

O Object O Oriented P Programming

S1 = Python with Bianca

S2 = Java with Bryan

DT228(TU856)/DT282(TU858) - 2

Inheritance

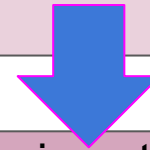
Objectives

- Revise OOP basic principles
- Discover inheritance
- Design with OOP Principles: UML language

Object vs Class vs Instance of Class

Object

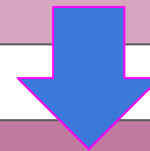
- Abstract idea, design principle
- Example: The idea of a **House**...probably has doors and windows



Class

- Implementation of the abstract idea in actual code, making it the blueprint of how to implement a **House**
- In Python starts with `class` flag
- `window` and `door` might become attributes that could be of value `"open"` or `"closed"`

Sometimes called an object.



Instance of a class

- A specific **House** with a variable name like `my_house`
- With dot notation you access states (attributes) and behaviour (methods) of `my_house`

Sometimes called an object.

Code Examples

- Everything in Python is a class
- Class consists of 2 parts: a header and a body
 - Class name can be followed by other class names, this means it inherits from those (more later)
- Body is indented list of statements
- Class name must start with a letter or an underscore,
 - the name can only contain letters, underscores and numbers

Indentation uses 4 spaces (tab).

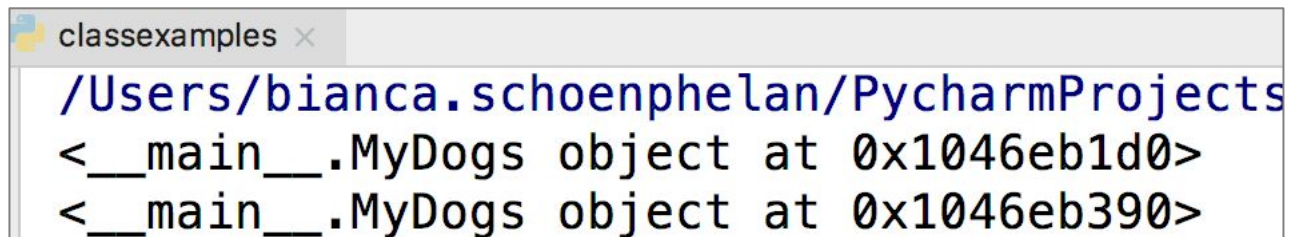
```
6  
7  
8  
9  
class MyDogs:  
    pass
```

Python style guide: search online for PEP8, recommends camel case naming for classes and `_` for methods.

Using the Example Class

```
6
7 class MyDogs:
8     pass
```

```
10
11 a = MyDogs()
12 b = MyDogs()
13 print(a)
14 print(b)
```



```
classexamples x
/Users/bianca.schoenphelan/PycharmProjects
<__main__.MyDogs object at 0x1046eb1d0>
<__main__.MyDogs object at 0x1046eb390>
```

- Two new objects a and b have been instantiated from the class MyDogs()
- Looks like a function call, Python knows what to do

[5]

Now with Attributes

```
a.age = 5
a.colour = 'black'

b.age = 2
b.colour = 'white'

print('Dog a: ', a.age, a.colour)
print('Dog b: ', b.colour, b.age)
```

```
classexamples
/Users/bianca.scho
Dog a:  5 black
Dog b:  white 2
```

- First empty class
- Then class with attributes
- Dot notation:
 - `<object>.<attribute>=<value>`
- Value can be many things, a python primitive, a built in data type, another object, even a function or another class

[5]

Behaviours

```
class MyDogs:  
    def bark(self):  
        print('wuff wuff')
```

```
rex = MyDogs()  
rex.bark()
```

```
classexamples x  
/Users/bianca.  
wuff wuff
```

- Starts with keyword `def` followed by space and the the name of the function
- Parentheses for parameter list
- Terminated by colon :
- Next line is indented to contain the statements of the method

All methods have one required argument: `self`

- This is a convention, but never seen anyone not use it
- It's a reference to the object that the method is being invoked on

Functions/Methods

- Functions in Python:

This function is **not** part of a class.

