Python for Informatics

LESSON 1

Course Orientation

- Email: All email should be sent via the Blackboard email facility.
- Please indicate “Python for Informatics” in the subject line, otherwise I won’t know which one of my courses you are referring to.
- Contact by email is preferred, however, if and when it is necessary the instructor can be contacted by phone (see syllabus).

Course Orientation

- All projects will be posted on the Blackboard web site. See syllabus for schedule.
- Students are responsible for periodically logging on to the Blackboard web site.
- Completed projects must be submitted via the Blackboard web site.
- The instructor will not be able to rigorously debug student projects. However, if a code error has a student stumped, then the error message and the associated source code should be copy-and-pasted and posted on the course discussion list.

Introduction

- A *program* or *algorithm* is like a *recipe*.
- Fundamentally, computers are *encoding/decoding machines*.
- Everything that a computer represents, whether it be text, numbers, graphics, sound, metadata, or the signal radio source SHGb02+14a, is an encoding of a series of ones and zeroes.

Introduction

- The ones and zeroes within a computer are physically rendered as voltage levels on wires and electro-magnetic fluctuations within memory.
- Encoding schemes can be layered on top of encoding schemes up top of…

Introduction

- What is Python?
  - Python is a highly flexible and dynamic object-oriented programming language developed by Guido van Rossum in 1990.
  - Python offers a wealth of extension modules that enable the programming to quickly develop software by leveraging freely accessible software that is domain-specific and reusable.

Introduction

- Python is used by companies like Google, Industrial Light & Magic, Nextel, Kodi, and others.
- Python is powerful, yet relatively easy to learn.
- Python is a great first language for anyone learning to program for the first time.

Introduction

- Varieties of Python – There are three major implementations of Python.
  - CPython
  - IronPython
  - Jython

Introduction

- What is CPython?
  - CPython is the “Classic” Python is an implementation of Python written in the standard C language. Accordingly, it is the fastest implementation, while still assuring portability across all major platforms.

Introduction

- What is IronPython?
  - IronPython is an implementation of Python that on the Common Language Runtime (CLR) virtual machine shared by the Microsoft .NET languages. If you want to code in Python as well as utilize the .NET Framework, then IronPython is an excellent choice.

Introduction

- What is Jython?
  - Jython is an implementation of Python that is written in the Java language. It will run on the Java Virtual Machine (JVM), and thereby enjoys broad portability across all major platforms. Since Jython is written in Java, it can also easily utilize Java packages.

Why Python?

- Why not some other language?

- Alternative languages to consider:
  - R
  - SAS
  - SQL

Why not R?

- R could actually be a very good choice!
- Strengths of R
  - Functionality is roughly equivalent to Python, with differences being syntactical and presentational.
  - R includes its own Integrated Development Environment (IDE) as a standard component of the language (whereas with Python, you need to employ third-party IDEs if you want to work outside of the command line window).
- Weakness of R
  - Utilizing other languages within R is not well supported.

Why not SAS?

- SAS could actually be a very good choice!
- Strengths of SAS – Offers strong support for…
  - Data Management
  - Data Analysis
  - Business Intelligence
  - Predictive Analytics
- Weakness of SAS
  - For non-business fields such as scientific investigation, SAS is not particularly well-suited.
  - Expensive! SAS is a proprietary product, and is therefore not a practical choice for start-ups or research teams operating on a small budget.

Why not SQL?

- SQL (Structured Query Language) could actually be a very good choice!
- Strengths of SQL
  - Data Management
  - Ad-hoc queries of highly structured data
  - Fast
- Weakness of SQL
  - Non-structured data sources are difficult to integrate.
  - While R and Python can easily make use of SQL, SQL cannot easily make use of R or Python.
  - Strictly speaking, SQL is not a full computer language, and thereby does not have the expressiveness of a full language.

Why Python?

- Strengths
  - Flexible
  - Dynamic
  - Easily utilizes other languages such as R and SQL
  - Wide availability of software modules, providing a established treasure trove of prewritten, extensively tested, and free software
  - New modules addressing new and emerging domains are continually being developed.
  - Works with a variety of third-party software libraries

Why Python?

- Supports Multiple ***Programming Paradigms***
  - **Procedural**
    - Data centric approach using step-wise sequential processing, with selection and iteration constructs, and procedure blocks – tends to promote centralized architectures
  - **Functional**
    - Based on Alonzo Church's *Lambda Calculus*, this approach functional abstraction as the primary language unit – scope of data is elegantly managed
  - **Imperative**
    - Focuses on the direct transformation of program state
  - **Object-Oriented**
    - Uses encapsulation of data and methods, inheritance, and polymorphism to support extensible and reusable code that can be effective in managing complexity and change

What's with the Name?

