INTRODUCTION TO PYTHON

ECRF Tutorial

Stuart Lacy

7th July 2015

LEARNING OUTCOMES

- Understand why Python is becoming a popular language for Scientific Research
- Know how to run basic Python scripts
- Understand basic Python syntax, data structures and control flow
- Be aware of how Python can be used in a similar fashion to Matlab

TABLE OF CONTENTS

OVERVIEW

WHY LEARN PYTHON?

- Commonly used in research software

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages
- Used in a large range of areas, not just research software

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages
- Used in a large range of areas, not just research software
- Batteries included standard library

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages
- Used in a large range of areas, not just research software
- Batteries included standard library
- Cross-platform

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages
- Used in a large range of areas, not just research software
- Batteries included standard library
- Cross-platform
- Free (speech and money)

WHY LEARN PYTHON?

- Commonly used in research software
- 'Beautiful' language - "code is read more than it is written"
- Very concise, can write programs much quicker than in other languages
- Used in a large range of areas, not just research software
- Batteries included standard library
- Cross-platform
- Free (speech and money)
- Large community of users

FUN

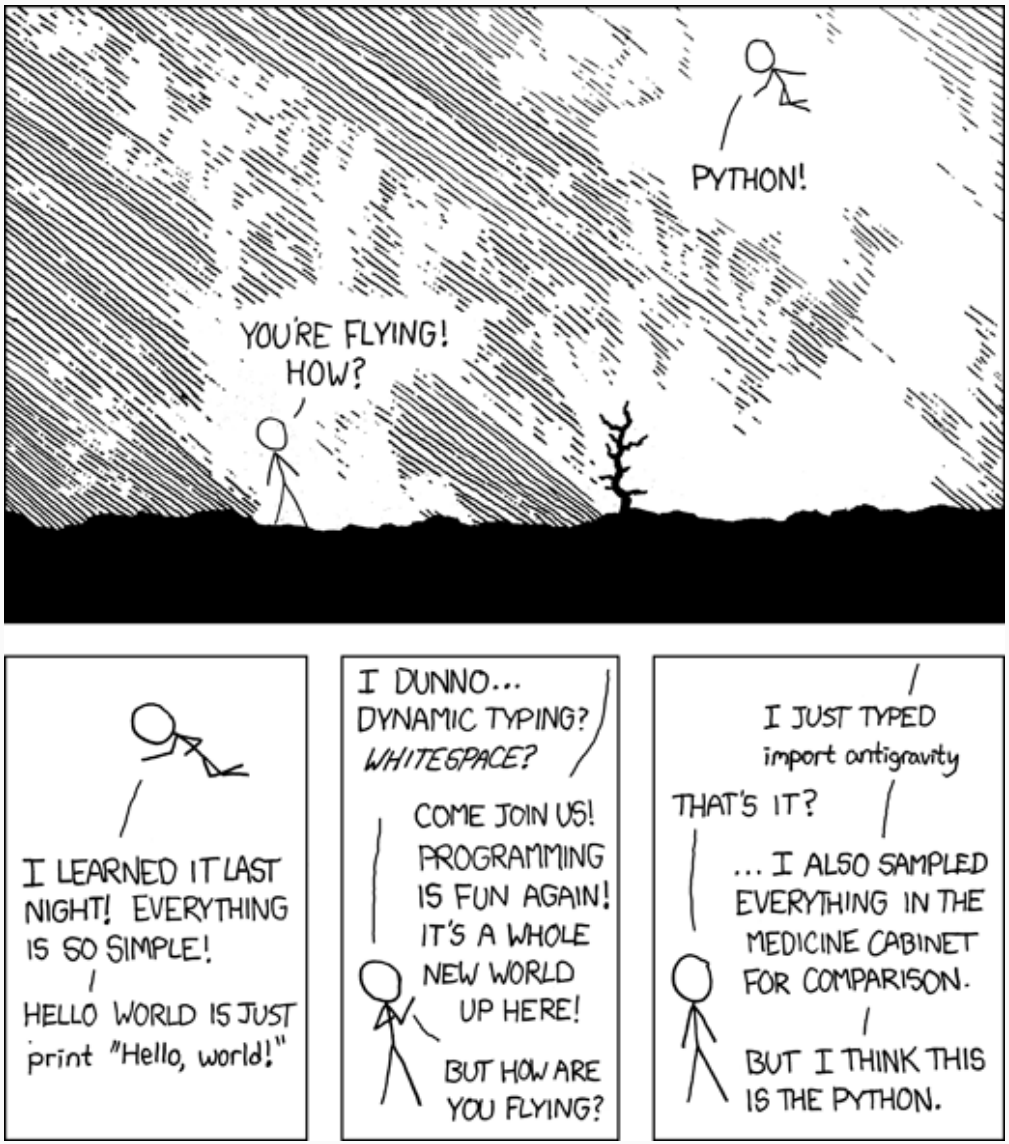


Figure: Taken from https://xkcd.com/353/

POPULARITY

Jun 2015	Jun 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	17.822%	+1.71%
2	1	↓	C	16.788%	+0.60%
3	4	↑	C++	7.756%	+1.33%
4	5	↑	C#	5.056%	+1.11%
5	3	↓	Objective-C	4.339%	-6.60%
6	8	↑	Python	3.999%	+1.29%
7	10	↑	Visual Basic .NET	3.168%	+1.25%
8	7	↓	PHP	2.868%	+0.02%
9	9		JavaScript	2.295%	+0.30%
10	17	⇈	Delphi/Object Pascal	1.869%	+1.04%