Def statement

Function name

Parameter name(s)

Python also offers `*args` and `**kwargs` as parameters to a function. We'll talk about it another time.

```
def hello_world(say_something):  
    return say_something
```

Body of the function

Return statement

Value that is returned.

```
print("We say: ", hello_world("I love programming"))
```

output

```
/usr/local/bin/python3.7 "/Vo  
We say: I love programming
```

Functions/Methods in a Class

```
 class HelloWorld:
```

```
    def hello_world(self, say_something):  
        return say_something
```

Attaches the method to the function.
Pycharm adds this automatically.

```
hi = HelloWorld()
```

```
print(hi.hello_world("I love programming"))
```

- Create an instance of the object first
- Then call the method with dot notation.

Functions/Methods some observations

- Python allows code outside of methods and classes, but in the same file, **which should be avoided**
- Python allows code inside classes that are not part of methods, **which should be avoided**. **Put code into**

methods inside the class!

- `if name == "__main__":`
`main()`

You might find this syntax online. Don't use it (in this course)!

- should be **avoided** by your classes
- this is for classes that may become modules and allows you to specify what part of the class to use for the module. You are not going to programme “modules” as part of this course. **Don't use this syntax!**

OOP Principles

Main OOP Principles

1. Encapsulation (<- last week, aka what can be accessed or seen from outside, more in tutorial tomorrow on how to use as part of inheritance)
2. Abstraction
3. **Inheritance**
4. Polymorphism

[1]

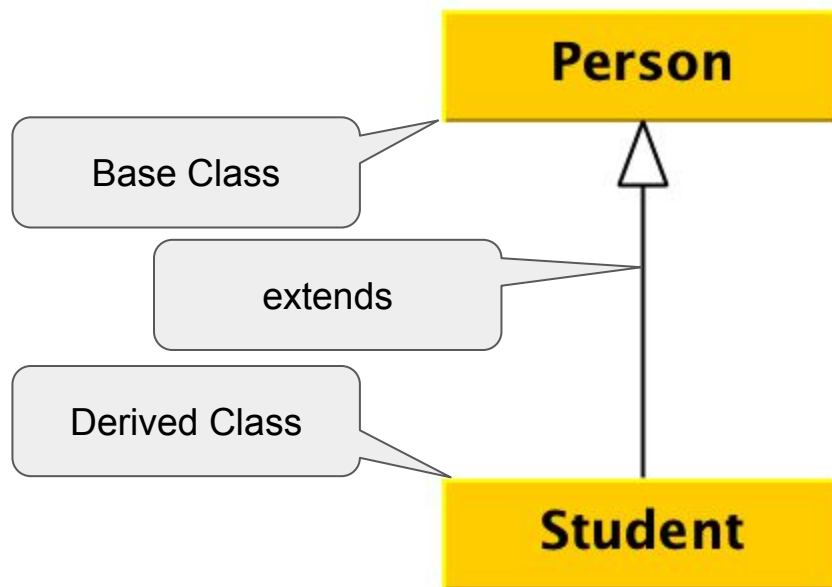
Inheritance

- Inheritance allows us to reuse code
 - See for an example the card game lab
- We create a class and it is allowed to use all the methods and attributes from another class
- This essentially creates a hierarchy from parent class down to child classes, which is often illustrated using a tree structure
- It's a big part of what makes an object-oriented programming language OOP in the first place

The class we inherit from is called the parent class or the superclass.

Python supports multiple inheritance.

Inheritance Example



- **Student** inherits from **Person**
- What do they inherit?
 - Methods
 - Attributes

Inheritance models an **is-a** relationship.
For example, a Student **is-a** Person.

Inheritance Example

```
class Person:
    def __init__(self):
        self.name = 'Bianca'
        self.age = 18
```

```
def get_age(self):
    print(self.name, "is", self.age, "years old.")
```

```
p = Person()
p.get_age()
```

```
/usr/local/bin/python3.7
Bianca is 18 years old.
```

It really only makes sense to create a child class if the child class is going to have some **unique** behaviours or attributes that the parent class did not need.

```
class Student(Person):
    pass
```

```
s = Student()
s.get_age()
```

```
/usr/local/bin/python3.7 "/Vol
Bianca is 18 years old.
Bianca is 18 years old.
```


Liskov's Substitution Principle

- In a computer programme, if S is a subtype of T, then objects of type T may be replaced with objects of type S without altering any of the desired properties of the programme

Meaning: use the child class.

