University of
Applied Sciences

Programming 2

Introduction to OOP

Dr. Dhungana
Dr. Klikovits
SS2024 • Krems

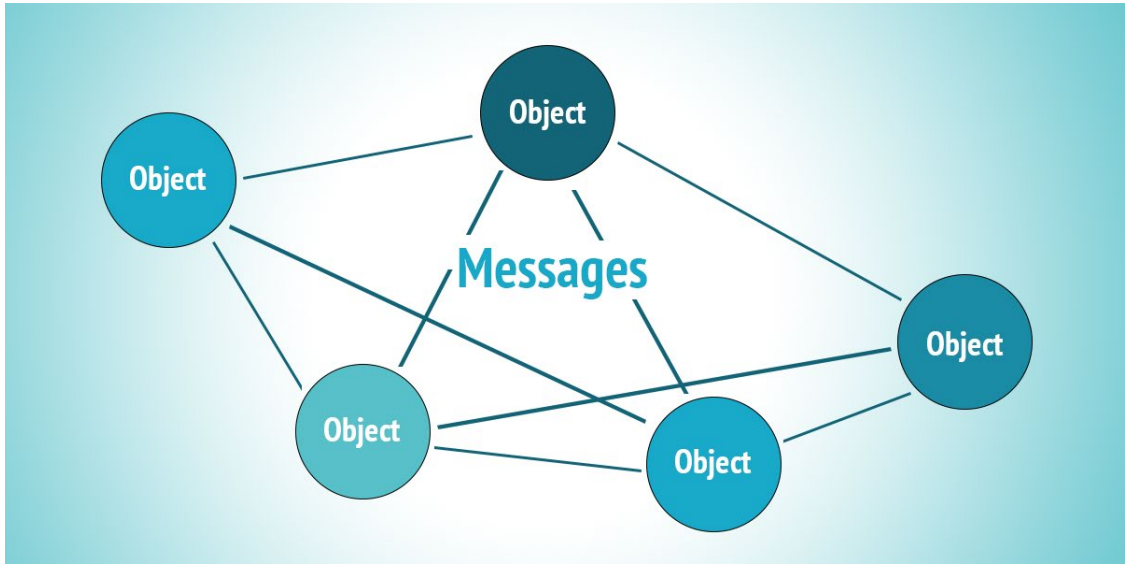
Object-orientation

Python is an object-oriented programming language

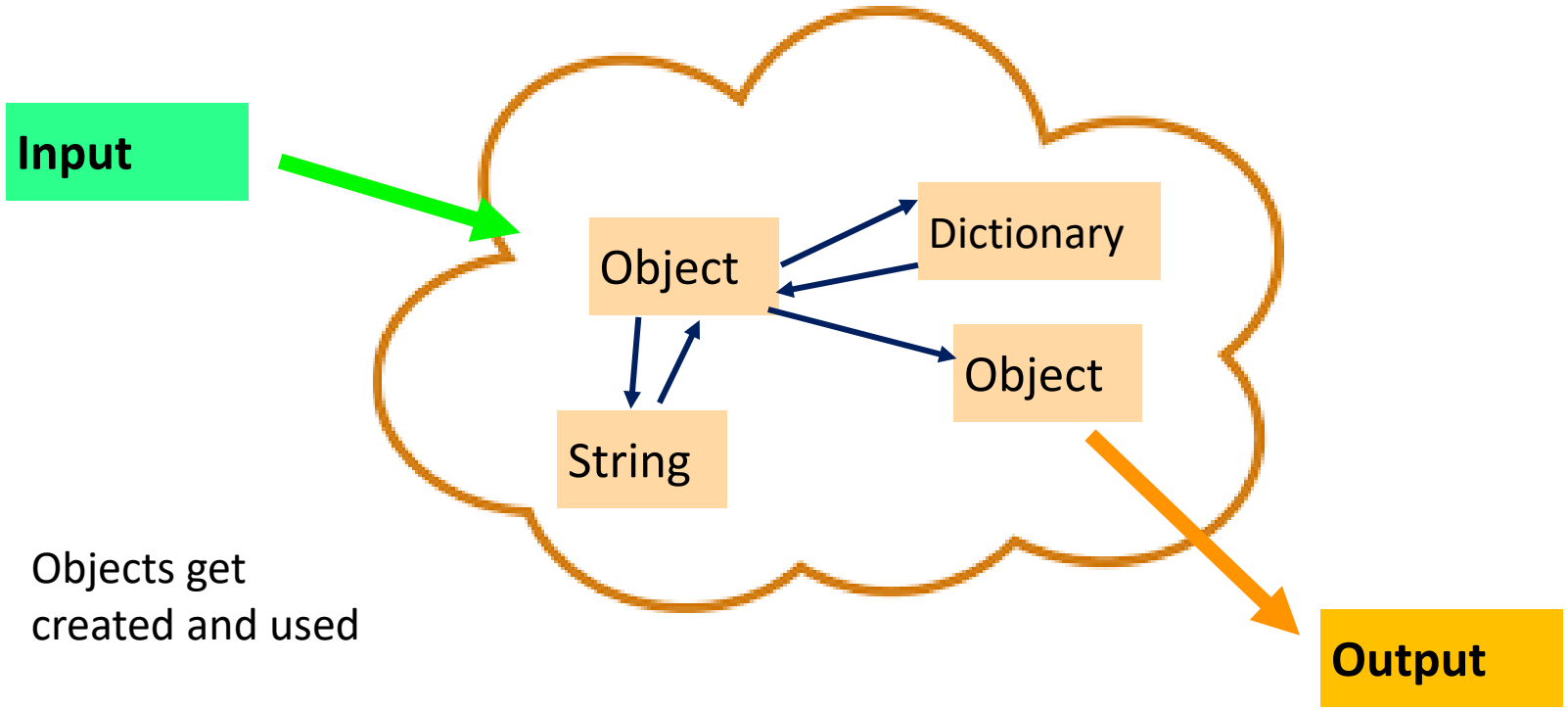
A program is made up of many cooperating objects

Instead of being the “whole program” - each object is a