Figure: Taken from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

READABLE CODE

- Advantages:
  - Dynamic typing = No types cluttering up screen (`int`, `String` etc...)
  - No semi-colons
  - Significant whitespace forces you to write clean code
- Disadvantages:
  - Dynamically typed means you lack compile time checking
  - Also means it can't be compiled ahead of time so it's run in an interpreter = slow

APPLICATION AREAS

- Research software
- Web development
- Rapid prototyping
- Creating GUI front ends
- Glue code
- Utility scripts
- Quick scripts

SETTING UP PYTHON

Python 2 vs Python 3

- Two versions of Python in current use, Python 2 and Python 3
- Python 3 first came out 7 years ago and last major version of Python 2 (2.7) came out 5 years ago
- Yet there's still a considerable userbase still on 2.7
- The only reason some people are still on Python 2 is due to package compatibility
- Use Python 3 (current version is 3.4)

INSTALLING PYTHON

- Installed by default on most Linux operating systems (including Mac), or will be in official repositories
- Will probably have both Python2 and Python3, check which version you have by running `python -V`
- On Windows download from https://www.python.org/downloads/

INSTALLING PACKAGES

- There is a package manager called Pip that installs new packages for you (ships with 3.X, not 2.X)
- `pip install <package>`
- don't run pip with sudo
- If using an IDE can install through that as well

REPL

- Comes with a Read-Evaluate-Print-Loop (REPL) (separate executable on Windows)
- Useful for checking algorithms, or small functions
- Use as a calculator

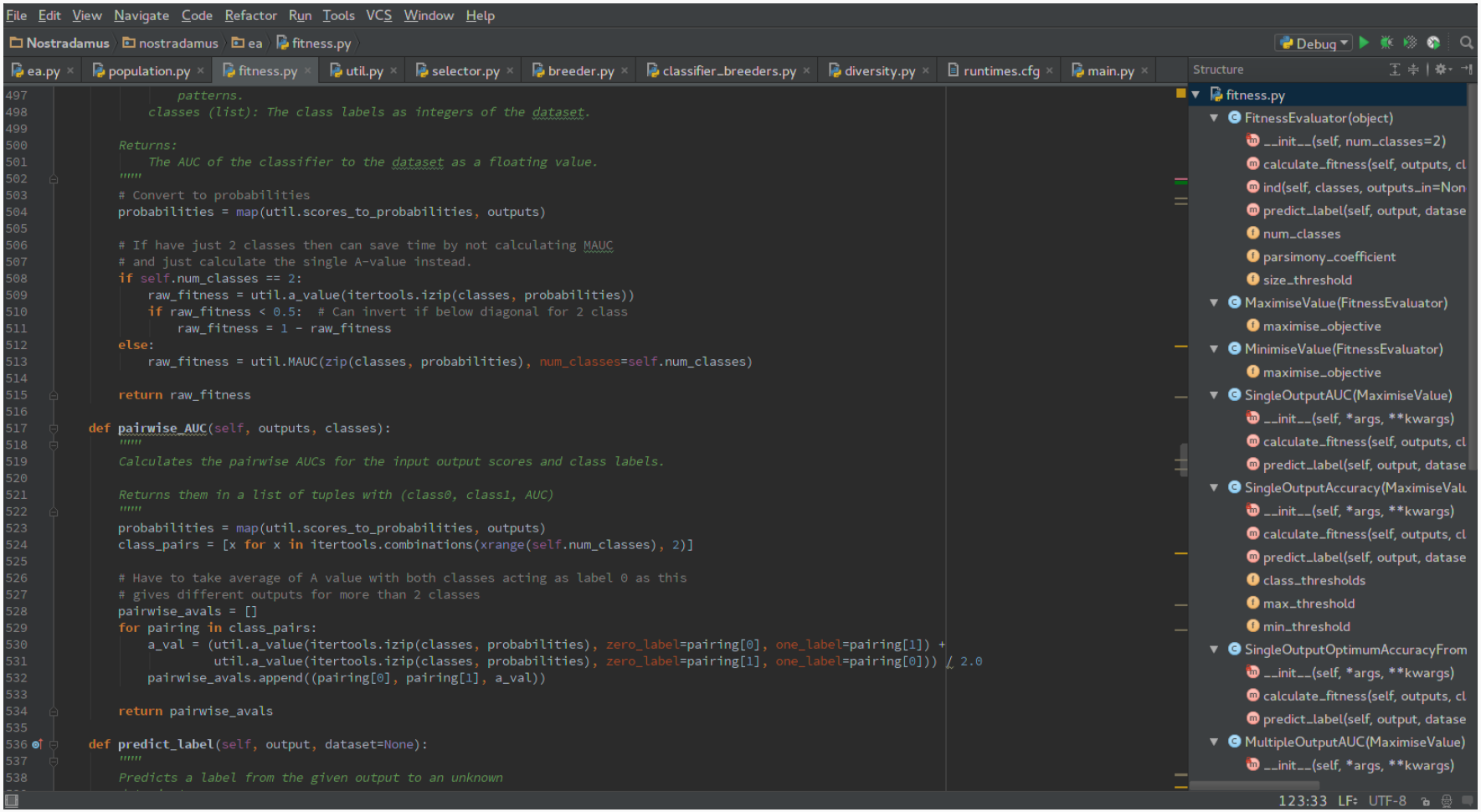
RUNNING SCRIPTS

- Python files (known as modules) are saved with .py file extension
- No limit to what's saved in a module, can have as many classes and functions as you want
- To run manually just execute `python file.py` from command line
- **NB: In Windows you'll need to set Path before doing this**

