# Effective Programming Practices for Economists

# Basic Python

## "for" loops

Janoś Gabler and Hans-Martin von Gaudecker

# Contents

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

# Don’t repeat yourself

```
>>> names = ["Guy", "Ray", "Tim"]
>>> lower_names = [
>>>   names[0].lower(),
>>>   names[1].lower(),
>>>   names[2].lower(),
>>> ]
>>> lower_names
['guy', 'ray', 'tim']
```

- This code repetition is problematic
  - 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_var in iterable:
    do_someth(running_var)
    and_someth_else(running_var)
```

- 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 = ["Guy", "Ray", "Tim"]
>>> for name in names:
...     print(name.lower())
'guy'
'ray'
'tim'
```

- Looping over lists and tuples works in the same way
- Running variable is iteratively bound to the iterable's elements
- Try to choose a good name for the running variable!

# Looping over dictionaries

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

>>> for let in let_to_pos:
...     print(let)
a
b
c

>>> for let, pos in let_to_pos.items():
...     print(let, pos)
a 0
b 1
c 2
```

- By default you loop over dictionary keys
- Use `` `.items()` `` for looping over key/value pairs

# Looping patterns

- Mapping loops
- Reduction loops
- *(Filtering loops)*

# Mapping loops

```
>>> names = ["Guy", "Ray", "Tim"]
>>> lower_names = []
>>> for n in names:
...     lower_names.append(n.lower())
>>> lower_names
['guy', 'ray', 'tim']

>>> name_to_lower = {}
>>> for n in names:
...     name_to_lower[n] = n.lower()
>>> name_to_lower
{
  'Guy': 'guy',
  'Ray': 'ray',
  'Tim': 'tim'
}
```

- Create a new container by transforming each element of another container
  - Arbitrarily complex transformations
  - Often custom functions
- Examples:
  - 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 n in numbers:
...     mean += n / len(numbers)
```

- 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