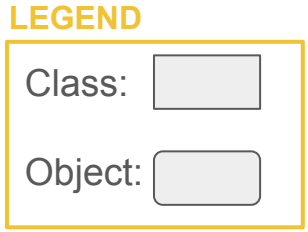
9. Classes

Building your own Data Types

Object oriented programming

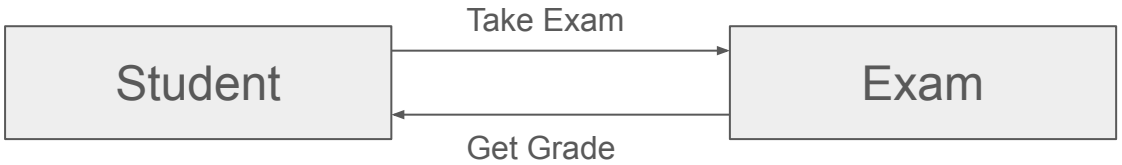


What is object-oriented programming?

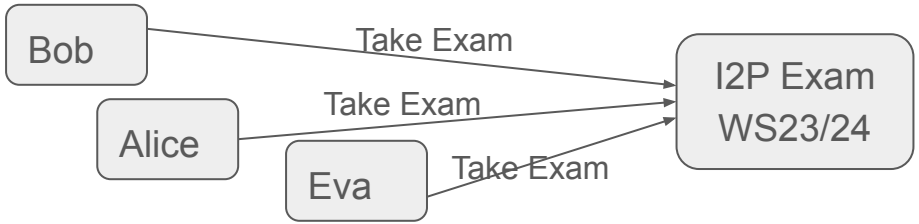
Remember **primitive** data types and there are more **complex** ones?
→ You can also **create** your **own** complex **objects**!

Object-oriented programming (OOP):

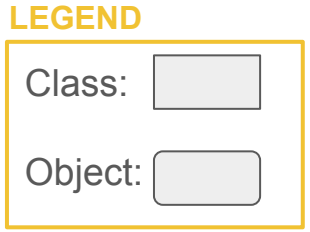
- Model your problem as **objects** that **interact** with each other



- Many objects of same **type** (same ***class***)

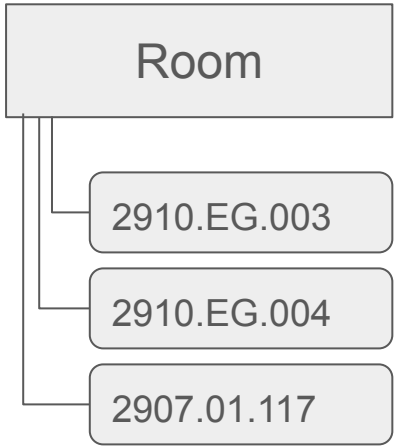


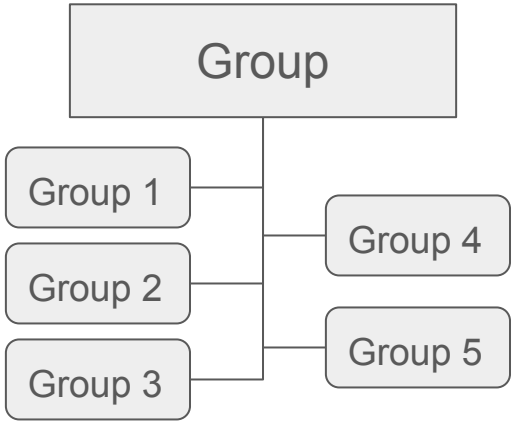
What are Classes?

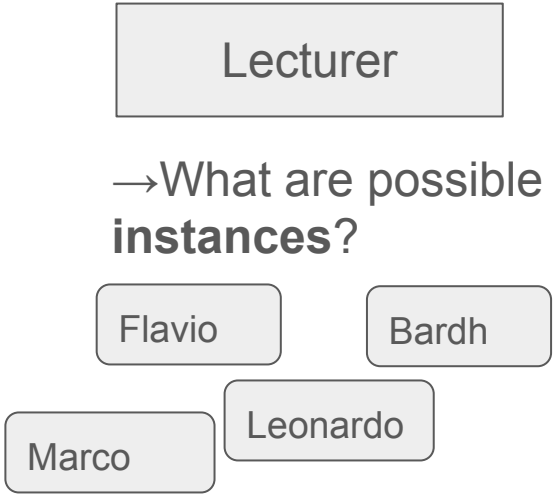


Classes are the **fundamental** concept in OOP. They are the **blueprint** for your own objects.

