Python Big Picture

Some of these slides you will need to review on your own – primarily the python introductory slides.

numPy

- Offers Matlab-ish capabilities within Python
- Fast array operations
- 2D Arrays, multi-D arrays, linear algebra and more.

Tutorial:
- [http://www.scipy.org/Tentative_NumPy_Tutorial](http://www.scipy.org/Tentative_NumPy_Tutorial)

sciPy: Scientific Python

- Extends numPy
- Gathers a variety of high level science and engineering modules together:
  - stats: statistical functions
  - spatial: KD-trees, nearest neighbors, distance functions
  - interpolate: interpolation tools e.g. IDW, RBF
  - optimize: optimization algorithms including linear programming

pandas

- Python “Spreadsheets”
- Pandas uses a DataFrame object which can be thought of as a table of data
- Handles Time Series
- It was built by the finance sector to aid with data manipulation and data analysis
- It has loads of brilliant functions to dig into your data
- It has useful functions for reading and writing to file types such as csv (excel, google sheet)

matplotlib

- plotting library to make graphs.
- easily customized and produce publication quality plots
- Using the Matplotlib, NumPy and Pandas libraries together make data analysis much easier and reproducible than in Excel

Versions

- Version History
  - Python 0.9.0 (1991 first published version of code)
  - Python 1.x (1994 legacy)
  - Python 2.7.x (2000, list comprehensions, Haskell)
  - Python 3.2.x (3 branch started in 2008, remove redundancies in code, only one “obvious” way to do it)

- Developing environments:
  - IDLE (basic)
    - coded in 100% pure Python, using the tkinter GUI toolkit
    - cross-platform: works on Windows and Unix
  - Python shell window (a.k.a. interactive interpreter)
  - debugger (not complete, but you can do the basics, set breakpoints, view and step)
  - ipython, Spyder (Anaconda)
  - Eclipse module
Installing Python

- Easy to get and install for Win/Mac from (2.7)
  http://www.python.org
- Intro: Wikipedia’s Python
- We recommend Anaconda installation.
- When you submit

IDLE Development Environment

- Shell for interactive evaluation
- Text editor with color-coding and smart indenting for creating python files.
- Menu commands for changing system settings and running files.

Interpreter: On my Mac

- Type “python” to start interpreter
- Type CTRL-D to exit the interpreter
- Python evaluates all inputs dynamically

IDLE: Working with a file.py

- IDLE -
  1. File -> new window
  2. type commands in new window area
  3. save as “file name”.py (typical extension) – if you don’t you don’t see “colors” in IDLE – but programs still run.
  4. Select “Run module” (from menu)

Running Programs on UNIX

- `#!/path/to/python` (makes it runnable as an executable)

```python
(saffron:ingrid:1563) more
filename.py
#!/opt/local/bin/python
print "hello world"
print "here are the ten numbers from 0 to 9"
for i in range(10):
  print i
(saffron:ingrid:1562) filename.py
hello world
here are the ten numbers from 0 to 9
0 1 2 3 4 5 6 7 8 9 I'm done!
```

Anaconda’s Spyder Editor

- Debugger
- Help/Documentation easily accessible
import firstprog

Other IDE(s): Anaconda’s Spyder

- Why we recommend Anaconda Python?
  - We could use Python IDLE ... some code ahead is depicted using this interface ... BUT! Then
- Anaconda ...
  - Already has many packages installed
  - Has a Script Editor and Console Window
  - Allows for efficient debugging
  - Breakpoints, using Console
- Also has a Notebook feature

- Other IDEs – PyCharm, iPython Notebook. (PyCharm is also a good alternative).

Look at a sample of code...
( use your favorite development environment)

```python
x = 34 - 23  # A comment.
y = "Hello"  # Another one.
z = 3.45
if z == 3.45 or y == "Hello":
x = x + 1
    y = y + "World"  # String concat.
print x
print y
```

Enough to Understand the Code

- Assignment uses = and
- Comparison uses ==. # same gotcha as C++ (logic.py)
- For numbers + - / % are as expected.
  - Special use of + for string concatenation.
  - Special use of % for string formatting.
- Logical operators are words (and, or, not) not symbols (&&, ||, !).
- The basic printing command is “print.” (more later)
  - % and .format() # old schools
  - Fancier.
- First assignment to a variable will create it.
  - Variable types don’t need to be declared.
  - Python figures out the variable types on its own (inference).