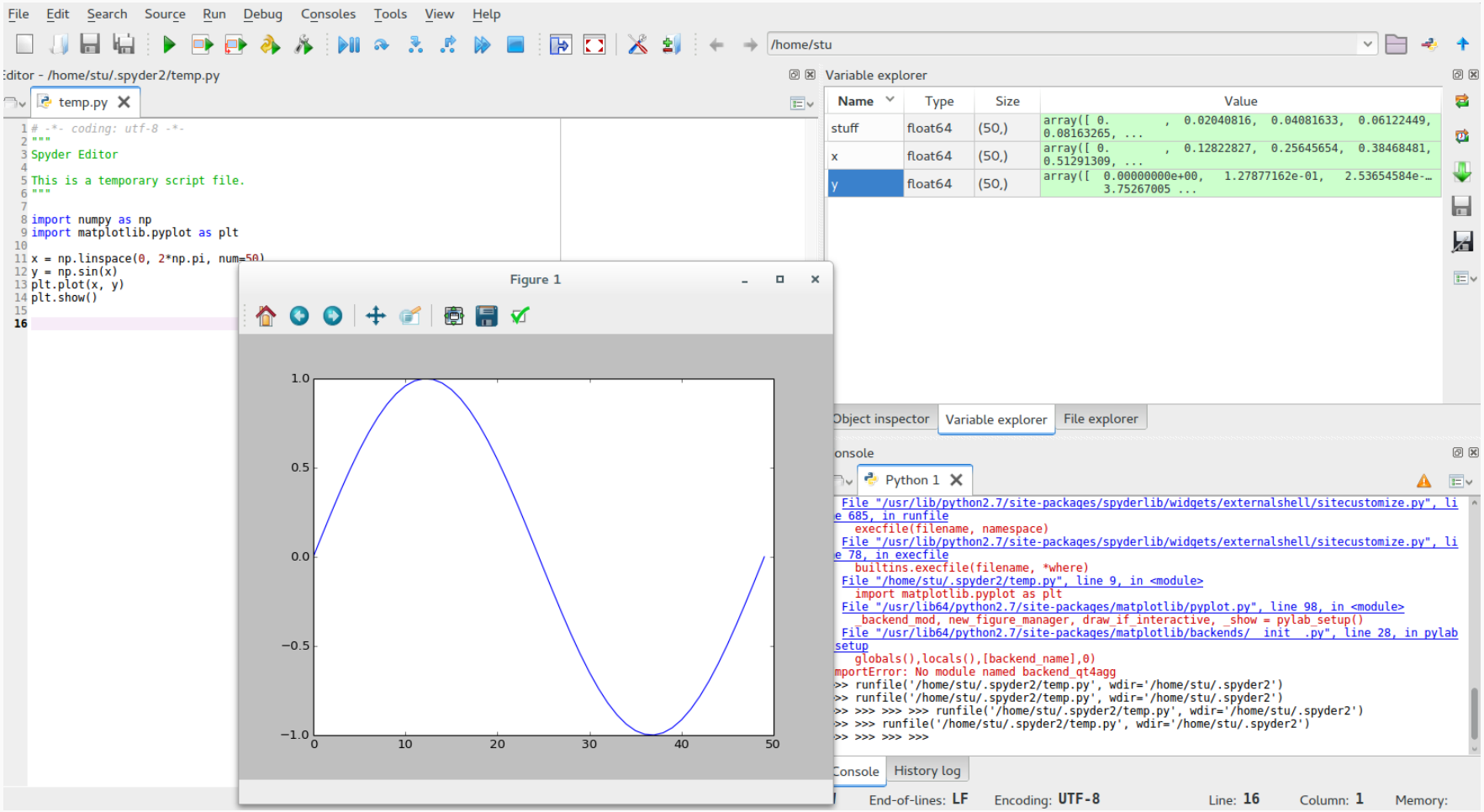
EDITOR CHOICE

- For quick scripts I personally use a text editor and run it manually
- Useful IDEs include:
  - PyCharm - My favourite, has full student licence
  - PyDev - Eclipse plugin
  - Spyder - Matlab style IDE, focuses on scientific computing
- IDLE - Comes with Python on Windows. **Rubbish**.