Creating an object from a class is called ***instantiation***.







LEGEND

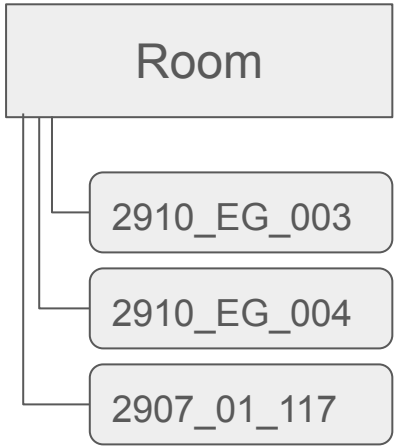
Class:

Object:

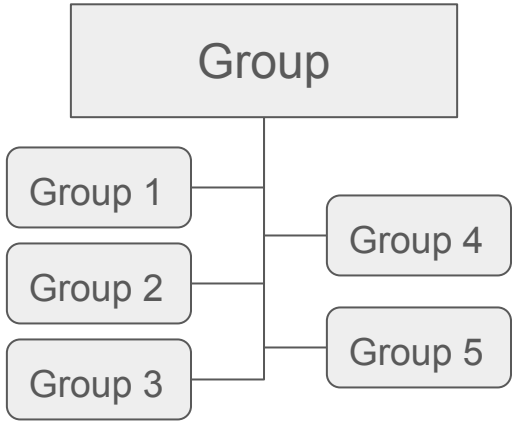
Attributes and Methods

Classes usually specify **attributes** (data) and **methods** (functions) that all of their instances have. Attributes can be thought of as the **object's** specific **state** and methods as the **object's behavior.**

→ What would be some **attributes** for the below classes?

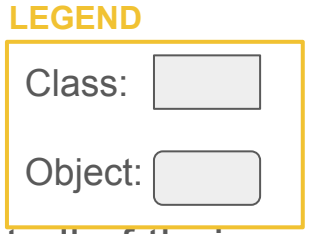


Seats
Floor
Building
Power plugs
…



Size
Week(time)
Lecturer
Room
Students
Grades
…

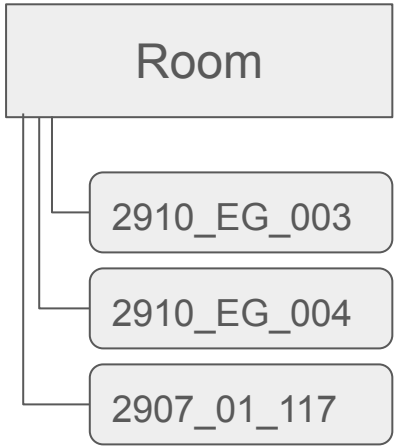
→ What could be the data type of the **attributes**?



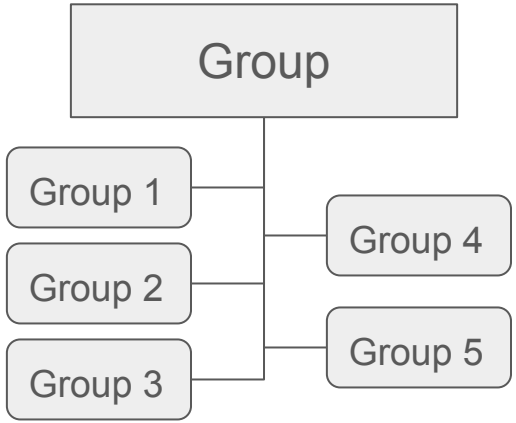
Attributes and Methods

Classes usually specify **attributes** (data) and **methods** (functions) that all of their instances have. Attributes can be thought of as the **object's** specific **state** and methods as the **object's behavior.**

→ What would be some **methods** for the below classes?



isFree()
book()
clean()
turnOnLights()
turnOffLights()
…



addStudent()
removeStudent()
correctExams()
setLecturer()
getTime()
…

→ What could be the **input** and **output** of these **methods**?

