ECON 815 - Computational Methods for Economists

Fall 2019

Notes on OOP and Python

What is Python?

- Python is a full programming language
  - Compare to Stata (for statistical/econometric analysis) and Matlab (for mathematical computing), Python allows one to do just about anything
  - You can do stats and math – but it can also be used to write webpages, make dynamic visualizations, apply GIS tools, etc.
- Python is an "Object-oriented Programming" (OOP) language
  - OOP is a programming paradigm (the alternative is procedural programming)
  - OOP languages differ from procedural languages because they focus on the objects not the process
  - Procedural languages have code that is written step-by-step in the way the program would be executed
  - OOP languages have code that is written in a more modular fashion - where different aspects of the process are separated with instructions about how to interact with other parts
  - Because of this OOP is less top-down and more abstract
  - But it makes the code more maintainable and (often) a better model of the real world thing you are trying to have the software represent
- Python is an "interpreted language"
  - This means that the Python syntax you write calls pre-compiled subroutines that then execute machine-readable code
  - Interpreted (as compared to compiled) languages generally require less coding and are easier to execute on the fly (no need to run a compiler)
- Python is an open source language
  - Thus it's freely available for any purpose
- Why Python for economics?
  - Easy to learn
  - Flexible (it's a full programming language!)
  - Numerical computing straight-forward and can be fast (almost Fortran speed with much easier syntax)
  - Econometrics applications coming along, but:
    - Great for data munging
    - Can integrate R into Python code
  - Strong market share for data science (which employs more and more economists)

Setting up and Running Python

- At this point, you should have downloaded and install the Anaconda distribution of Python

- There are several ways to interact with Python:
  1. Though iPython (interactive Python)
     - All objects in script executed are available to inspect (helps with debuggin)
     - Sometimes error messages are not clear
     - Need to be careful about reloading modules after editing them
  2. In Jupyter Notebooks
     - Great for exploring data (visualizations can be produced in line)
     - Nice to sharing (rendered on GitHub, can print to html or pdf)
     - Not ideal for developing large amounts of code
  3. In a Python-specific IDE like Spyder
     - Uses iPython to run - so many of same pluses and minuses of iPython
     - Text editor with syntax highlighting and linter
     - GUI interfaces to explore objects (much like Matlab's Workspace)
  4. Edit files in text editor and execute in command line
     - Can use powerful text editor with syntax highlighting, linter, and other features
     - No issues needing to reload modules
     - Good error messages
     - Cannot explore objects interactively
- I, and most developers I know, use the last method most often.
- But I've found Jupyter Notebooks extremely helpful for exploring data and certain types of debugging
- I'll use iPython in some situations, but almost never use Spyder
- But YMMV and you should be aware of all of these

Getting Started

- Python built-in types:
  - Numbers -int, float, long, complex
    - Can cast as different type.
    - E.g., `int(8), float(8)`, etc
  - Booleans
    - =True/False (1/0)
  - Strings
    - Enclosed in single '' or double "" quotes
  - Lists
    - Defined with square brackets.
    - E.g., `this_list = [1, 2, 3]`
    - Lists can contain any type - a list of numbers, strings, lists, dictionaries, etc.
      - Lists can have mixed types, but typically you want to have a given list contain elements all of the same type
      - Use a tuple when mixing types
  - Tuples
    - Defined with parentheses.

    - E.g., `this_tuple = (1, 2, 3)`
    - Tuples can contain any type - a list of numbers, strings, lists, dictionaries, etc.
    - You can think about a tuple as a row in a database – a container for things that go together but may be of varying types
  - Sets
    - Like a list, but only contains unique elements
    - So it's useful for defining unique sets of things or doing membership tests
    - Defined with curly brackets.
    - E.g., `this_set = {1, 2, 3}`
  - Dictionaries
    - Related key value pairs
    - E.g., `this_ dict = {'first': 1, 'second': 2, 'third': 3}`
    - Useful for looking things up using the mapping provided
    - E.g., `this_dict['first'] = 1`

- Python syntax
  - No end of line punctuation
  - Loops or conditional statements have no `end` or closing bracket
    - The action inside loop/if the condition is met is denoted as being indented
  - A color comes at the end of the `if/for/while` (or similar) statement

- Coding conventions
  - PEP8 rules are the standard python style guide (line length, spacing, etc)
  - You'll want to use docstrings to define functions and in-line comments where appropriate
  - "Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."