Indentations
Weird Characters (how to see)
in vi in command mode (bottom : mode)
```bash
set list
```

Colors Please
• Review programming features of python
  – Slide 19-44. (on your own)
  – Understand python data structures, lists, dictionary etc.
• We will continue our review at:
  – for loop structure Slide 45. "More for Loops"

Basic Datatypes

• Integers (default for numbers)
  z = 5 / 2  # Answer is 2, integer division.
• Floats
  x = 3.456
• Strings
  Can use "" or ' ' to specify. "abc" 'abc' (Same thing.)
  Unmatched ones can occur within the string. "maria"
  Use triple double-quotes for multi-line strings or strings that contain both ' and " inside of them: """"a""b"c"""

Whitespace

• Whitespace is meaningful in Python: especially indentation and placement of newlines.
  – Use a newline to end a line of code.
    (Not a semicolon; like in C++ or Java.)
  – No braces {} are needed to mark blocks of code in Python...
    Use consistent indentation instead. The first line with a new indentation is considered outside of the block.
  – Often a colon : appears at the start of a new block. (We’ll see this later for function and class definitions.)

Comments #, """

• Start comments with # – the rest of line is ignored.
• A "documentation string" is often the first line of any new function or class that you define.
• The development environment, debugger, and other tools use it: it’s good style to include one.
  def my_function(x, y):
    """This is the docstring. This function does blah blah blah."""
    # The code would go here...
    x = y + 1
    return x

Look at more sample of code...

```python
x = 34 - 23  # A comment.
y = "hello"  # Another one.
z = 3.45
if z >= 3.45 or y == "Hello":
x = x + 1
y = y + " World"  # String concat.
print x
print y
```

Python and Types

Python determines the data types in a program automatically at runtime. "Dynamic Typing"

But Python is not casual about types, it enforces them after it figures them out. "Strong Typing"

So, for example, you can’t just append an integer to a string. You must first convert the integer to a string itself.

```python
x = "the answer is "  # Decides x is string.
y = 23  # Decides y is integer.
print x + y  # Python will complain about this.
```

```
Traceback (most recent call last):
  File "example.py", line 1, in <module>
    print x + y
TypeError: can only concatenate str (not "int") to str
```
Naming Rules

- Names are case sensitive and cannot start with a number. They can contain letters, numbers, and underscores.
  
  \[
  \text{bob} \quad \text{Bob} \quad \_\text{bob} \quad \_2\text{bob} \quad \text{BoB}
  \]
- There are some reserved words (good to memorize some of these):
  
  and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Accessing Non-existent Name

- If you try to access a name before it’s been properly created (by placing it on the left side of an assignment), you’ll get an error.
  
  ```
  >>> y
  Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
  NameError: name 'y' is not defined
  >>> y = 3
  >>> y
  3
  ```

Multiple Assignment

- You can also assign to multiple names at the same time.
  
  ```
  >>> x, y = 2, 3
  >>> x
  2
  >>> y
  3
  ```

String Operations

- We can use some methods built-in to the string data type to perform some formatting operations on strings.
  
  ```
  >>> "hello".upper()
  'HELLO'
  ```
- There are many other handy string operations available. Check the Python documentation for more.

Printing with Python

- You can print a string to the screen using "print."
- Using the % string operator in combination with the print command, we can format our output text.
  
  ```
  >>> print "%s xyz %d" % ("abc", 34)
  abc xyz 34
  
  "$xyz$" automatically adds a newline to the end of the string. If you include a list of strings separated by a comma [,], it will concatenate them with a space between them.
  
  >>> print "%s", "$abc $def"
  abc def
  ```

Strings

- Concatenation
  
  ```
  >>> "Hello" + " World"
  'Hello World'
  ```
- Repetition
  
  ```
  >>> "UGA" * 3
  'UGAUGAUGA'
  ```
- Indexing
  
  ```
  >>> "UGA"[0]
  'U'
  ```
- Slicing
  
  ```
  >>> "UGA"[1 : 3]
  'GA'
  >>> "UGA"[1 : 1]
  ''
  ```
  
  - \textbf{Looking ahead:} General method to get "subsets" of string, and also - list, tuple, array or custom data structures
  
  ```
  >>> theList = [0, 1, 2, 3, 4, 6, 8, 9, 10]
  >>> print theList[0:9:2]
  ```
  
  ```
  Slicing:
  ```
Strings

- Concatenation
  "Hello" + "World" -> 'HelloWorld'
- Repetition
  "UGA" * 3 -> 'UGAUGAUGA'
- Indexing
  "UGA"[0] -> 'U'
- Slicing
  "UGA"[1:3] -> 'GA'
- Size
  len("UGA") -> 3
- Comparison
  "Maria" < "maria" -> True
- Search
  "i" in "maria" -> True