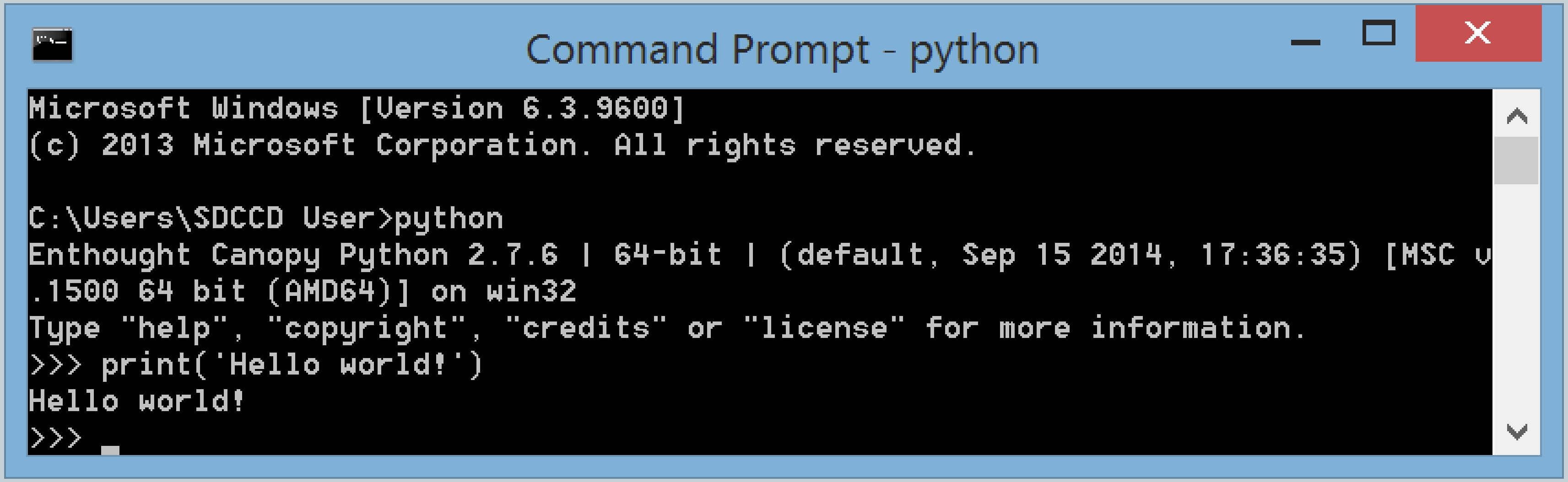
- The Python language was created by a guy named Guido.
- He was (is) a fan of the British comedy troupe *Monty Python*. He named his language in "honor" of them.
- Guido van Rossum created Python in 1989 as an improvement upon the ABC language.

The One Humongous Wart!

- Python has a big, ugly wart.
- The are two version branches of Python: 2.x and 3.x.
- Version 3.x is the most recent version.
- However, 3.x is ***not* backward compatible** with 2.x!
- Moreover, most informatics and data analytics libraries are based on version 2.x.
- Therefore, ***we will be using version 2.x for this course***.

The Command Line, IDLE, and IPython

- The most rudimentary means of working with Python is via the command line. Just type "python"…



