Effective Programming Practices for Economists

Basic Python

`for` loops

Janoś Gabler and Hans-Martin von Gaudecker

Topics

- **D**on’t **R**epeat **Y**ourself
- for loop syntax
- Are for loops bad?
- Looping over lists and tuples
- Looping over dicts
- Common looping patterns

Don’t repeat yourself

```
>>> names = ["Guido", "Raymond", "T
>>> lower_names = [
>>>   names[0].lower(),
>>>   names[1].lower(),
>>>   names[2].lower(),
>>> ]
>>> lower_names
['guido', 'raymond', 'tim']
```

- The code on the left has problematic code repetition
  - If we have a typo, we need to fix it multiple times
  - Cumbersome if list becomes longer
- In many situations we want to do similar things multiple times
  - Cleaning several similar variables
  - Fitting several models
  - …

A simple for loop

```
# example
>>> for i in range(5):
...     print(i ** 2)
0
1
4
9
16

# general pattern
for running_variable in iterable:
    do_something(running_variable)
    and_something_else(running_vari
```

- for loops let us do things repeatedly
- First line ends with a `:`
- In each iteration, the running variable is bound to a new value
- Loop body with one or several lines is indented by 4 spaces

Are for loops bad

- For loops have a bad reputation for being slow and inelegant, but:
  - Having unnecessary code repetition is worse than a for loop!
  - Slowness only matters if it is a bottleneck
  - Sometimes they are the most readable solution
  - Sometimes they are the fastest solution!
- For now, use for loops without hesitation
- Later you will learn when to use alternatives

Looping over lists and tuples

```
>>> names = ["Guido", "Raymond", "T
>>> for name in names:
...     print(name.lower())
'guido'
'raymond'
'tim'
```

- Looping over lists and tuples works the same
- Running variable is iteratively bound to the list elemnts
- Try to choose a good name for the running variable!

Looping over dictionaries

```
>>> letter_to_position = {
...     "a": 0,
...     "b": 1,
...     "c": 2,
... }

>>> for key in letter_to_position:
...     print(key)
a
b
c

>>> for key, pos in letter_to_posit
...     print(key, pos)
a 0
b 1
c 2
```

- By default you loop over dictionary keys
- Use `` `.items` `` for looping over keys and values at the same time

Mapping loops

```
>>> names = ["Guido", "Raymond", "T
>>> lower_names = []
>>> for name in names:
...     lower_names.append(names.lo
>>> lower_names
['guido', 'raymond', 'tim']

>>> name_to_lower = {}
>>> for name in names:
...     name_to_lower[name] = name.
>>> name_to_lower
{'Guido': 'guido', 'Raymond': 'raym
```

- Create a new container by applying some transformation to each element in another container
- Transformations can be arbitrarily complex
- Often you will define a custom function to do the transformation
- Examples:
  - Create dict of results from dict of model specifications
  - Apply mathematical functions to lists of inputs

Reduction loops

```
>>> numbers = [1, 2, 3]
>>> mean = 0.0
>>> for number in numbers:
...     mean += number / len(number
```

- Examples of reductions are averages, sums, products and counts
- Assign the identity element of the reduction as initial value
- Update the result in each iteration