

# **Effective Programming Practices for Economists**

## **Software engineering**

### **Defining custom containers**

Janoś Gabler and Hans-Martin von Gaudecker

# Some drawbacks of dictionaries

```
>>> student = {  
...     "first_name": "Janos",  
...     "last_name": "Gabler",  
...     "email": "janos@uni-bonn.de",  
... }  
  
>>> student["frist_name"]  
-----  
KeyError Traceback (most recent call last)  
...  
KeyError: 'frist_name'
```

- Typos lead to runtime errors
- Mutable
- Hard to document/know which keys should be there
- No autocomplete for keys

# NamedTuples

```
>>> from typing import NamedTuple
```

```
>>> class Student(NamedTuple):
```

```
...     first_name: str
```

```
...     last_name: str
```

```
...     email: str
```

```
>>> student = Student(
```

```
...     first_name="Janos",
```

```
...     last_name="Gabler",
```

```
...     email="janos@uni-bonn.de",
```

```
... )
```

```
>>> student.first_name
```

```
'Janos'
```

- Typos can be detected by an IDE
- Immutable
- Easy to document/know which attributes are there
- Autocomplete for attributes works

# Dataclasses

```
>>> from dataclasses import dataclass
```

```
>>> @dataclass
... class Student:
...     first_name: str
...     last_name: str
...     email: str
```

```
>>> student = Student(
...     first_name="Janos",
...     last_name="Gabler",
...     email="janos@uni-bonn.de",
... )
```

```
>>> student.first_name
'Janos'
```

- Same advantages as as [NamedTuple](#)
- Mutable by default but can be made immutable
- Many powerful options:  
[Documentation](#)

# Reminder

- Dictionaries are awesome! One of the most optimized data structures you can imagine.
- You'll need to learn when to use
  - dicts
  - NamedTuples
  - dataclasses