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\SDCCD User>python
Enthought Canopy Python 2.7.6 | 64-bit | (default, Sep 15 2014, 17:36:35) [MSC v
.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello world!')
Hello world!
>>> _
```

- Then by typing `print("Hello world!")`, you will have written the simplest program in Python.

The Command Line, IDLE, and IPython

- A step up from the command line is an editor with special support for Python syntax, format, and presentation.
- A popular example of such an editor or Integrated Development Environment (IDE) is the IDLE application.
- The name IDLE is a reference to Eric Idle, a member of the *Monty Python* comedy troupe.

The Command Line, IDLE, and IPython

- A step up from an IDE such as IDLE, is IPython.
- IPython is a feature-rich Python development environment that makes programming in Python much easier. Think of it as IDLE on steroids!

The Command Line, IDLE, and IPython

- IPython can also facilitate the learning of the Python language
- For these reasons, ***we will be using IPython for this course***.

Canopy vs Anaconda

- Canopy and Anaconda are the two major IPython IDEs.
- Both of them are free and excellent!
- Both of them have great community support.

Canopy vs Anaconda

- Canopy boasts having a community that provides a variety of strong, academic resources.
- Therefore, ***we will be using the Canopy IDE in this course***.
- While Canopy is free for academic purposes (that means you!), you must buy a license for any commercial use.

Jupyter?

- ***Jupyter*** is latest, greatest thing in IPython-like IDEs. It is still relatively new.
- However, Jupyter does not operate quite the same as IPython.

Jupyter?

- Feel free to experiment with Jupyter at your own risk!
- To serve as a stable and common basis of our learning Python, ***we will be using the Canopy IDE to interact with IPython***.

The IPython Notebook

- One of the coolest and most powerful features of IPython is the IPython Notebook.
- IPython Notebook is a way of capturing IPython programming sessions in a special format that can be saved and exchanged with colleagues.

The IPython Notebook

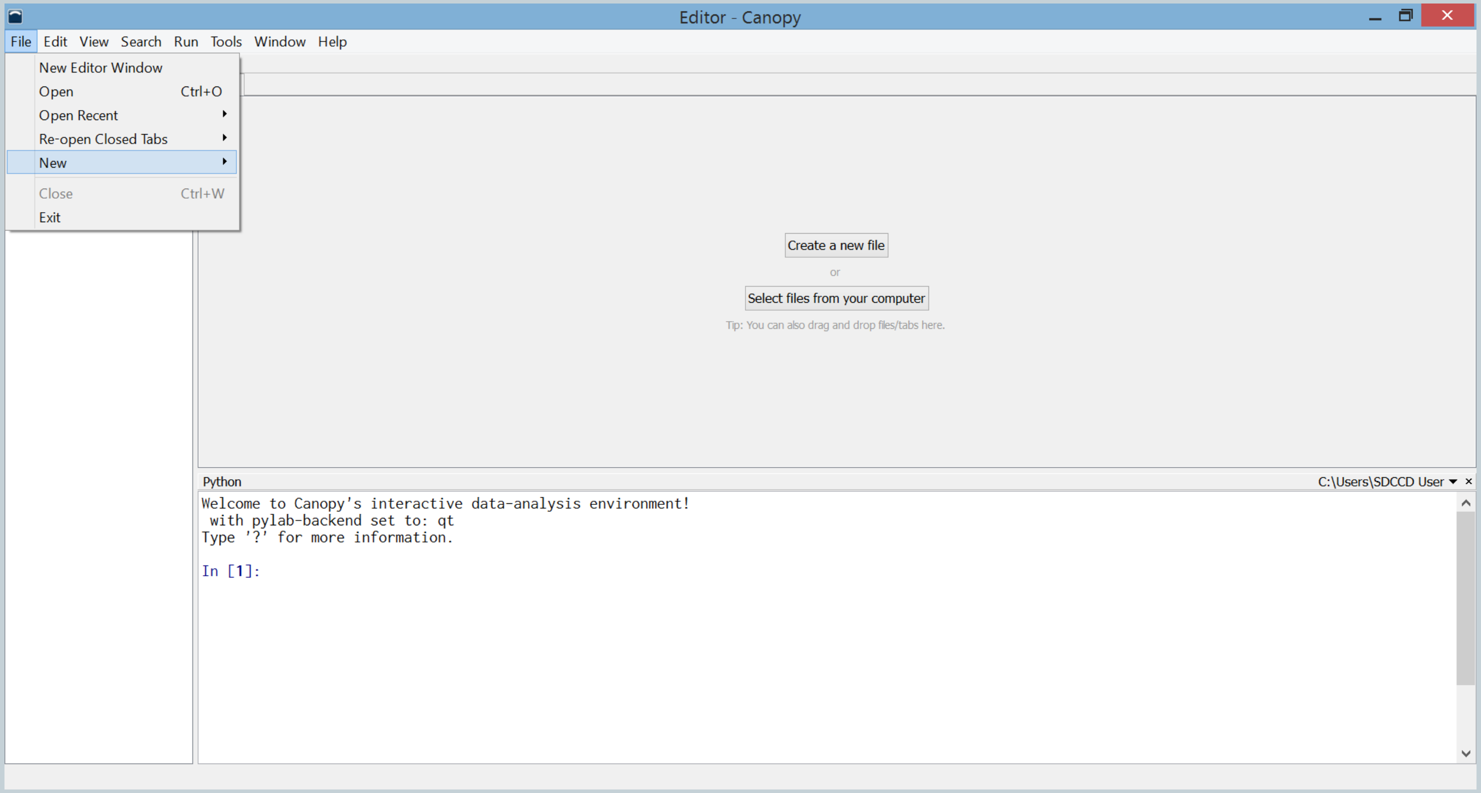
- IPython Notebooks are HTML-based, and are displayed and executed within a standard HTML browser.
- IPython Notebooks are dynamic! Not only can your colleagues see your IPython session, they can also tweak your session with their own input parameters and code statements. These are then executed in their browser!

The Spirit of the Internet

- Since a primary motivation for the creation of Internet was to enable the sharing of information in research and academia, ***the spirit of the Internet is alive with IPython Notebooks***!

Creating an IPython Notebook

- Creating an IPython Notebook is easy.
- From the *File* menu, select *New*, *IPython Notebook*



