Effective Programming Practices for Economists

Basic Python

Lists, tuples and sets

Janoś Gabler and Hans-Martin von Gaudecker

Contents

Lists

```
>>> a = [1, 2, 3]
>>> type(a)
<class 'list'>

>>> a.append(4)
>>> a
[1, 2, 3, 4]

>>> a[0] = "bla"
>>> a
['bla', 2, 3, 4]

>>> len(a)
4
```

- Created with square brackets
- Definition: Mutable sequence of objects
  - **mutable**: Can change it after creation
  - **sequence**: An ordered collection
  - **of objects**: Items can consist of anything
- Lists are used a lot!
- Highly optimized for fast appending!
- `len` works for all collections

Tuples

```
>>> a = (1, 2, 3)
>>> type(a)
<class 'tuple'>

>>> b = (1)
>>> type(b)
<class 'int'>

>>> c = (1,)
>>> type(c)
<class 'tuple'>

>>> d = 2,
>>> type(d)
<class 'tuple'>
```

- Created with round brackets
- Definition: Immutable sequence of objects
  - **immutable**: Cannot change after creation
- Single element tuples need a comma
- But sometimes you don’t need the brackets!
- Less flexible than lists, less common
- Somewhat unfair:
  - immutable: often helps to prevent bugs
  - hashable: can use in more locations

Selecting elements

```
>>> a = [1, 2, 3, 4, 5]
>>> a[1]
2

>>> a[1: 2]
[2]

>>> a[:2]
[1, 2]

>>> a[2:]
[3, 4, 5]

>>> a[-1]
[5]
```

- Selecting elements is the same for lists, tuples, and other sequences
- Indexing starts at 0
- Upper index of slices is not included
- lower and upper index can be left implicit
- negative indices start from the end

Sets

```
>>> a = {3, 2, 1, 3}
>>> a
{1, 2, 3}
```

```
>>> b = {}
>>> type(b)
<class 'dict'>

>>> c = set()
>>> type(c)
<class 'set'>
```

- Created with curly braces
- Definition: Mutable unordered collection of unique hashable items
  - **unordered**: order is undefined and can change
  - **unique**: duplicates are dropped at creation
  - **hashable**: ≈ immutable
- Empty set can not be created with curly braces

Set operations

```
>>> a = {1, 2, 3}
>>> b = {1, 5}

# membership checking
>>> 2 in a
True

# union
>>> a | b
{1, 2, 3, 5}

# intersection
>>> a & b
{1}
```

- Sets are highly optimized for:
  - membership checking
  - union
  - intersection
- Lists and tuples would be much slower