PYCHARM


File Edit View Navigate Code Refactor Run Tools VCS Window Help
Nostradamus nostradamus ea fitness.py
Debug
ea.py population.py fitness.py util.py selector.py breeder.py classifier_breeders.py diversity.py runtimes.cfg main.py
Structure
fitness.py
FitnessEvaluator(object)
__init__(self, num_classes=2)
calculate_fitness(self, outputs, cl
ind(self, classes, outputs_in=Non
predict_label(self, output, datase
num_classes
parsimony_coefficient
size_threshold
MaximiseValue(FitnessEvaluator)
maximise_objective
MinimiseValue(FitnessEvaluator)
maximise_objective
SingleOutputAUC(MaximiseValue)
__init__(self, *args, **kwargs)
calculate_fitness(self, outputs, cl
predict_label(self, output, datase
SingleOutputAccuracy(MaximiseValu
__init__(self, *args, **kwargs)
calculate_fitness(self, outputs, cl
predict_label(self, output, datase
class_thresholds
max_threshold
min_threshold
SingleOutputOptimumAccuracyFrom
__init__(self, *args, **kwargs)
calculate_fitness(self, outputs, cl
predict_label(self, output, datase
MultipleOutputAUC(MaximiseValue)
__init__(self, *args, **kwargs)
123:33 LF: UTF-8

