Effective Programming Practices for Economists

Basic Python

`if` conditions

Janoś Gabler and Hans-Martin von Gaudecker

Topics

- if, elif and else
- More on Booleans
- Filtering loops

Motivation

- So far, all of our instructions in Python were very explicit
- There was no way of reacting to different situations:
  - Collecting elements of a list that fulfill a condition
  - Doing different things for different types of variables
  - ...
- This is what if conditions are for

Conditional statements

Example: clipping a number

```
>>> number = -3.1

>>> if number < - 3:
...     clipped = -3.
... elif number > 3:
...     clipped = 3.
... else:
...     clipped = number
>>> clipped
-3.0
```

- `if`, `elif` and `else` are special keywords
- End each condition with a `:`
- What happens if that condition is True needs to be indented by 4 spaces and can span one or multiple lines
- The code related to False conditions is skipped
- elif means else-if

More on booleans

```
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(1)
True
>>> bool([])
False
>>> bool([1, 2, 3])
True
>>> bool("")
False
>>> bool("abc")
True
```

- What is not a boolean can be converted to a boolean
- This conversion happens implicitly after if and elif
- Can be useful but and elegant but might compromise readability
- Rules of thumb:
  - 0 is False-ish, other number are True-ish
  - Empty containers are False-ish
  - Non-empty containers are True-ish

Filtering loops

```
>>> names = ["Guido", "Raymond", "T
>>> names_with_i = []
>>> for name in names:
>>>     if "i" in name:
>>>         names_with_i.append(name)
>>> names_with_i
['Guido', 'Tim']
```

- Can filter lists based on properties of items
- Can filter dictionaries based on properties of keys and/or values
- Example usecases: