

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 2020 – Version Python

Separate lecture slides

Compilation: July 27, 2020

Overview

Principles of Programming in Econometrics

D0: Syntax, example 2⁸

D1: Structure, scope

D2: Numerics, packages

D3: Optimisation, speed

Day 0: Syntax

9.30 Introduction

Example: 2^8

Elements

Main concepts

Closing thoughts

Revisit E0

13.30 Practical (at VU, main building)

- ▶ Checking variables, types, conversion and functions
- ▶ Implementing Backsubstitution

Programming by example

Let's start simple

- ▶ Example: What is 2^8 ?
- ▶ Goal: Simple situation, program to solve it
- ▶ Broad concepts, details follow

Power: Steps

First steps:

- ▶ Get a first program (pow0.py)
- ▶ Initialise, provide (incorrect) output (pow1.py)
- ▶ for-loop (pow2.py)
- ▶ Introduce function (pow3.py)
- ▶ Use a while loop (pow4.py)
- ▶ Recursion (pow5.py)
- ▶ Check output (pow6.py)

Power: First program

Listing 1: pow0.py

```

"""
pow0.py
Purpose:
    Calculate 2^8
Version:
    0          Outline of a program
Date:
    2017/6/19
Author:
    Charles Bos
"""
#####
### Imports
# import numpy as np

#####
### main
print ("Hello world\n")

```

To note:

- ▶ Explanation of program, in triple quotes `"""` ((docstring))
- ▶ Comments `#`
- ▶ Possible imports
- ▶ Main code at bottom

Power: Initialise

Listing 2: pow1.py

```
# Magic numbers
dBase= 2
iC= 8
# Initialisation
dRes= 1
# Estimation
# Not done yet...
# Output
print ("The result of ", dBase, "^", iC,
      " = ", dRes, "\n")
```

To note:

- ▶ Each line is a command
- ▶ Distinction between 'magics', 'initialisation', 'estimation' and 'output'
- ▶ Function `print(a, b, c)` is used

Power: Estimate

Listing 3: pow2.py

```
#####  
### main  
# Magic numbers  
...  
# Estimation  
for i in range(iC):  
    dRes = dRes * dBase  
  
# Output  
...
```

To note:

- ▶ For loop, counts in extra variable *i*
- ▶ Function `range(iStop)`, counts from 0, ..., *iStop*-1
- ▶ Executes indented commands after `for i in range(iC):`
- ▶ Mind the `:` after the `for` statement

Intermezzo 1: Check output

Intermezzo 2: Check [The for and while loops](#).

Intermezzo 3: Discuss [why](#) the `range()` function (and indexing, later), is [upper-bound exclusive](#).

Power: Functions

Listing 4: pow3.py

```
def Pow(dBase, iPow):
    """
    Purpose:
        Calculate dBase^iPow
    Inputs:
        dBase      double, base
        iPow       integer, power
    Return value:
        dRes
    double, dBase^iPow
    """
    dRes = 1
    for i in range(iPow):
        # print ("i= ", i)
        dRes = dRes * dBase
    return dRes
### Main
dRes = Pow(dBase, iC)
```

To note:

- ▶ Function has own [docstring](#)
- ▶ Function defines two arguments
dBase, iPow
- ▶ Function *indents one tab forward*
- ▶ Uses local dRes, i
- ▶ returns the result
- ▶ And dRes= Pow(dBase, iC)
catches the result dRes= 256.

- ▶ Allows to re-use functions for multiple purposes
- ▶ Could also be called as dRes= Pow(4, 7)
- ▶ Here, only one output

Power: While

Listing 5: pow3.py

```
dRes= 1
for i in range(iC):
    dRes= dRes*dBase
```

Listing 6: pow4.py

```
dRes= 1
i= 0
while (i < iPow):
    dRes= dRes*dBase
    i+= 1
```

To note:

- ▶ The `for i in range(iter)` loop corresponds to a `while` loop
- ▶ Look at the order: First init, then check, then action, then increment, and check again.
- ▶ The `for`-loop is slightly simpler, as beforehand the number of iterations is fixed.
- ▶ A loop command can be a *compound* command, multiple commands all indented equally.

Power: Recursion

Listing 7: pow5.py

```
def Pow_Recursion(dBase, iPow):  
    # print ("In Pow_Recursive, with iPow= ", iPow)  
    if (iPow == 0):  
        return 1  
  
    return dBase * Pow_Recursion(dBase, iPow-1)
```

To note:

- ▶ $2^8 \equiv 2 \times 2^7$
- ▶ $2^0 \equiv 1$
- ▶ Use this in a recursion
- ▶ New: If statement

Intermezzo: Check [Python manual on if statement](#), or a simpler [Wiki](#) on the same topic.

Q: What is *wrong*, or maybe just *non-robust* in this code?

Power: Recursion

Listing 8: pow5.py

```
def Pow_Recursion(dBase, iPow):  
    # print ("In Pow_Recursive, with iPow= ", iPow)  
    if (iPow == 0):  
        return 1  
  
    return dBase * Pow_Recursion(dBase, iPow-1)
```

To note:

- ▶ $2^8 \equiv 2 \times 2^7$
- ▶ $2^0 \equiv 1$
- ▶ Use this in a recursion
- ▶ New: If statement

Intermezzo: Check [Python manual on if statement](#), or a simpler [Wiki](#) on the same topic.

Q: What is *wrong*, or maybe just *non-robust* in this code?

A: Rather use `if (iPow <= 0)`, do not continue for non-positive `iPow`!

Power: Check outcome

Always, (*a/ways...!*) check your outcome

Listing 9: pow6.py

```
import math
...
# Output
print ("The result of ", dBase, "^", iC, "= ")
print (" - Using Pow(): ", Pow(dBase, iC))
print (" - Using Pow_Recursion(): ", Pow_Recursion(dBase, iC))
print (" - Using **: ", dBase ** iC)
print (" - Using math.pow: ", math.pow(dBase, iC))
```

Listing 10: output

```
The result of 2 ^ 8 =
- Using Pow(): 256
- Using Pow_Recursion(): 256
- Using **: 256
- Using math.pow: 256.0
```

Power: Check outcome II

To note:

- ▶ Yes, indeed, Python has (multiple...) power operators readily available.
- ▶ Always check for available functions...
- ▶ And carefully check the manual, for difference between $x^{**}y$, `pow(x,y)`, `math.pow()`.

Q: And what is this difference between the powers?

Power: Check outcome II

To note:

- ▶ Yes, indeed, Python has (multiple...) power operators readily available.
- ▶ Always check for available functions...
- ▶ And carefully check the manual, for difference between $x^{**}y$, `pow(x,y)`, `math.pow()`.

Q: And what is this difference between the powers?

A: According to the [manual](#), `math.pow()` transforms first to floats, then computes. The others leave integers intact.