Modeling complex Problems

It can get more complex

Building

Group

Study Program

Slides

Lecturer

Student

Exercises

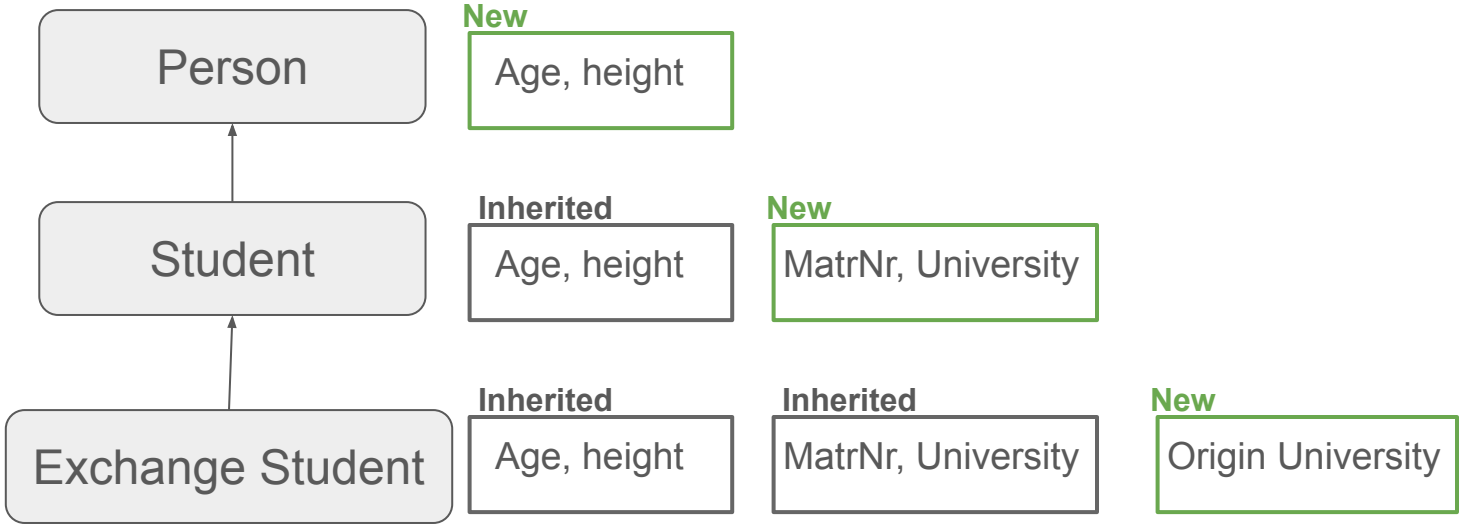
Room

Exam

→ What could be possible question/problems we want to answer?

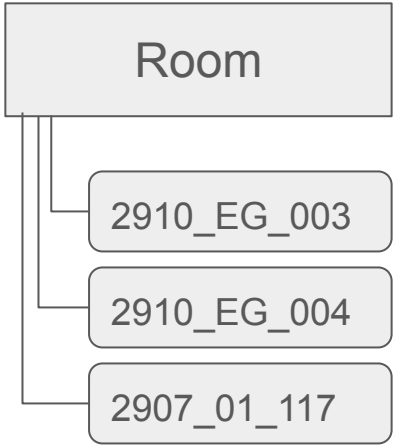
Inheritance

- Same attributes and function can be **copied** from **similar classes**
- Example: The Attributes and functions of a **Person** (age, height, etc.) also apply for a **Student** as it is a subcategory of a Person.



Classes versus Databases

Why use objects and not tables?



Room	Seats	Floor	Building	Power Plugs
2910_EG_003	48	0	2910	48
2910_EG_004	32	0	2910	12
2907_01_117	20	1	2907	18

- **Many problems** can be solves/modeled with **tables/databases** but:
- OOP can be more **flexible** (inheritance, modularity, reusability, etc.)
- OOP is more **intuitive** for many real world problems (**entity based** problems)