Container Types

- Tuple
  ( 100, 200, 300 ) # Tuple
- List
  [ 42, 3.14, "hello" ] # List
- Dictionary
  { 'x':42, 'y':3.14 } # Dictionary

Tuple
- a simple immutable ordered sequence of items

List
- a mutable ordered sequence with more powerful manipulations

Dictionary
- a lookup table of key-value pairs

Lists

>>> alist = [631, "maria", [331, "maria"]]
>>> print alist
[123, 'maria', [331, 'maria']]

- List items need not have the same type
- Same operators as for strings
- operations append(), insert(), pop(), reverse() and sort()

More List Operations

>>> a = range(5)  # [0,1,2,3,4]
>>> a.append(5)  # [0,1,2,3,4,5]
>>> a.pop()      # [0,1,2,3,4]
>>> a.insert(0, 42)  # [42,0,1,2,3,4]
>>> a.pop(0)  # [1,2,3,4]
>>> a.reverse()  # [4,3,2,1,0]
>>> a.sort()    # [0,1,2,3,4]
>>> a.append([22,33])  # [0,1,2,3,4,[22,33]]
>>> a.extend([10,20])  # [0,1,2,3,4,[22,33],10,20]

More Lists

- List multiplication
  - list = ["aa", "bb"] * 3
- Printing out lists
  - print "\n".join(list) # better formatting
- More operations
  - list.count("aa") # how many times
  - list.index("bb") # returns the first match location
- More on slices
  - list[-1] # last element
  - list[0:3] # starting ele 0 and up to 2
  - list[3:] # starting ele 3 to end of list
  - list[:] # a complete copy of the list

Dictionaries

- Hash tables, "associative arrays" with key/value pairs
  - d = {"duck": "bird", "bee": "insect"}
- Lookup:
  - d["duck"] # "bird"
  - d["lion"] # raises KeyError exception
  - d["bird"]
- Delete, insert, overwrite:
  - del d["bee"] # delete
  - d["lion"] = "cat" # insert
  - d["duck"] = "unknown" # overwrites
More Dictionary Ops

- **Keys, values, items:**
  - `d.keys()` # returns dictionary keys
  - `d.values()` # returns all values
  - `d.items()` # returns a list of key/value pairs

- **Presence check:**
  - `d.has_key("duck")` # True
  - `d.has_key("spam")` # False

- **Values of any type**
- **Keys almost any type (needs to be immutable – tuples OK, but not lists).**

```python
{
    "name": "Maria",
    "age": 25,
    "flag": ["red", "white", "blue"]
}
```

Dictionary Details

- **Keys must be immutable:**
  - numbers, strings, tuples of immutables
  - these cannot be changed after creation
  - Keys are hashed [fast lookup technique]

- **not lists or other dictionaries**
  - these types of objects can be changed “in place”

- **no restrictions on values**

- **Keys will be listed in arbitrary order**
  - again, because of hashing

Tuples

- **Immutable lists**
- **Faster than lists**

```python
# key = ("lastname", "firstname")
# last = key[0]
# point = x, y, z  # parentheses optional
# singleton = {1}  # trailing comma!!!
# (required otherwise a value)
# empty = ()  # parentheses!
```

Variables

- **Need to assign (initialize)**
- use of uninitialized variable raises exception

- **No need to declare type (dynamically typed)**

```python
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

- However once set the type matters
  - Can’t treat integer as a string

Reference Semantics (Java Like)

- **Assignment manipulates references**
  - `x = y` – does not make a copy of `y`
  - makes `x` reference the object `y` references

- **Very useful; but beware!**

- **Example:**
  ```python
  >>> a = [1, 2, 3]
  >>> b = a
  >>> a.append(4)
  >>> print b
  [1, 2, 3, 4]
  ```

Changing a Shared List