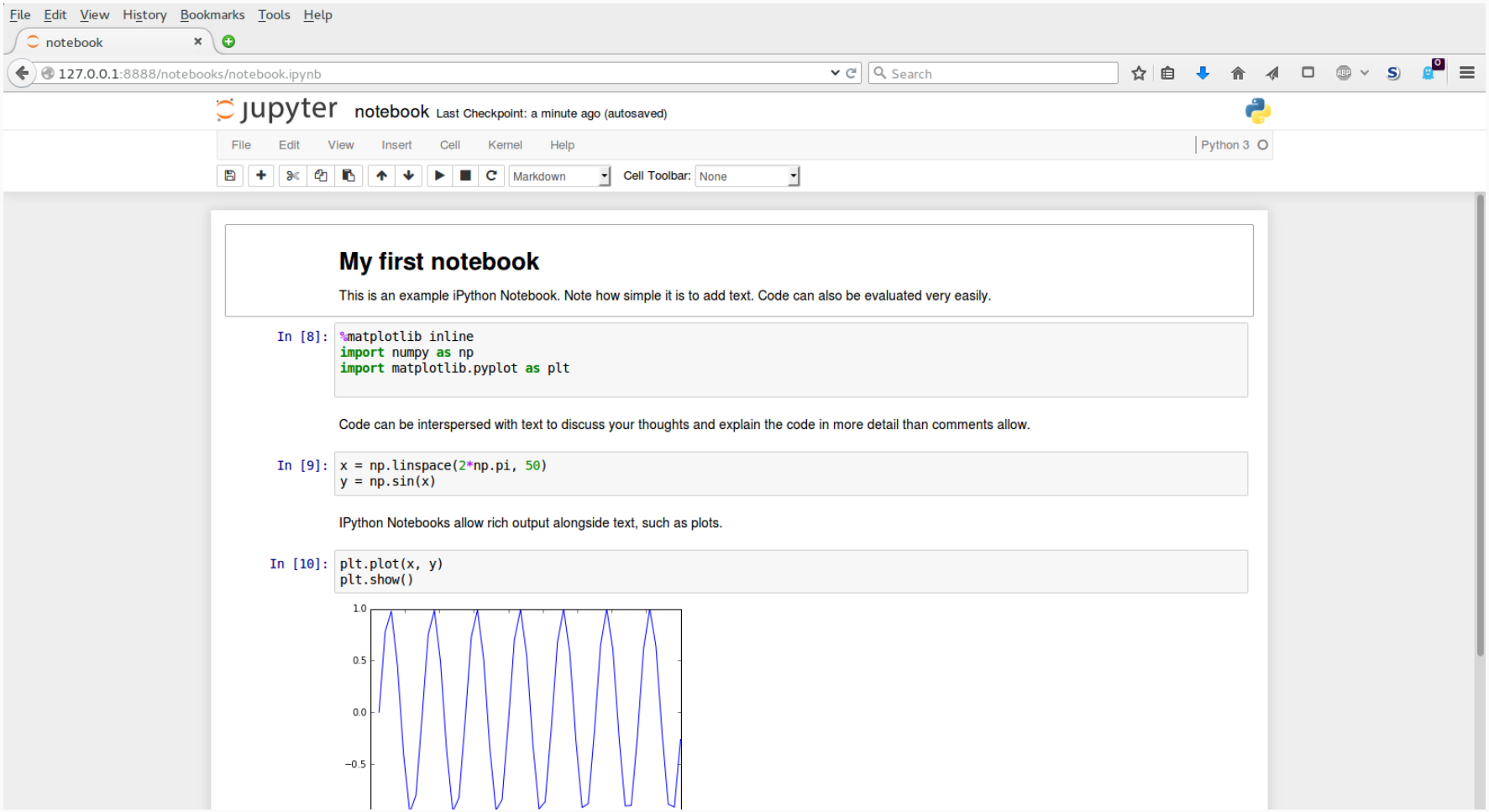
SPYDER


File Edit Search Source Run Debug Consoles Tools View Help
/home/stu
editor - /home/stu/.spyder2/temp.py
temp.py
-*- coding: utf-8 -*-
"""
Spyder Editor
This is a temporary script file.
"""
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, num=50)
y = np.sin(x)
plt.plot(x, y)
plt.show()
Figure 1
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Variable explorer
Name Type Size Value
stuff float64 (50,) array([0. , 0.02040816, 0.04081633, 0.06122449, 0.08163265, ...
x float64 (50,) array([0. , 0.12822827, 0.25645654, 0.38468481, 0.51291309, ...
y float64 (50,) array([0.00000000e+00, 1.27877162e-01, 2.53654584e-... 3.75267005 ...
Object inspector Variable explorer File explorer
onsole
Python 1
File "/usr/lib/python2.7/site-packages/spyderlib/widgets/externalshell/sitecustomize.py", li e 685, in runfile
execfile(filename, namespace)
File "/usr/lib/python2.7/site-packages/spyderlib/widgets/externalshell/sitecustomize.py", li e 78, in execfile
builtins.execfile(filename, *where)
File "/home/stu/.spyder2/temp.py", line 9, in <module>
import matplotlib.pyplot as plt
File "/usr/lib64/python2.7/site-packages/matplotlib/pyplot.py", line 98, in <module>
backend_mod, new_figure_manager, draw_if_interactive, _show = pylab_setup()
File "/usr/lib64/python2.7/site-packages/matplotlib/backends/__init__.py", line 28, in pylab setup
globals(),locals(),[backend_name],0)
mportError: No module named backend_qt4agg
>> runfile('/home/stu/.spyder2/temp.py', wdir='/home/stu/.spyder2')
>> runfile('/home/stu/.spyder2/temp.py', wdir='/home/stu/.spyder2')
>> >>> >>> >>> runfile('/home/stu/.spyder2/temp.py', wdir='/home/stu/.spyder2')
>> >>> runfile('/home/stu/.spyder2/temp.py', wdir='/home/stu/.spyder2')
>> >>> >>> >>>
Console History log
End-of-lines: LF Encoding: UTF-8 Line: 16 Column: 1 Memory:

IPYTHON NOTEBOOKS

- Multimedia environments with code, output, text and plots all alongside one another
- Very useful for scientific work
- Also very handy for sharing work with colleagues
- Can expand on ideas and explain implementation details in more depth than comments

IPYTHON NOTEBOOKS



Hello World

A QUICK NOTE ON MAIN

- You can see that unlike C or Java there is no main function
- Instead it runs the script entirely
- Useful but also means that when we import the script run into problems (discuss later)

Data Structures and Functions

PRIMITIVES

- Basic ints, floats, strings, booleans
- Variables are all references
- Variables can be used to reference different types

```
a = 5          # int
a = "foo"      # Previously referenced an int

b = 5.0        # float
c = "hello"    # string
d = True       # boolean
```

ITERATION

- Control flow in Python is denoted by whitespace.
- Python replaces {}s from Java with indentation (4 spaces)
- No parentheses around condition
- No end statement, just revert back to higher indentation level

WHILE LOOP

```
i = 0
while i < 10:
    print(i)
    i += 1
print("The last value is",i)
```

```
0
1
2
3
4
5
6
7
8
9
The last value is 10
```

FOR LOOP

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

CONDITIONALS

- If and Else statements are implemented in a similar fashion
- Else-if statements take the syntax `elif`
- Compound statements are formed with `and`, `or`

```
# Obtain a random number somehow
my_number = random_number()
if my_number > 1000:
    print("Number is greater than 1000!")
elif my_number < 10 and my_number % 2 == 0:
    print("Number is less than 10 and even")
else:
    print("Not interested in this number")
```

INTRODUCTION TO DATA STRUCTURES

- One of the main advantages of Python is the extremely powerful data structures it has
- These come in the default namespace and don't need to be imported
- Three main ones:
  - Lists
  - Tuples
  - Dictionaries

LISTS

- The most common data structure
- Heterogeneous dynamic array
- Heterogeneous = multiple different types
- Dynamic = non-fixed size
- Create lists with []s
- Use []s for indexing as well

LISTS

```
foo = [1, "hello", 3.4]  # Literal instantiation
foo.append(5)  # Can add items
foo.remove("hello")  # ... And remove them
foo[0]   # Evaluates to 1
foo[-1]  # Returns the last value in the list, 5


# Can check for membership with 'in'.
contains_1 = 1 in foo  # Evaluates to True
```

```
[1, 3.4, 5]
```

ITERABLE DATA STRUCTURE

- Any object that overrides the iterable property allows for direct iteration over (think for each loops in Java)
- This includes all Python data structures
- Remember the for loop example from before? `range(x)` is a function which returns a list

ITERABLE DATA STRUCTURE

- Any object that overrides the iterable property allows for direct iteration over (think for each loops in Java)
- This includes all Python data structures
- Remember the for loop example from before? `range(x)` is a function which returns a list

```
foo = range(5)  # [0, 1, 2, 3, 4]

# Therefore could have rewritten for loop
for i in foo:
    print(i)
```

Docs show full range of functions which work on lists
https://docs.python.org/3/tutorial/datastructures.html

LIST COMPREHENSIONS

- A common operation is to iterate through adding items to a list
- This is so common that Python designers decided to add syntactical sugar to reduce typing - list comprehension
- Can be used any time have iteration