Real world example - Neural Networks

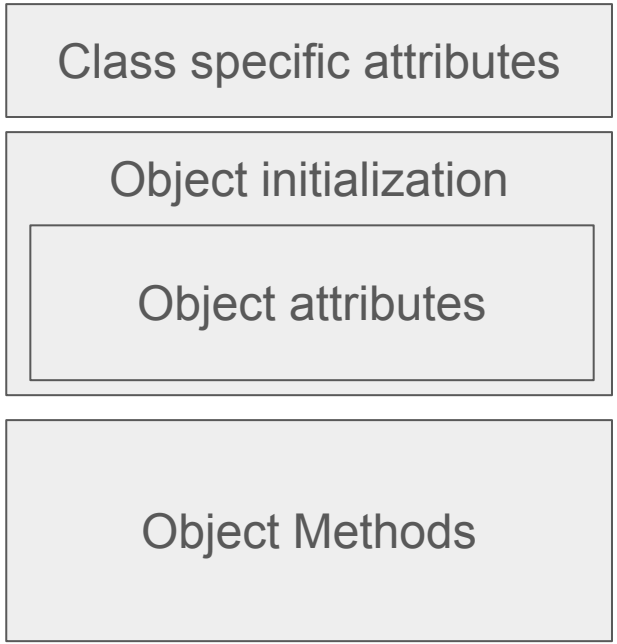
Creating **complex objects** that need **many modifications** → OOP!

```
model = NeuralNetwork(...)
model.addLayer(...)
model.modifyLayer(...)
model.initializeWeights(...)
model.trainOn(trainingSet)
model.evaluateOn(validationSet)
model.fineTune(specialSet)
model.evaluateOn(testingSet)
model.visualizeResults(...)
model.uploadPlots(website)
```

Classes in Python

Defining a Class

```
class Person:
```



Use the keyword `class` followed by the class name.

This class has **no useful properties** though → It's a quite useless Datatype!

We might want to also add:

- Attributes
- Methods
- Special initialization steps
- Class specific attributes
- etc.

The __init__ Method

The `__init__` method is a special method that is **called** when an **object** is **created** (instantiation).

It is used to **initialize** the object's **attributes**. The first parameter has to be `self`.

You can also add **other code** that shall be executed during the object instantiation.

```
def __init__(self, parameter1, parameter2, …):
        self.attribute1 = parameter1
        self.attribute2 = parameter2
         …
         [other code to be executed during object initialization]
```

Using __init__

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

Use the keyword `class` followed by the class name.

Use `__init__` to take `name` and `age`. Both are the **initial object attributes**.

Creating instances of classes

To **create an object** of a class (*instantiate*), use the class name followed by **parentheses**.

Within the parentheses, the initial **parameters** must be **provided**. You are basically calling the `__init__` function here!

```
object = ClassName(parameter1, parameter2, …)
```

Creating an Object with Attributes

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

alice = Person("Alice W.", 45)
bob = Person("Bob M.", 36)
```

Use the keyword `class` followed by the class name.

Use `__init__` to take `name` and `age`. Both are the **initial object attributes**.

Create a **new instances** of the `Person` class.

Note: `__init__` takes **3** arguments, but we **only** need to provide **2 arguments** during instantiation! This is because `self` is **provided automatically**!

Accessing Attributes and Methods

To access an object's attributes and methods, use **dot notation**.

Examples:

- `alice.name`
- `bob.age`
- `print(alice.age)`
- `bob.name = "Bobby"`
- `meanAge = (alice.age + bob.age)/2`

Exercise

What is the issue with the following code:

```
class MyClass
    def __init__(self, x):
        self.x = x
```

→ Missing “:” after class definition

Exercise

What is the issue with the following code:

```
class MyClass:
    def __init__(x):
        self.x = x
```

→ Missing “self” parameter

Exercise

What is the issue with the following code:

```
class MyClass:
    def __init__(self, x):
    self.x = x
```

→ Incorrect Indentation

Exercise

What is the issue with the following code:

```
class MyClass:
    def __init__(self, x):
        self.x = x

obj = MyClass
print(obj.x)
```

→ Incorrect Instantiation (missing parenthesis)

Exercise

What is the issue with the following code:

```
class MyClass:
    def __init__(self, x):
        self.x = x

obj = MyClass()
print(obj.x)
```

→ Incorrect Instantiation (missing parameter)

Exercise

What is the issue with the following code:

```
class Animal:
  def __init__(self, name, legs):
      self.name = name

obj = Animal("Dog", 4)
print(obj.legs)
```

→ Attribute `legs` not set

Exercise 1

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

alice = Person("Alice W.", 45)
bob = Person("Bob M.", 36)
```

1. Copy the code and print the age of bob (using the dot notation)
2. Create an if-statement that prints the name of the oldest of the two Persons
3. Create three other Persons. Make a list called `people` with all 5 Persons.
4. Make a loop that finds and prints the youngest person's name