```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]:
```

Using an IPython Notebook

- Notice that there are two major Canopy panes that are available to you when you are working with a notebook.
- The large, upper pane is the notebook editor pane—this is your view of the notebook within Canopy. It is what you see and use to create your notebook.

Using an IPython Notebook

- The large, lower pane is the interactive, command line Python pane.
- The command line pane allows you to experiment with Python commands outside of the notebook session.

Using an IPython Notebook

- Within the Canopy notebook pane, you can build your IPython Notebook step by step, one *cell* at a time.
- IPython *cells* are listed vertically in the order that you enter them.

Using an IPython Notebook

- Each IPython code *cell* is composed of input/output element pairs. The input element of the cell accepts input from the user, and the output element displays the result of evaluating and executing the input element.
- *Cells* are numbered to reflect the order of entry.

Using an IPython Notebook

- The Notebook Editor has a *cell toolbar* that provides quick access icons for performing cell editing operations.
- Cell editing operations include:
  - Copying
  - Pasting
  - Cutting
  - Inserting
  - Moving (up or down)
  - Running (executing the selected cell)
  - Interrupting (the running of the notebook)
  - Restarting (the Kernel—this will clear all variable values)

Using an IPython Notebook

- These same cell editing operations can be selected through the set of menus at the top of the editor pane.
- The cell toolbar has a combo box that is set to “Code” by default. This indicates that your current entry mode is for creating “code” cells.

Using an IPython Notebook

- Instead, you can select from a variety of "heading" modes so you can create "heading" cells.
- Any text you enter while in a heading mode will display that text in a heading cell with the associated heading format.

Printing Values

- Within either the notebook editor pane or the command line pane, type the following:

```
print(5)
```

- You should see an *integer* output value of *5*.
- Now let's print a float value:

```
print(1.87)
```

- You should see a *float* output value of *1.87*.

Printing Values

- Now print a series of characters, known as a "string" or of type *str*:

  print('Hello world!')

- You should see an output value of *Hello world!*

Printing Expressions

- Expressions are created by specify one or more operations that be performed or *evaluated*.
- When an expression is evaluated, we say that it *resolves to a value*.

Printing Expressions

- The resolved value of an expression can then be used as a part of a surrounding statement.
- For example, if you type the following:

  ```
  print(1 + 0.87)
  ```

- You should see an evaluated *float* value of *1.87*.

Working with Variables

- There are many circumstances in programming, wherein we need to hold onto a value for later use.
- Often when we refer back to a value, when also want to update or change its value for some purpose.
- It is for these reasons that we benefit from the use of *variables*.
- ***Variables are named recepticles that hold values that may vary.***

Would you like a box for that?

- Most of us hear this question almost any time that we are at a restaurant and are unable to finish our meal.
- Boxes are very convenient receptacles for things that we are done with for now, but will likely want to refer back to later.

Would you like a box for that?

- The analogy of a box is an excellent visualization of a program *variable*.
- *Think of a variable as a box in memory that contains whatever value you put into it.*
- You can always overwrite a previous value with a new value.

Types and Encoding

- What's a *data type*?
  - The concept of a *data type* is fundamental to understanding how to program.
  - There are *two important aspects* to any given data type
    - *Size* – How much memory is set aside (allocated) to be used by the data of that data type?
    - *Encoding Scheme* – How will the bits in memory be interpreted based on that data type?

Types and Encoding

- Whereas *size* deals with the implementation details of *how much space* is need to manage the data, the *encoding scheme* deals with the *semantics* (meaning) of the data residing in that space.

- Fortunately, with Python you rarely need to be concerned about the size of your data types. What really matters is what your objects *mean*.

Working with Integers vs Strings

- answer = 42

  answer becomes a name for the integer *42*.

- myJerseyNumber = '42'

  myJerseyNumber *becomes a name* for the string *"42"*

- *42* and *'42'* are ***not*** the same!

  The first is an integer number.

  The second is a string of two characters, a '4' followed by a '2'.

Working with Integers vs Strings

- *42* and *'42'* are examples of *literal values* (they are literally exactly what they say they are).

  The first is an integer that is literally *42*.

  The second is a string that is literally *'42'*

- A string can be specified using double OR single quotation marks.
  - i.e., *"42"* and *'42'* are the same.

Literal Values

- *42* and *'42'* are examples of *literal values* (they are literally exactly what they say they are).
  - The first is an integer that is literally *42*.
  - The second is a string that is literally *'42'*
- A *string* is a ***sequence*** of characters.

Literal Values

- As an analogy, envision a piece of twine upon which you thread a series of beads. Think of the beads as being individual characters—the whole strand of characters is a *string*.
- Here are some more examples of printing *string literals*:

