Basics of Python

ACE 592 SAE

Benevolent Python Dictator for Life:

Guido Van Rossum

Why are we using Python?

Some things about Python:

1. It is general purpose.
2. It is open source.
3. It is *relatively* easy to use.

Here I'll go through why each of these things is important.

1. General Purpose

- R and Stata are aimed primarily at **data analysis and data processing**.
- Python is not specific to data analysis. It is used to do machine learning, web scraping, or even writing applications.

Advantage: Since it isn't specific to any one task, it can do nearly all of them competently.

Disadvantage: Its "econometrics" support is lacking compared to R and Stata.

2. Open Source

- Stata and Matlab cost **a lot of** money, being sold by private companies.
- Open Source languages (e.g. R, Python) are **free to use.**
- Maintained by a community, there are **packages for practically everything.**

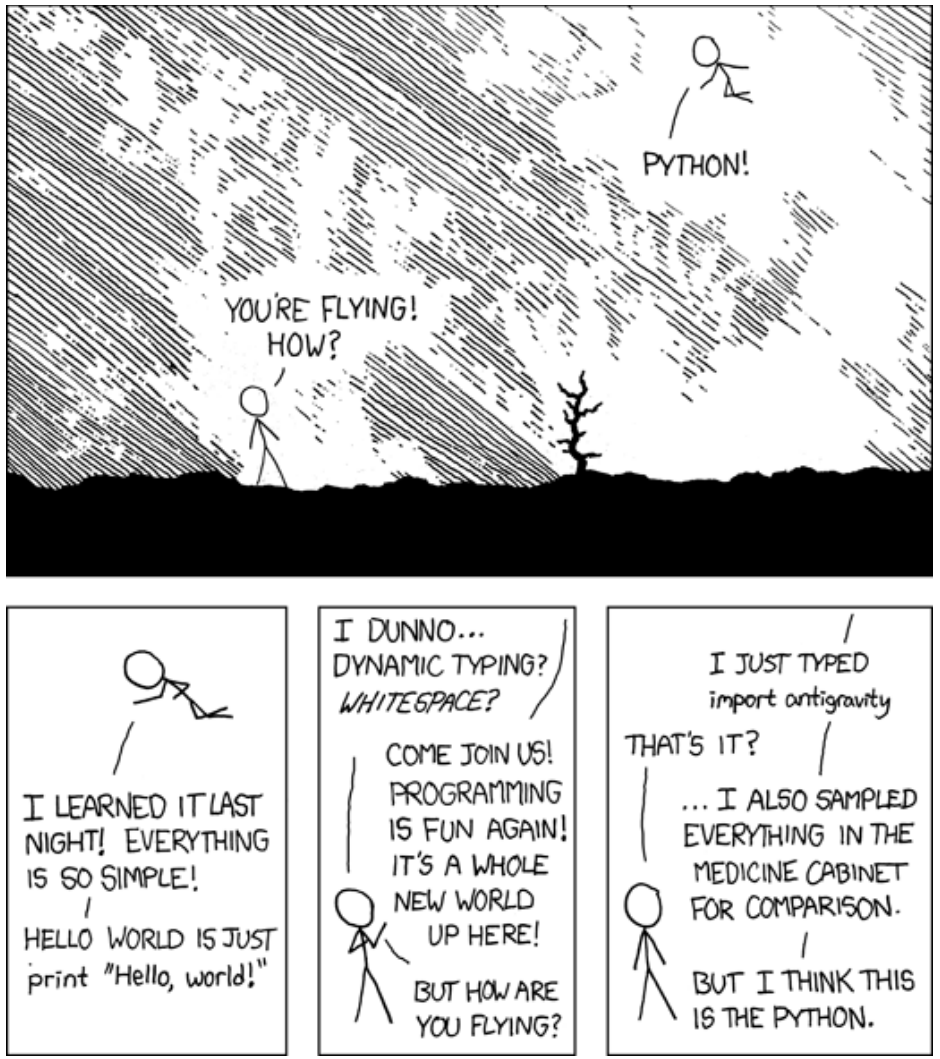
Disadvantage: packages don't always play nice with each other.

open source initiative
Approved License®

3. Easy to Use

- Relative to other languages, Python is intuitive and **easy to understand.**
- **High level**, **object oriented**, and i**nterpreted** make it easy to use and debug.

Disadvantages: It is slow and not always memory efficient.



When is Python not enough?

- Econometrics support is still lacking, especially for complex models.
  - The packages **statsmodels** and **econtools** have made some strides in this area, but not near the support of R and Stata.

- Python is **an interpreted language** which means for some operations it will by slow.
  - The packages **numba** and **cython** can help us do pretty well. Otherwise the language **Julia or C** is good for faster programming.

