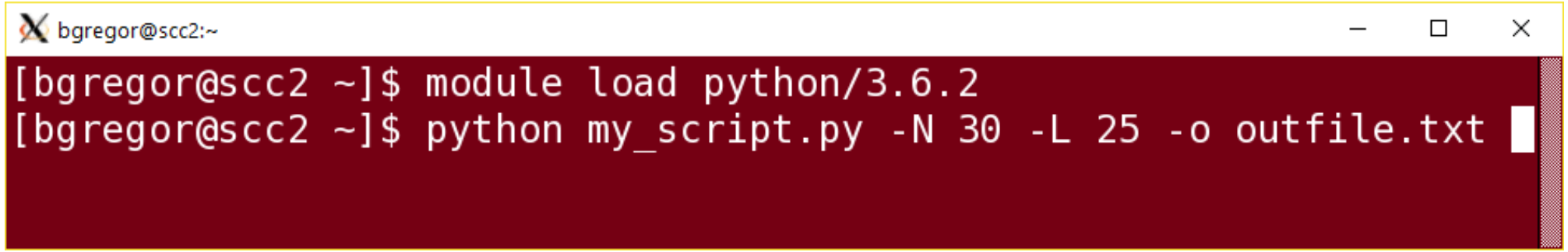


- Enter command line arguments

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.

Tutorial Outline – Part 2

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

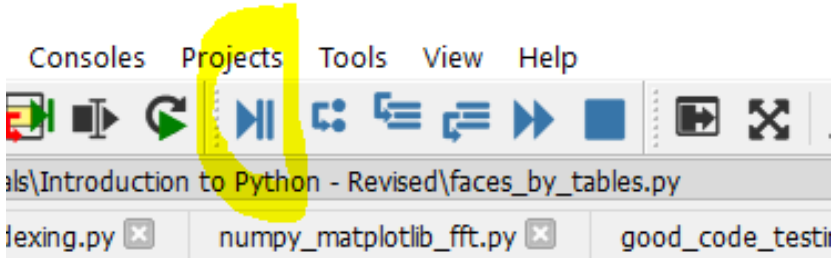
Ways to debug

- There are many ways to debug Python programs.
- As an interpreted language Python is easier to debug than compiled ones.
  - The interpreter has a complete understanding of the state of the program and it can manipulate it at any time.
  - A debugger can freely change variable values, function definitions, etc. inside a running program!
- The main tool is *pdb*, the Python debugger.
  - IDEs like Spyder make this very easy to use.

Alternate methods

- Sprinkle print() functions throughout your code.
  - This is **very popular**.
  - It can be effective.
  - Not as fast or as reliable as using pdb, but it has its place.
- Make code changes until it works.
  - Alas, also popular.
- Implement tests on your functions so that you know they work correctly.
  - If you do this while developing it's easy!
  - The more people there are working on a program (concurrently or over time) the greater the advantage of this method.

Use the debugger



- Let's go back to the *faces_by_table.py* file and step through it using the debugger to watch how it runs.

- Double-click to the left of line number 227 to make the debugger pause when we reach that line:

```
227    classroom = Classroom(os.path.join(input_dir,seating_csv),students)
```

- Now run the program using the debugger. Normally we'd click the icon but to use the command line arguments enter this into the console:

```
debugfile('faces_by_tables.py',args='student_images seating_chart.csv
seating_charts')
```

BOSTON UNIVERSITY

Tutorial Outline – Part 2

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

Where to get help…

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

BOSTON UNIVERSITY