Defining Methods

To add an **method** to a class, define a **function** in the **class body**. Just like we did with `__init__`.

```
class className:
    …
    def methodName(self, parameter1, parameter2, ...):
        [method Code]
        …
```

Note:

- Your method **must take `self`** as argument (just like __init__ does).
- You can access the objects attributes and methods though `self.`

Object Methods

```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def printData(self):
        print(f"{self.name}: {self.age}")

alice = Person("Alice W.", 45)
alice.printData()
```

```
>>> Alice W.: 45
```

Exercise

What is the issue with the following code:

```
class Animal:
    def __init__(self, name, legs):
        self.name = name
        self.legs = legs
    def removeLeg():
        self.legs -= 1
obj = Animal("Dog", 4)
obj.removeLeg()
```

→ Missing `self` in method definition

Exercise

What is the issue with the following code:

```
class Animal:
    def __init__(self, name, legs):
        self.name = name
        self.legs = legs
    def changeName(self, newName):
        name = newName
obj = Animal("Dog", 4)
obj.changeName("Doggo")
```

→ Missing `self` in attribute access

Exercise 2 *(Folder 9 ex_2.py)*

1. Write a class called `Student` with the attributes `name` (str) and `studyProgram` (str)
2. Create three instances. One for yourself, one for your left neighbour and one for our right neighbour.
3. Add a method `printInfo` that prints the `name` and `studyProgram` of a `Student`. Test your method on the objects!
4. Modify the code and add `age` and `gender` to the attributes. Modify your printing methods respectively too.

EXOTIC

Class Attributes and Methods

It is also possible that the **class itself** (independent of its instances) has **attributes** and **methods**!

```
class Person:
    personCount = 0
    def __init__(self, name):
        self.name = name
        self.personCount += 1
alice = Person("Alice W.")
bob = Person("Bob M.")
print(Person.personCount)
```

A variable defined in the class body is a **class variable**.

Inside the class, they can be accessed using `self`

Outside the class, they must be accessed using the **class name**

<u>Question</u>: What is the **output** of this code?

EXOTIC

Class Attributes and Methods

A **class method** is defined as a normal method but takes `cls` not `self` as mandatory first parameter. `cls` allows to access everything that is defined on **class level** (not object level)

```
class Student:
    studentGrades = []
    def __init__(self, studentName, grade):
        self.name = studentName
        self.studentGrades.append(grade)
    def getGroupAverage(cls):
        avg = sum(cls.studentGrades) / len(cls.studentGrades)
        return avg
```

Question: Can we call `Group.getGroupAverage()` if there is no instance of Group?

Question: Can we access `cls.name` inside `getGroupAverage()`

Recap

- **Classes** are created with the keyword `class`
- **Objects** are **instances** of classes
- **Attributes** are **variables** that belong to an object and **methods** are **functions** that belong to an object
- During object **instantiation**, the method `__init__()` is called:
  - `__init__()` must take `self` as first argument
  - It creates/sets (initial) **attributes** (`self.<attribute> = <value>`)
  - It can also execute **other code**
- Functions and attributes are **selected** with the **“.”** notation → `alice.name`
- Methods also require `self` as first argument
  - Inside the method’s code you can use `self`. To access the objects attributes and methods → `self.name`

Exercise 3 *(Folder 9 ex_3.py)*

Given is the class `Animal`. For each task, **test your changes**!

1. Create two realistic instances of `Animals`
2. Print the `name` of each object
3. Change the amount of legs of one object using the dot notation
4. Add a method `setLegs()` to set the `legs` of an object and repeat task 3 but this time using the method.
5. Add a method `getLegs()` to return the amount of `legs`
6. Add a method named `printInfo` that prints all attributes of the `Animal`

Exercise 4 *(Folder 9 ex_4.py)*

1. Write a new class called `Food`, it should have `name`, `price` and `description` as attributes.
2. Instantiate at least three different foods you know and like.
3. Create a new class called `Menu`, it should have a list (of `Food`s) as attribute. `__init__` should take a list of `Food`s as optional parameters (default=`[]`)
4. Create a `addFood()` and `removeFood()` for the `Menu`
5. Create a few new `Food` instances. Add each to the `Menu` using the respective Method.
6. Add a method `printPrices()` that list all items on the Menu with their prices.
7. Add a `Menu` method `getAveragePrice()` that returns the average Food price of the `Menu`