Otherwise, Python can usually do just as good as other languages.

The Building Blocks of Python

What does "object oriented" mean?

This has no short answer. But one way to understand it is that Python is mostly made up of **objects**. Objects:

1. Nest **attributes** and **methods.**
2. Are represented by one higher level interface.

The innovation of this is not having to muck around with several very similar variables or functions. Instead objects package them all together.

How Python Solves a Problem

Imagine an “object” called “LinearRegression.”

What do we want to do with a linear regression “object”?

- Feed it data.
- Make it estimate a model using the data.
- Report the parameters back to us.

Python Building Blocks

Here we'll talk about:

- Data types.
- Functions.
- Classes.

And motivate it using the example of the "LinearRegression" object.

Python Data Types

Python offers the following *basic data types*, which we will use in this course:

Data type	Description
`int()`	Integers
`float()`	Floating point numbers
`str()`	Strings, i.e., unicode (UTF-8) texts
`bool()`	Boolean, i.e., `True` or `False`
`list()`	List, an ordered array of objects
`tuple()`	Tuple, an ordered, unmutable array of objects
`dict()`	Dictionary, an unordered, associative array of objects
`set()`	Set, an unordered array/set of objects
`None()`	Nothing, emptyness, the void..

Source: PyEcon, slide 38

Examples with the Number 2

Name	Python Name	Description	Example
String	*str()*	A text string	“2”
Integer	*int()*	An integer	2
Float	*float()*	A number with decimal places	2.0
Boolean	*bool()*	A statement that is either *True* or *False*	2>1, which would code *True* 2>4, which would code *False*
List	*list()*	An ordered array of data	[2, “2”,2.0,2>1]
Dictionary	*dict()*	A mapping of elements	{“Bart”:2, “Harley”:3}

Code Example

So to review

- We went over the different ***types of data*** in Python.
- We wrote a ***function*** that takes two lists of numbers and finds the sum of squared errors.
- We wrote a ***class*** that bundles them all together.

Why are classes useful?

How to Use Python

Scripting (Non-Interactive)

Write it first

theory_models.py

projects > culling-analysis > lib > estimation > theory_models.py > ...

```
                                quad(self.trans_integrand,self.cutoffs[i],self.cutoffs[i+1],args=(j,))[0]
        return Q

    def trans_integrand(self,u,j):
        return np.exp((-(u-self.mu)**2)/(2*self.var_y))*\
                (norm.cdf((self.cutoffs[j+1] - self.mu*(1-self.rho) - self.rho*u)/self.sig)-\
                 norm.cdf((self.cutoffs[j] - self.mu*(1-self.rho) - self.rho*u)/self.sig))

    def cond_mean_AR1(self,i):
        return self.N*self.sig_y*(norm.pdf((self.cutoffs[i]-self.mu)/self.sig_y)-\
                                  norm.pdf((self.cutoffs[i+1]-self.mu)/self.sig_y)) + self.mu


def create_shocks(sige2,rho,m,d,dep=True):
    sige = sige2**.5
    mu1e = 0

    yn = round(sige*m)

    ShockGrid = np.arange(-yn,yn+d,d)
    E = len(ShockGrid)
```

Anaconda Prompt (Anaconda3)

```
(base) C:\Users\jhtchns2>python theory_models.py
```

Then run it from the terminal

Integrated Development Environment (IDE)

Write it, run it, change it, etc.

How to Use Python

Scripting (Non-Interactive)

- Less overhead, good for things that need to be run “under the hood.”
- Best for things you know you need to run over and over again e.g. a simulation, data cleaning.

Integrated Development Environment (IDE)

- Good for testing code to make sure it works.
- Best for things that need extensive comments or need to be “interactive.”

Managing Packages in Python

- “Packages” are libraries that Python can use for various tasks.
- Each script usually begins with a block of code “importing” all the libraries we need.
- To install packages:
  - **Pip:** Very common package manager that comes standard on Anaconda.
    - Example: “pip install pandas”
  - **Conda:** Package manager that comes with Anaconda that can also manage **environments.**
    - Example: “conda install pandas”

Why We Are Using Conda

- For the class, we need to work with a specific set of packages specified in an “environment” file.
- An environment is a self contained “version” of Python.
- Useful when you need specific versions of packages (default installs the newest one).
- Conda can manage environments easiest.

Your Next Task:

1. Install Anaconda.
2. Install our environment "`ace-592-sae`" onto your computer using conda.
   - **Use the documentation in the "preliminaries" document**
   - **You cannot do this step unless you have first cloned our repo on to your computer to get the "environment.yml" file.**

Demonstration