

# Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

Vrije Universiteit Amsterdam  
Tinbergen Institute

`c.s.bos@vu.nl`

August 2019 – Version Python

Installation

Syntax

Commenting

Loops

Matrices

Operators

Praise of `t.py`

Mutable vs immutable

## Syntax frames

Below a series of slides on syntax in Python  
Read them through, try out in small programs if you understand  
the meaning.

## Placeholder

Material on Python syntax should appear here...

## Base installation of Python

Many ways. . . Here:

- ▶ MiniConda (<https://conda.io/miniconda.html>): This installs the base Python 3.7, with minimal fuss. On Windows, add the Miniconda3 and Miniconda3\scripts directories to your path.
- ▶ At Conda command prompt (= terminal on OSX/Linux), install packages spyder IPython, Matplotlib, NumPy, SciPy, HDF5, Pandas and StatsModels through

```
conda install ipython matplotlib numpy scipy hdf5 \
pandas statsmodels
```

- ▶ Once in a while, update it all from Conda command prompt, using

```
conda update --all
conda clean --all
```

## Full installation of Python

Alternatively, use a full installation of anaconda:

- ▶ AnaConda (<https://www.anaconda.com/download/>): This installs the base Python 3.7+packages+Spyder, with minimal fuss.
- ▶ At Conda command prompt (= terminal on OSX/Linux), update occasionally, using

```
conda update --all
conda clean --all
```

## Editor/IDE

For editing/running programs, several options again:

- ▶ Whatever editor of choice, run from command line (go ahead)
- ▶ Spyder: Install (if needed) through

```
conda install spyder
```

- ▶ Atom: Install from <https://atom.io> with packages Hydrogen, Autocomplete-python, and add

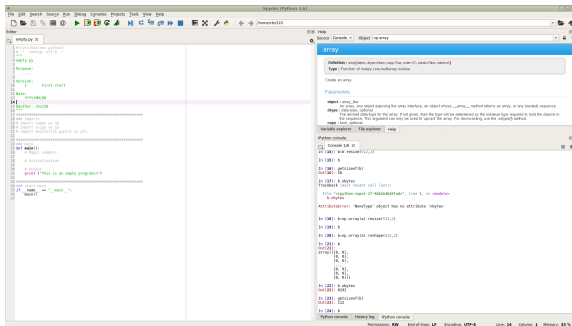
```
conda install jupyter
```

- ▶ IPython: Install (if needed) through

```
conda install ipython
```

(You'll probably see me switching; I use Atom for all editing of Python, R, Ox,  $\text{\LaTeX}$ , but sometimes prefer Spyder, IPython for quick testing)