LIST COMPREHENSIONS

```
# Creates a list of squared integers up to 10
foo = []
for i in range(10):
    foo.append(i**2)
```

LIST COMPREHENSIONS

```
# Creates a list of squared integers up to 10
foo = []
for i in range(10):
    foo.append(i**2)

# Exact same output as above
foo = [i**2 for i in range(10)]
```

LIST COMPREHENSIONS

```
# Creates a list of squared integers up to 10
foo = []
for i in range(10):
   foo.append(i**2)

# Exact same output as above
foo = [i**2 for i in range(10)]

# Can be used for any iterable object

# Creates a list of lines in a text file
# Python will automatically detect line breaks
myfile = open('testfile.txt', 'r')
lines = [line for line in myfile]
```

TUPLES

- Heterogeneous fixed-size arrays
- Uses less memory and bit quicker to access than lists
- Useful for data structures such as Java classes which just contain fields
- Denoted by parentheses

TUPLES

```
foo = (1, "hello", 3.4)

# Can't add or remove items

# Iterable in the same fashion as lists
for item in foo:
    print(item)
```

DICTIONARIES

- Hash map / associative array
- Indexes items in a container by a hash (typically a string)
- Create with {}s in the form
  `{<key1>: <value1>, <key2>: <value2>... }`
- Iterating returns the keys

DICTIONARIES

```
# Dictionary instantiation
foo = {'cat': 'miaow', 'dog': 'woof'}

# Index with squared brackets
cat_noise = foo['cat']  # 'miaow'
dog_noise = foo['dog']  # 'woof'

# Add new values in the same way
foo['pig'] = 'oink'

# Prints dog, cat, pig
for item in foo:
    print(item)
```

COUNTING EXAMPLE

```
# A useful application is counting items

text = ("There", "are", "only", "ten", "types", "of" "people",
        "in", "the", "world", ",", "those", "that", "understand",
        "binary" "and" "those" "that" "don't")
counts = {}
for word in text:
    if word not in counts:
        counts[word] = 1
    else:
        counts[word] += 1
```

```
{'ten': 1, 'types': 1, 'people': 1, 'in': 1, 'the': 1, 'those': 2,
 'binary': 1, 'world': 1, ',': 1, 'only': 1, 'are': 1, "don't": 1,
 'and': 1, 'understand': 1, 'of': 1, 'that': 2, 'There': 1}
```

FUNCTIONS OVERVIEW

- Differences with Java functions:
  - No notion of public/private
  - Don't state argument type
  - Don't state return type
  - Function body defined by indentation rather than {}s
  - def is the keyword which creates a function

```
def function_name(arg1, arg2):
    code here
    ...
    more code
    return value

new_value = function_name(val1, val2)
```

EXAMPLE FUNCTION

- Will calculate mean of a list
- Will use the inbuilt functions sum and len

KEYWORD ARGUMENTS

- Specify arguments by name rather than position
- Allows for optional arguments to set default values
- Makes code self-documenting

```
# How readable is this?
my_plot(x, y, True, False, True, 'Time',
        'Amplitude')
```

KEYWORD ARGUMENTS

- Specify arguments by name rather than position
- Allows for optional arguments to set default values
- Makes code self-documenting

```
# How readable is this?
my_plot(x, y, True, False, True, 'Time',
        'Amplitude')

# Now you can see what each argument does
my_plot(x, y, show_gridlines=True, save=False,
        x_label='Time', y_label='Amplitude')
```

KEYWORD ARGUMENTS

```
# Here's how you define a function with keyword args
def run_ea(population, data, generations=1000):
    for i in range(generations):
        outputs = run_population(population)
        fitnesses = fitness_function(outputs)
        population = breed(population, fitnesses)
    return population.winner

# Can omit keyword args in the call to use
# the default value
run_ea(my_pop, some_data)

# Or pass in a value
run_ea(my_pop, some_data, generations=5000)
```

IMPORTING MODULES

- Can reuse code we've already in other modules with import statements
- Same syntax for using library code
- Have to be careful with top level functions!

```
import foo

foo.some_function()

# OR can use aliases for long module names
import somereallylongmodulename as m

m.some_function()
```

MAIN FUNCTIONS

- Importing a Python module runs all code at the highest indent level
- Can lead to unexpected behaviour
- Want to be able to control the entry point into the program using a main function like other languages
- In Python this is achieved with the following

```
if __name__ == "__main__":
    # Code here will only be run when this module
    # is executed directly with
    # 'python modulename.py',
    # and *not* when imported
    do_stuff()
```

Replacing MATLAB with Python

DIFFERENCES FROM MATLAB

- Matlab is designed for Scientific Programming, Python is a general purpose programming language
- Matlab comes with an IDE, Python is just the language
- Python is fully cross-platform compatible
- Python has more sane syntax choices, such as indexing with square brackets and 0-based indices
- String manipulation is far easier in Python than in Matlab
- Python has a much better modular system

TURNING PYTHON INTO A MATLAB COMPETITOR

- The Python language is great for general programming uses but by itself it lacks little support for scientific programming
- It needs:
  - Quick data structures (static arrays)
  - Vectorized operations
  - Library of Scientific functions