```
        print "And now, for something completely different!"
```

...displays *And now, for something completely different!*

```
        print "42 / 5"
```

...displays *42 / 5*

Integer vs Floating Point Division

- Using Numeric Literals

  print 42 / 5

  …displays *8*, since the operands are *integer* values, integer division is performed. (This has changed with version 3, however, where floating point division is performed even with two integer operands.)

  print 42.0 / 5.0

  …displays *8.4*, since if one or both of the operands is a *floating point* value, floating point division is performed, and the result is a floating point value. Therefore,…

  print 42.0 / 5

  …displays *8.4*, since *42.0* is a floating point value

Mixed Expressions

- An expression is a combination of values, variables, and operations that are evaluated to yield a resulting value.
- *print 42.0 / 5* is an example of what is called a *mixed expression*, because the expression is composed of more than one data type. In this case, we have an expression of one *floating point number* (the *42.0*) and one *integer* (the *5*).

Mixed Expressions

- A data type that supports a larger range of values and/or more precise values is said to be *wider* than a (*narrower*) data type that supports a smaller range of value and/or less precision.
- When a *mixed numeric expression* is evaluated, the *narrower* data type value (in this case, *5*) is converted to be of the *wider* data type, and the evaluated value will have the size and type of the *wider* data type.

Strings

- *string concatenation*
  - Strings can be *concatenated* by using the *concatenation operator*.
  - The *concatenation operator* is the plus sign, “+”, and it operates upon two *string* operands.
  - *print "Python " + "rocks!"*

    ...displays *Python rocks!*.

Strings

- You can think of *concatenation* as the way that you “add” *string*s together.
- You can also think of *concatenation* as being like gluing two *string* together.
- What really happens when you *concatenate* two *strings* is that a new *string* is created– the new *string* represents a fusion of the two *string* operands.

Backslashes and Slashes

- In Python, *backslashes* are used with String literals to denote what is called an *escape sequence*. Special characters are designated as an escape sequence. Here are some examples:

\t	*tab*
\n	*newline*
\r	*carriage return*
\¨	*double quote*
\\	*backslash*

- Note that \\ is the way that you indicate a backslash within a String literal.

Backslashes and Slashes

- Alternatively, if you are trying to indicate a backslash character *as a delimiter within a filepath*, then you can *use the forward slash instead*.
- The following two examples are equivalent, though the second one is simpler and preferred:

```
String photoName =
"C:\\Pictures\\Vacation\\Mazatlan\\sunset.jpg";
String betterName = "C:/Pictures/Vacation/Mazatlan/sunset.jpg";
```

- Using the slash character in this way is *platform independent* (it works for Windows, OSX, and Linux).

Variables and the Power of Naming

- The Power of Naming
  - The power we attain through the naming of variables is but one example of a more *generalized power of naming*.
  - We can name variables, methods, classes, modules, packages, and programs—each one of these serves as an abstraction. *Naming is abstraction*.

Variables and the Power of Naming

- Much of Computer Science can be seen as the shrewd development and utilization of abstractions.
- *Abstractions help us manage complexity.*

Maintaining a Running Total

```
# This is an example of code that maintains a running total

total = 8

total = total + 4
print(total)
12

total += 3
print(total)
15
```