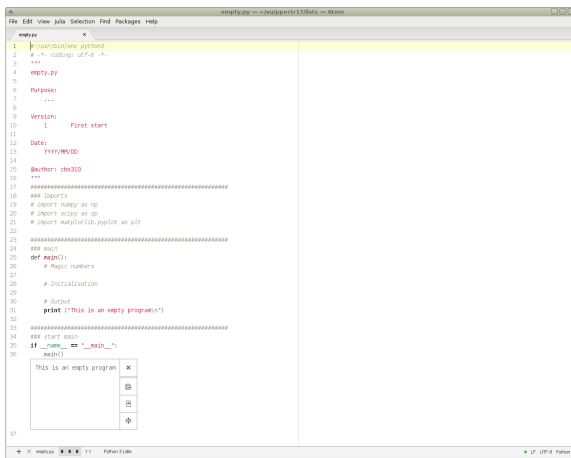
# Spyder



Spyder environment



# Atom



```

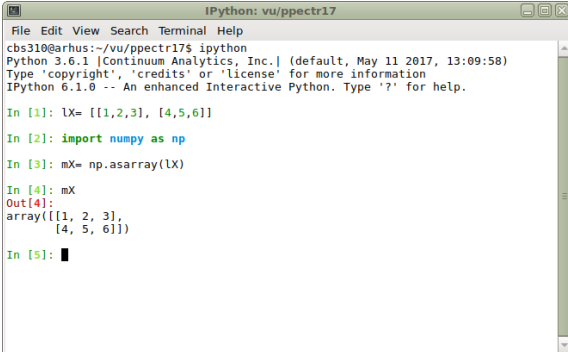
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  ...
4  empty.py
5
6  Purpose:
7  ...
8
9  Version:
10  1    First start
11
12  Date:
13  YYYY/MM/DD
14
15  @author: chs310
16  ...
17  =====
18  ## Imports
19  # Import numpy as np
20  # Import scipy as sp
21  # Import matplotlib.pyplot as plt
22
23  =====
24  ## main
25  def main():
26      # Magic numbers
27
28      # Initialisation
29
30      # Output
31      print ("This is an empty program\n")
32
33  =====
34  ## start main
35  if __name__ == "__main__":
36      main()
37

```

This is an empty program

Atom environment

# IPython



```
IPython: vu/ppectr17
File Edit View Search Terminal Help
cbs310@arhus:~/vu/ppectr17$ ipython
Python 3.6.1 |Continuum Analytics, Inc.| (default, May 11 2017, 13:09:58)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: lX= [[1,2,3], [4,5,6]]

In [2]: import numpy as np

In [3]: mX= np.asarray(lX)

In [4]: mX
Out[4]:
array([[1, 2, 3],
       [4, 5, 6]])

In [5]: █
```

IPython environment

## Commenting: Begin of program

### Listing 1: pow0.py

```
"""  
pow0.py  
  
Purpose:  
    Calculate 2^8  
  
Version:  
    0          Outline of a program  
  
Date:  
    2017/6/19  
  
Author:  
    Charles Bos  
"""
```

At least:

- ▶ Name of program
- ▶ Purpose
- ▶ Version
- ▶ Date
- ▶ Author

## Commenting: Start of function (CSB)

### Listing 2: pow3.py

```
def Pow(dBase, iPow):  
    """  
    Purpose:  
        Calculate  $dBase^{iPow}$   
  
    Inputs:  
        dBase      double, base  
        iPow       integer, power  
  
    Return value:  
        dRes       double,  $dBase^{iPow}$   
    """
```

At least:

- ▶ Purpose
- ▶ Inputs (if any)
- ▶ Return value (if any)

NB: This is style CSB, used in course. Define the type of inputs/return values, and size in case of matrices

## Commenting: Start of function (NumPy)

### Listing 3: pow3.py

```
def Pow(dBase, iPow):  
    """  
    Calculate dBaseiPow  
  
    Arguments  
    -----  
    dBase : double  
           base  
    iPow  : integer  
           power  
  
    Returns  
    -----  
    dRes : double  
           dBaseiPow  
    """
```

At least:

- ▶ Purpose/summary
- ▶ Arguments (if any)
- ▶ Return values (if any)

NB: This is style NumPy, see [docstring](#) definition

# For Loops

```
# Simple loop
for i in range(3):
    print ("i= %i" %i)

# Loop over elements
vX= ('cat', 'dog', 'horse')
for sX in vX:
    print ("I like a %s" % sX)

# Loop over letters
sX= 'abc'
for s in sX:
    print ("This is letter %s" % s)

# Enumerate letters
for (i, s) in enumerate(sX):
    print ("Letter %i is %s" % (i, s))
```

## Note:

- ▶ Loop over a construct of elements
- ▶ Construct may be simple range(iN)
- ▶ Or tuple, list, array, string
- ▶ Use enumerate to get both index and element

# For Loops

```
# Simple loop
for i in range(3):
    print ("i= %i" %i)

# Loop over elements
vX= ('cat', 'dog', 'horse')
for sX in vX:
    print ("I like a %s" % sX)

# Loop over letters
sX= 'abc'
for s in sX:
    print ("This is letter %s" % s)

# Enumerate letters
for (i, s) in enumerate(sX):
    print ("Letter %i is %s" % (i, s))
```

## Note:

- ▶ Loop over a construct of elements
- ▶ Construct may be simple range(iN)
- ▶ Or tuple, list, array, string
- ▶ Use enumerate to get both index and element

# Matrices

```
iN= 10; iK= 3; dSigma= 2;  
  
mX= np.random.randn(iN, iK)  
vEps= dSigma*np.random.randn(iN)  
vBeta= 1+np.array(range(iK))  
  
vY= mX@vBeta + vEps
```

Note:

- ▶ Check shapes:  
 $(n \times k) \times (k \times 1) \equiv (n \times 1)$ ,  
and  $(n \times k) \times (k) \equiv (n)$ :  
 $\beta$  might be  
one-dimensional
- ▶ Matrices are *mutable*:  
 $mX[1, 2] = 5$  resets an  
element
- ▶ **Warning**: This works also  
for function arguments;  
elements of incoming  
mutable arguments may  
be changed!



