Python Basics
Python

• Scripting language
  – Can execute individual lines of code and observe results immediately
  – Executed code changes the state of the environment

• But …
  – We can also define functions and object classes

Good practice: prototype with direct interaction, but push refined code into reusable functions/classes
A Jupyter notebook consists of a list of **cells**

- Each cell can contain code to be directly executed or function / class definitions
- When a cell is executed, it is “pushed” to the python environment
- An executed cell may generate some form of output (text or graphics)
- Cells can be executed as many times as you like
Python Variables

• Primitive variable types include: integers, floats, strings, Boolean
• Aggregate variable types include a variety of lists plus hashes
• Variables types are not explicitly declared (much of the time), but instead are determined automatically by the context of the variable assignment
• Variable type checking is done at run time!
• Live Jupyter Demonstration
Python Basics: Conditionals

CV_M3_L02
Python Basics: Conditionals
Python Basics: Conditionals

• Standard if/then/else structure
• Caveat:
  – Blocks of code are not surrounded by “{“ and “}”
  – Instead, the block is inferred by the indentation level
• Live Jupyter Demonstration
Python Basics: Lists and Tuples

CV_M3_L03
Python Basics: Lists and Tuples
Python Basics: Lists and Tuples

• Both Lists and Tuples:
  – Implementations of array-like structures
  – Zero indexed
  – Elements can be anything (primitive variables or objects)

• Difference:
  – Lists are mutable (can be changed after creation)
  – Tuples are immutable

Which one you choose is context dependent…
• Live Jupyter Demonstration: python-lists
Python Basics: For Loops

CV_M3_L04
Python Basics: For Loops
For loops step through all of the elements of a collection
– Collection can be an explicit group of objects (such as an array)
  or can be produced by an iterator object
• Lists and tuples: the elements are “visited” in index order
• Body of the `for` loop is indented (just like the body of the `if`
  statements)
range() returns an iterator object

- range(5) produces an iterator that generates 0, 1, 2, 3, 4
  - Stop = 5
- range(2, 5) generates 2, 3, 4
  - Start = 2, stop = 5
- range(1, 5, 2) generates 1, 3
  - Start = 1, stop = 5, step = 2
• Live Jupyter Demonstration: python-lists
Python Basics: For Loops with Zip

CV_M3_L05
Python Basics: For Loops with Zip
Python Basics: For Loops with Zip

zip()
• Input parameters: some number of iterable objects
• Produces a new iterator that generates tuples
  – Each tuple has one item from each iterable object

Can then use a for loop to iterate over these tuples
• Live Jupyter Demonstration: python-lists
Insert IPAD_M3_L05
• Live Jupyter Demonstration: python-lists
Python Basics: Dictionaries

CV_M3_L06
Python Basics: Dictionaries

Dictionaries: Implementation of a Hash Map

• Set of unique keys
• Each key is associated with some value
  – Can be anything (primitive data or objects)
• Fundamental Python data structure
• Live Jupyter Demonstration: python-lists
Python Basics: List Comprehension

CV_M3_L07
Python Basics: List Comprehension
Python Basics: List Comprehension

• There are many cases where we would like to perform the same operation on each item in a list
• One could implement a `for` loop to do this
• List comprehension provides a compact way of implementing these for loops
• Live Jupyter Demonstration: python-lists
Python Basics: Functions
Python Basics: Functions

Functions:
• Provide a way for us to construct reusable pieces of code
• Give us a mechanism to organize code in more manageable units

In Jupyter:
• Define a function in a cell
• For this function to be “pushed” into the active python environment, we must execute the cell
• Live Jupyter Demonstration: python-lists
Python Basics: Classes

CV_M3_L09
Python Basics: Classes
Python Basics: Classes

Objects are composed of:

• A set of *instance variables* that describe the state of a single object

• A set of operations that can be performed on that object (i.e., *instance methods*)

• Underlying representation for both is a dictionary!  
  – Python is happy to let us exploit this property
Python Basics: Best Practices

CV_M3_L10
Python Basics: Best Practices
Python Basics: Best Practices

Power of Python as a scripting language:
• No explicit variable type declaration
• “Lazy” variable type checking
• Can execute lines of code immediately & observe the results

-> Can quickly throw together solutions to problems
Python Basics: Best Practices

Functions and Classes

• Provide ways of constructing modular, reusable blocks of code
• Once you have developed and tested a procedure, it is often worth taking the time to push the implementation into one or more functions or classes
• This step makes it easier to use and debug your code, and to apply it to new situations in the future
Global Variables

- Useful to declare a high-level context for your code to execute in (e.g., configuring paths or model parameters)
- But:
  - Avoid referencing global variables inside of functions and class methods
  - Instead, the values contained within a global variable should be passed as a parameter to these functions / class methods
Python Basics: Best Practices

Code Examples on the Net

- There are many examples out there that solve various problems
- But, these examples are often poor examples of proper programming
  - Often avoid the use of functions / classes
  - Ugly use of global variables

- You should strive to:
  - Understand code that you are writing
  - Develop quality code