TURNING PYTHON INTO A MATLAB COMPETITOR

- The Python language is great for general programming uses but by itself it lacks little support for scientific programming
- It needs:
  - Quick data structures (static arrays)
  - Vectorized operations
  - Library of Scientific functions
- The 3rd party libraries in the SciPy stack solve this!
- Three packages which are often included together
  - NumPy
  - SciPy
  - Matplotlib

NUMPY

- Lists are dynamic sized and heterogeneous (memory and slow to access)
- Tuples are heterogeneous (memory)
- NumPy provides fixed size homogeneous C-style arrays
- And vectorized operations like Matlab (can vectorise your own code)

NUMPY

- Lists are dynamic sized and heterogeneous (memory and slow to access)
- Tuples are heterogeneous (memory)
- NumPy provides fixed size homogeneous C-style arrays
- And vectorized operations like Matlab (can vectorise your own code)

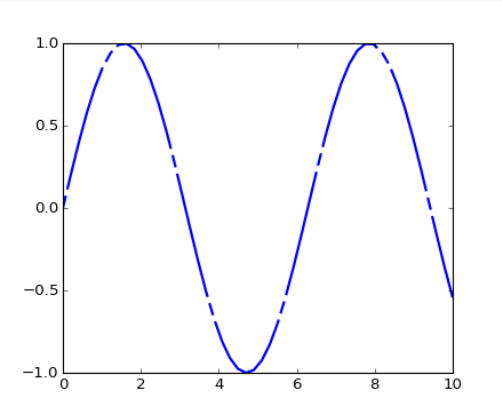
```
import numpy as np

# Creates an array of 60 values between 0 and 2*PI
x = np.linspace(0, 2*np.pi, 60)

# Calculates the corresponding sine values
y = np.sin(x)
```

SCIPY

- Built on the top of NumPy
- Complex numbers
- Signal processing
- Polynomials
- Integration
- Statistics
- Most functionality from all Matlab toolboxes (even paid ones) except Simulink
- Look on website to see the full range of functionality

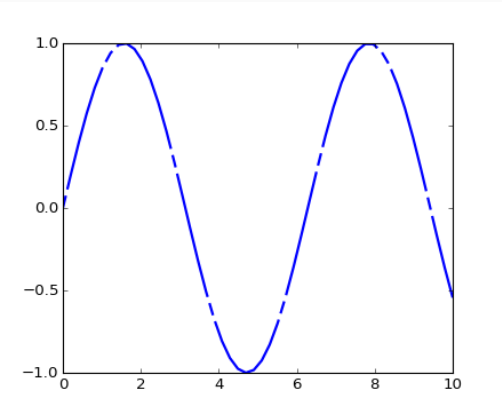
MATPLOTLIB

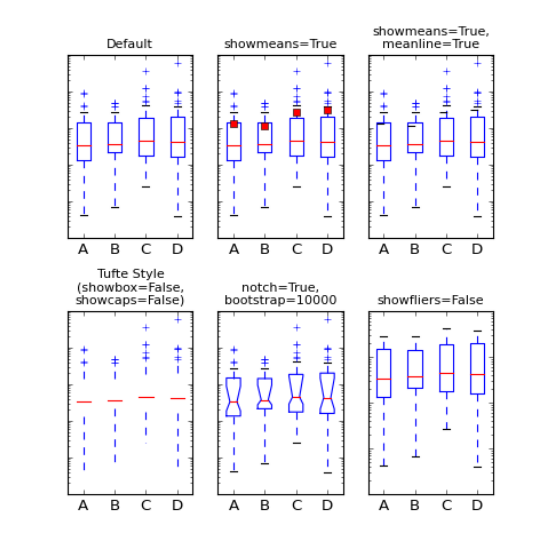
- Easy to use plotting interface
- Mimics Matlab
- Images taken from www.matplotlib.org where documentation is also kept



MATPLOTLIB

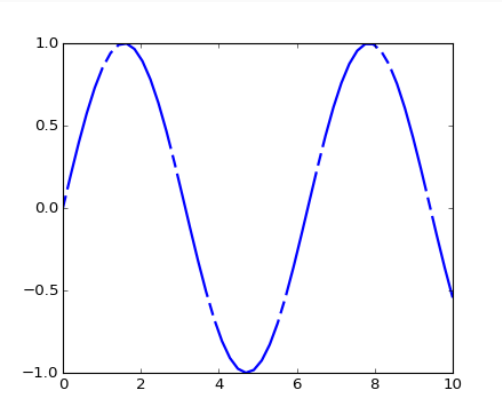
- Easy to use plotting interface
- Mimics Matlab
- Images taken from www.matplotlib.org where documentation is also kept