# Operators

- + Addition
- − Subtraction
- \* Element-wise multiplication
- @ Matrix multiplication
- \*\* Exponentiation
- % Modulus
- / Element-wise division
- // Floor of division

<https://www.>

[tutorialspoint.com/python/python\\_basic\\_operators.htm](https://www.tutorialspoint.com/python/python_basic_operators.htm)

To add: Logical, comparison, assignment

## Praise of t.py

Remember the **Droste effect** :

- ▶ Any program is written in small building blocks
- ▶ Each block has clearly defined input/output
- ▶ Hence each block can be tested independent of the rest

⇒ Each block *should* be tested independently of the rest!

Conclusion:

- ▶ Copy routine to separate file t.py
- ▶ Prepare input data (also border cases, robustness?)
- ▶ Check output

## Mutable vs immutable

| Immutable  | Mutable  |
|--|--|
| Boolean, integer, double, string, tuple, frozenset | List, set, dictionary, np.array, pd.dataframe, ... |

Immutable objects cannot be changed after formation; mutable object can be changed.

Python tries to lower memory use; a statement as

```
a= 'hello'; b= 'hello'
print (a is b)           # Check if a and b point at the same object, True
```

results in 'True'.

If *a* later is linked to a new string (another *immutable*), it becomes a new object:

```
a= 'there'
print (a is b)           # Check if a and b point at the same object, False
```

## Mutable vs immutable II

With mutables, this creates a risk:

```
a= [1, 2, 3]; b= a
print (a is b)           # Check if a and b point at the same object, True
a[0]= 5
print (a is b)           # Check if a and b point at the same object, True
print ("b is now :", b)  # Note how the first element of b is changed to 5...
```

With mutables:

- ▶ assignment leads to two variables, two names, pointing to one and the same object
- ▶ changes to elements of one variable change the other
- ▶ this may **lead to unwanted side-effects!**
- ▶ and also allow **desired** side-effects...

# Mutable vs immutable: Unwanted side effects

Solution to *unwanted* side effects:

- ▶ be careful
- ▶ use `b= a.copy()` when needed

```
a= [1, 2, 3]; b= a.copy()
print (a is b)           # Check if a and b point at the same object, False
a[0]= 5
print ("b is now :", b) # Note how the first element of b is NOT changed to 5...
```

# Mutable vs immutable: Desired side effects

Option for *desired* side effects:

- ▶ be careful, handing function argument of correct size
- ▶ alter function arguments, while explicitly mentioning in documentation

```
def MyOls(vY, mX, avBeta):
    """
    Purpose:
        Run OLS on Y and X

    Inputs:
        ...
        avBeta        vector of size iK

    Outputs:
        avBeta        vector of size iK, filled with OLS estimates

    Return value:
        br            boolean, True if all went well
    """
    avBeta[:] = np.linalg.inv(mX.T@mX)@mX.T@vY
    return not np.any(np.isnan(avBeta))
```

**NB:** For historical reasons, I tend to use 'a' as indicator for an address to a variable that will be changed in a function.

## More syntax?

- ▶ Python for Econometrics, [https://www.kevinsheppard.com/Python\\_for\\_Econometrics](https://www.kevinsheppard.com/Python_for_Econometrics) of Kevin Sheppard
- ▶ Quantitative Economics, <https://lectures.quantecon.org/py/> in Python, by Thomas Sargent and John Stachurski
- ▶ Check the packages with documentation, e.g. NumPy, <https://docs.scipy.org/doc/numpy/reference/>
- ▶ Check with [Google](#) or colleagues