Python for Beginners

Object Orientation 2

Inheritance

- Follow the principle: write once and reuse

```
# the base class (or super class, parent class)
class Person:
    def __init__(self, first, last):
        self._firstname = first
        self._lastname = last

    def say_hello(self):
        print(f"Hello, my name is {self._firstname} {self._lastname}!\n"
              f"I'm an object of {self.__class__}")

    def __str__(self):
        return self._firstname + ' ' + self._lastname
```

Inheritance (2)

```
# the derived subclass with inherited fields and methods
class Employee(Person):

    def __init__(self, first, last, income):
        # reuse the constructor of the super class
        # to initialize firstname and lastname
        super().__init__(first, last)
        # an additional field
        self._income = income

    # overriding the inherited __str__ method
    def __str__(self):
        # reuse the __str__ method from the super class
        return super().__str__() + ', Income: ' + str(self._income) + '€'
```

Inheritance (3)

```
x = Person('Marge', 'Simpson')
y = Employee('Homer', 'Simpson', 3870)

x.say_hello()
y.say_hello()
```

→

Hello, my name is Marge Simpson!
I'm an object of <class '__main__.Person'>
Hello, my name is Homer Simpson!
I'm an object of <class '__main__.Employee'>

```
print(x)
print(y)
```

→

Marge Simpson
Homer Simpson, Income: 3870€

Class Fields and Class Methods

```
class Person:
    # this field belongs to the class (=all objects together)
    _counter = 0

    def __init__(self, first, last):
        self._firstname = first
        self._lastname = last
        Person._counter += 1 # access the class field via the class not via self!!

    def say_hello(self):
        print(f"Hello, my name is {self._firstname} {self._lastname}!\n"
              f"I'm an object of {self.__class__}")

    def __str__(self):
        return self._firstname + ' ' + self._lastname

    # this method can be called from the class and doesn't need an object to be called
    @classmethod
    def get_counter(cls):
        return cls._counter
```

Class Fields and Class Methods (2)

```
# create two objects of class Person
x = Person('Marge', 'Simpson')
y = Employee('Homer', 'Simpson', 3870)


# call the class method from the class
print(Person.get_counter())
```

→ 2

Deleting Objects

- Add this method to class Person:

```
def __del__(self):
    print(f"Oh dear, I'm gonna be destroyed! {self.__class__}")
```

- Create two objects:

```
x = Person('Marge', 'Simpson')
y = Employee('Homer', 'Simpson', 3870)
```

- Delete the object x

```
del x

# this should be the last line of the program
print('End of program')
```

→

Oh dear, I'm gonna be destroyed! <class '__main__.Person'>
End of program
Oh dear, I'm gonna be destroyed! <class '__main__.Employee'>

Type Hinting

Typ Hinting is an optional feature that

- Helps Type Checkers
- Helps with documentation
- Helps IDEs develop more accurate and robust tools

With Type Checkers:

- Find bugs sooner
- The larger your project the more you need it

Type Hinting Examples

You can add types to parameters and returns

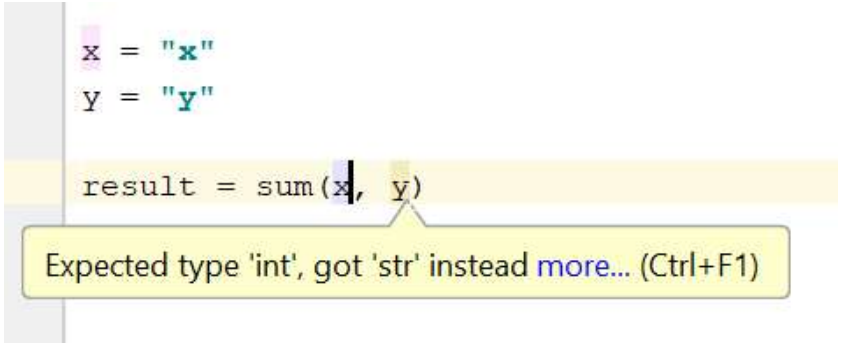
```
def sum(a: int, b: int)->int:
    return a + b
```

```
x = "x"
y = "y"

result = sum(x, y)

print(result)
```

→ xy



Type hinting doesn‘t prevent bugs but PyCharm offers type checking

The typing module

```
from typing import Dict, List

# A dictionary where the keys are strings and the values are ints

name_counts: Dict[str, int] = {
    "Adam": 10,
    "Guido": 12
}

# A list of integers

numbers: List[int] = [1, 2, 3, 4, 5, 6]

# A list that holds dicts that each hold a string key / int value

list_of_dicts: List[Dict[str, int]] = [
    {"key1": 1},
    {"key2": 2}
]
```