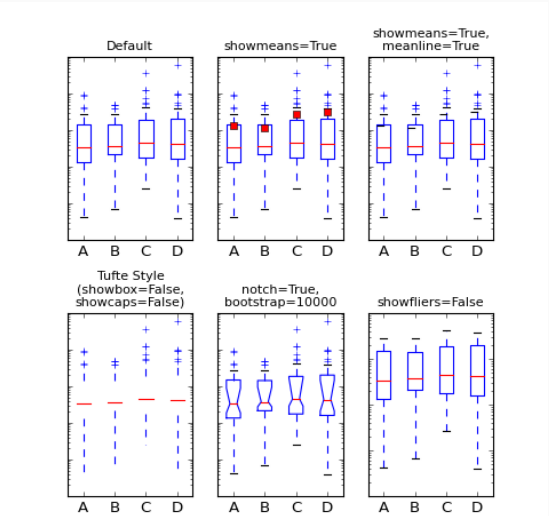


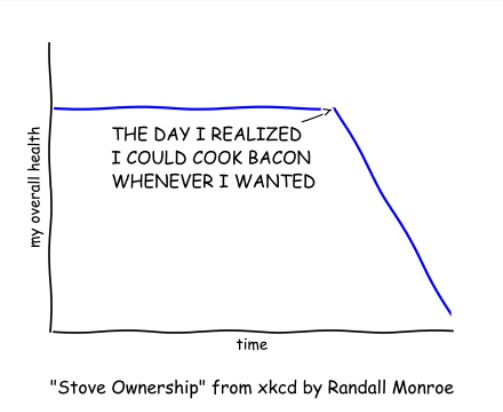


MATPLOTLIB

- Easy to use plotting interface
- Mimics Matlab
- Images taken from www.matplotlib.org where documentation is also kept







SCI-KIT LEARN

- Machine learning in Python
- Classification and regression
- Covers feature selection and resampling as well
- Model examples:
  - SVMs
  - Decision trees
  - Neural networks
  - Ensembles

PANDAS

- Data analysis package - very useful
- Introduces the data frame data structure from R
- Access columns by name, summary measures easily available
- Plays well with the rest of the SciPy stack

SYNTAX

```
    mutation_rate  run  accuracy
0            0.04    1      0.93
1            0.04    2      0.92
2            0.04    3      0.91
3            0.08    1      0.94
4            0.08    2      0.93
5            0.08    3      0.92
6            0.12    1      0.90
7            0.12    2      0.89
8            0.12    3      0.88
```

```python
import pandas as pd
import numpy as np

# Can access columns by name
data = pd.read_csv('dummy_data.csv')
grouped = data.groupby('mutation_rate')
summaries = grouped['accuracy'].agg([np.mean, np.std])
```

```
              mean   std
mutation_rate
0.04          0.92  0.01
0.08          0.93  0.01
0.12          0.89  0.01
```

SPEEDING UP CODE

- If Numpy wasn't sufficient or applicable there are a few other options
- Cython (www.cython.org)
  - Compiles to C
  - Can run straight away (after compiling) or can get far better speed ups by adding types

- If Numpy wasn't sufficient or applicable there are a few other options
- Cython (www.cython.org)
  - Compiles to C
  - Can run straight away (after compiling) or can get far better speed ups by adding types
- PyPy (www.pypy.org)
  - JIT compiler for Python
  - My tool of choice for speeding up code
  - Don't need to do anything special, just install it and run pypy foo.py rather than python foo.py
  - It can run every standard library but not necessarily third party ones, particularly those that have a lot written in C
  - I.e. No PyPy + Numpy combo...yet

Object-Oriented Programming in Python

OVERVIEW

- Duck typing
- If it looks like a duck and quacks like a duck, it's probably a duck
- I.e. as it's statically typed we can pass any object into a function, as long as it can run the method require it's fine
- Less verbose code, but not checked at compile time

```
class Foo:
    # Constructor is a special method
    def __init__(self, x):
        self.x = x
        self._y = x * 20  #

class Bar(Foo):  # Declare superclass after the name
    pass  # Pass is used to provide an empty class or method body

my_foo = Foo(5)
print(my_foo.x)  # Attributes are always public

# Nothing is preventing me from accessing this private attribute
# although I should be aware that I'm not using the code as
# intended
print(my_foo._y)
```

SYNTAX

- Instance methods must explicitly call self as the first argument
- Referring to an instance attribute must be prepended with self
- All attributes and methods are public and cannot be made private
- The convention is to prepend ‘private’ attributes and methods with an underscore, so that the programmer is aware that the particular object should only be accessed within that class
- Some built-in methods are pre and post pended with two underscores, indicating that you are overriding a **special** method

DIFFERENCES FROM JAVA

- Not obliged to use OO
- No interfaces
- Multiple inheritance
- Classes are first order objects, can pass them around

ERROR HANDLING

- Extremely similar to Java, but different keywords
- Exceptions get raised up the call stack, don't need to keep explicitly try to catch the error at every level
- try-and-except and throwing errors

```
try:
    my_num = int(input("Please enter a number: "))
# If the user enters a string the cast will fail
except ValueError:
    print("That wasn't a valid number!")
```

STANDARD LIBRARIES

- sys, os, and shutil: Allow interactions with the file system, the operating system and run external programs in a shell.
- collections: Additional useful data structures
- argparse: Easily parse arguments to make a fully functional command line program
- sqlite3: Access basic SQL databases.
- urllib: Access websites from within your programs.

OTHER REFERENCES

- General Python support:
  - Official Python Tutorial
  - Learn Python the Hard Way - Teaches general programming concepts as well as Python
  - Dive into Python - Written for experienced programmers
- Scientific libraries support:
  - numpy.org/
  - docs.scipy.org/doc/scipy-0.15.1/reference/
  - matplotlib.org
  - scikit-learn.org/stable/
  - pandas.pydata.org/

Thanks for listening!