```python
a = [1, 2, 3]
a.append(4)
```

```plaintext
a
b
```

```plaintext
1 2 3
```

```plaintext
a
b
```

```plaintext
1 2 3 4
```
Changing an Integer

\[
\begin{align*}
a & = 1 \\
b & = a \\
a & = a + 1
\end{align*}
\]

Control Structures

\[
\begin{align*}
\text{if condition:} & \\
& \text{statements} \\
[\text{elif condition:} & \\
& \text{statements}] \ldots \text{else:} & \\
& \text{statements} \\
\text{while condition:} & \\
& \text{statements} \\
& \text{for var in sequence:} & \\
& \text{statements} \\
& \text{break} & \\
& \text{continue}
\end{align*}
\]

More for Loops

- looping through list
  - for item in list:
    - print item
- looping through counter
  - for \(i\) in range(5):
    - print \(i\),
- Iterating through a 'built in' dictionary
  - import os
  - for \(k,v\) in os.environ.items():
    - print "\$s=$t" \% (\(k,v\))
- 'os.environ' is a dictionary of environment variables

Exercise I

Print (on separate lines)
1x1=1 1x2=2 1x3=3 ... 8x9=72 9x9=81

But don’t repeat. For example
print only 3x5=15
but don’t print 5x3=15
so print only if first_number \(<=\) second_num

Hint: use range
for num in range(1,10):
  ...

Output

*1x1=1
*1x2=2
*1x3=3
*1x4=4
*1x5=5
*1x6=6
*...

Exercise Answer

\[
\begin{align*}
a & = \text{range}(1,10) \\
b & = \text{range}(1,10)
\end{align*}
\]

for \(anum\) in \(a\):
  for \(bnum\) in \(b\):
    if \{ anum \(<=\) bnum \}:
      print \(\text{str(anum)},\"x\",\text{str(bnum)},\"=\",\text{str(anum*bnum)}

\]
Grouping Indentation

In Python:
for i in range(20):
    if i%3 == 0:
        print i
    if i%5 == 0:
        print "Bingo!"
print "---"

In C:
for ( i = 0; i < 20; i++ )
{
    if (i%3 == 0) {
        printf("%d
", i);
        if (i%5 == 0) {
            printf("Bingo:n");
        }
    }
    printf("---\n");
}

Example Function

def gcd(a, b):
    "greatest common divisor"
    while a != 0:
        a, b = b%a, a  # parallel assignment
    return b

>>> gcd.__doc__
'greatest common divisor'

>>> gcd(12, 20)
4

Exercise II

• Write script in the Editor window to convert a Fahrenheit temperature to a Celsius temperature and print out the result in the Console window

\[ ^\circ C = \left( ^\circ F - 32 \right) \times \frac{5}{9} \]

Exercise III

• Phone book application
  - 1) add
    • Ask for name and phone number
  - 2) print phone book
• To get input:
  - answer = raw_input("Enter your select

On your own…

• modules & packages
• exceptions
• files & standard library
• classes & instances

Hands On

• www.python.org/doc/current/tut/tut.html

Python Slogans

• Python Fits Your Brain, Bruce Eckel
• Life is Better Without Braces, Bruce Eckel
• Import This
• Batteries included (Tcl origin)
• Powered by Python
• Readability counts, Tim Peters

Bruce Eckel’s Top 10

10. Reduced clutter.
   Programs are read more than they are written
   Consistent formatting is important
   readability & compactness
   conversation of compactness
   Consistent use of programming idioms

09. It’s not backward-compatible with other languages. (This came with some hilarious one-liners:
   “C++’s backward compatibility with C is both its strength and its bane”;
   “Java causes repetitive-strain syndrome”;
   “Perl is compatible with every hacky syntax of every UNIX tool ever invented”;
   “C# and Microsoft .NET are backward-compatible with Microsoft’s previous marketing campaigns”;
   and
   “Javascript is not even compatible with itself.”)

08. It doesn’t value performance over my productivity.
   C++ memory leaks
   primitive types require awkward coding

07. It doesn’t treat me like I’m stupid.
   Java insists operator overloading is bad because
   you can make ugly code with it.
   Bruce observes, “And we all know there’s no ugly Java code out there.”

06. I don’t have to wait forever for a full implementation of the language.
   features invented in C++ takes a long time to appear in languages
   Unused features don’t get tested

05. It doesn’t make assumptions about how we discover errors.
   Is strong static type checking really the only way to be sure?
   Lack of good static typing in pre-ANSI C was troublesome
   Doesn’t mean it’s the best solution

04. Marketing people are not involved in it (yet).
   Java is flawless
   Microsoft happens “Visual” C++
   Of course Python isn’t immune

03. I don’t have to type so much.
   But what I do type is the right typing.

02. My guesses are usually right.
   I still have to look up how to open a file every time I do it in Java
   Most things I do in Java, I have to look up.
   Remember Python idioms easier because they are simpler

http://mindview.net/
Bruce Eckel’s Top 10

01. Python helps me focus on my concepts rather than on fighting with the language.