Effective Programming Practices for Economists

Basic Python

Functions

Janoś Gabler and Hans-Martin von Gaudecker

Topics

- Anatomy of functions
- Examples of functions
- Why functions are important!
- A few guidelines

Anatomy of Python functions

- Start with the `def` keyword
- Name is `lowercase_with_underscores`
- There can be one or several parameters (a.k.a. arguments)
- Function body is indented by 4 spaces and can have one or several lines
- Inside the body you can do everything you have seen so far!

Example: CRRA Utility function

```
>>> def utility_crra(c, gamma=1.5):
...     out = c ** (1 - gamma) / (1
...     return out

>>> utility_crra(1.0)
-2.0

>>> utility_crra(c=1.0, gamma=1.5)
-2

>>> utility_crra(c=1.0, gamma=0.0)
1.0
```

- You can assign default values for arguments
- Function calls work with positional and keyword arguments
- Use keyword arguments for any function with more than one argument!

Defining functions like a pro is the most important skill to become a good Python programmer!

Why functions are important

- Help to re-use code and avoid duplication
- Help to structure code and reduce cognitive load
- Make individual code snippets testable
- Help to make your projects more reproducible
- Unlock the power of functional programming concepts
- Are also the basis for good object oriented code

```
# bad example
>>> global_message = "Hello {}!"

>>> def greet_with_global(name):
...     print(global_message.format

>>> greet_with_global("Guido")
Hello Guido!

# solution 1: define inside functio
>>> def greet(name):
...     message = "Hello {}!
...     print(message.format(name))
>>> greet("Guido")
Hello Guido!

# solution 2: pass as argument
>>> def greet_explicit(name, messag
...     print(message.format(name))

>>> greet_explicit("Guido", "Hello
```

Do not modify mutable arguments

```
>>> def append_4(some_list):
...     some_list.append(4)
...     return some_list

>>> a = [1, 2, 3]
>>> append_4(a)
[1, 2, 3, 4]

>>> a
[1, 2, 3, 4]

# better solution
>>> def append_4(some_list)
...     out = some_list.copy()
...     out.append(4)
...     return out
```

- Arguments are passed by reference, i.e. without making a copy
- Make sure, functions do not modify mutable arguments!
  - Make copies
  - Avoid changing objects in the first place