Inheritance Unique Behaviours

```
class Person:
    def __init__(self):
        self.name = 'Bianca'
        self.age = 18

    def get_age(self):
        print(self.name, "is", self.age, "years old.")

p = Person()
p.get_age()
```

```
class Student(Person):
    def __init__(self):
        self.studentID = 1234

    def enrol_into_uni(self):
        print("I'm a student now")

    def get_studentID(self):
        print("My student ID is:", self.studentID)
```

Name and age not populated. However, if they are arguments of the init function, they have to be listed.

```
s.get_studentID()
```

```
...ocal/bin/python3.7
My student ID is: 1234
```

Inheritance getting parent's attributes

```
s.get_age()
```

```
File "/Volumes/GoogleDrive/My Drive/Vorlesungen/DT228+ 0  
print(self.name, "is", self.age, "years old.")  
AttributeError: 'Student' object has no attribute 'name'
```

Produces an error as this is undefined. Needs a change to the Student that calls the parent init! This is because the attribute wasn't set. Set the attribute and you can use the method.

```
class Student(Person):  
    def __init__(self, name, age, studentID):  
        self.studentID = studentID  
        super().__init__(name, age)  
  
    def enrol_into_uni(self):  
        print("I'm a student now")  
  
    def get_studentID(self):  
        print("My student ID is:", self.studentID)
```


```
s = Student("Anna", 19, 123)  
s.get_age()  
s.get_studentID()
```

```
Anna is 19 years old.  
My student ID is: 123
```

super() in Python

- Python 2 super syntax is much more complicated
- A call to super can be made inside **any** method, not just `__init__`
- All methods can be modified via overriding and calls to super
- Calls to super can happen at any stage of a method, not just as the first line

OOP Principle: Inheritance

- Different kinds of objects have certain things in common
 - Example: trucks, cars, motorbikes are all motorised vehicles
 - States: current speed, current gear, cc value...
 - Behaviours: drive, stop, change gear
 - Some are executed slightly differently, depending on vehicle  **method overriding**

OOP allows classes to inherit certain traits, aka common states and behaviours, and implement their own versions if required.

[1]

Example Method Overriding: get_age()

```
class Student(Person):  
    def __init__(self, name, age, studentID):  
        self.studentID = studentID  
        super().__init__(name, age)  
  
    def get_age(self):  
        print(self.name, "does not want to say an age")
```

```
s = Student("Anna", 19, 123)  
s.get_age()
```

```
tutorial_wk9 x  
/usr/local/bin/python3.7 "/Volume  
Anna does not want to say an age  
Method not found: 123
```

- The method has the same signature (aka. name and amount of arguments)
- The inside of the method does something different than in the parent class

Multiple Inheritance

- A class inherits from more than one method
- Less useful than it sounds
- Many experienced programmers advocate against it
- Often results in hard to maintain and understand code
- Gets messy if we try to call methods from the parent class
 - What does super refer to?
 - How do we know the order to call them in?
- Argument passing also tricky, Python uses 'pseudo pointer' called `**kwargs`
- Diamond problem (who's the parent?) - more in a later class

Summary

- ★ **Object Oriented Principles**
- ★ **OOP in Practice with Python**
- ★ **Method overriding**
- ★ **Parent init**



References

1. Concepts of objects, Oracle corp,
<https://docs.oracle.com/javase/tutorial/java/concepts/object.html>, accessed Oct 2018.
2. Python – overwriting methods, <https://pythonprogramminglanguage.com/overriding-methods/>,
accessed Oct 2018.
3. Python Course, Multiple Inheritance,
https://www.python-course.eu/python3_multiple_inheritance.php, accessed Oct 2018.
4. Python 3: Object-oriented programming, 2nd edition, Dusty Phillips, 2015, Packt Publishing.
5. https://www.w3schools.com/python/python_classes.asp, accessed Oct 2019
6. Python Course, Inheritance, https://www.python-course.eu/python3_inheritance.php,
accessed 5-11-2019.