little “island” within the program and cooperatively working with other objects.

A program is made up of one or more objects working together - objects make use of each other’s capabilities



Objects interact with each other!



Objects get
created and used

Procedural vs. Object-oriented

Classes & Objects

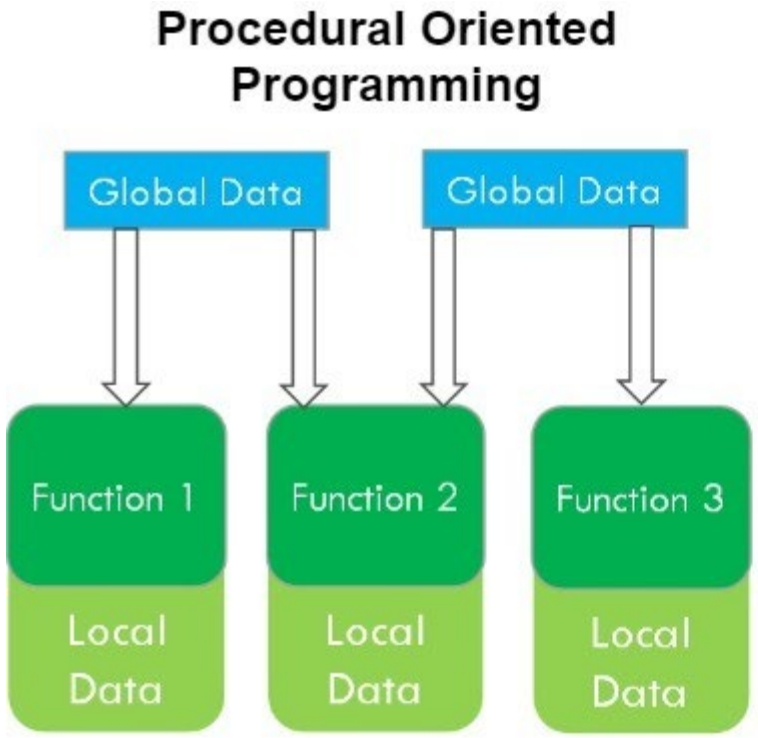
Data Abstraction

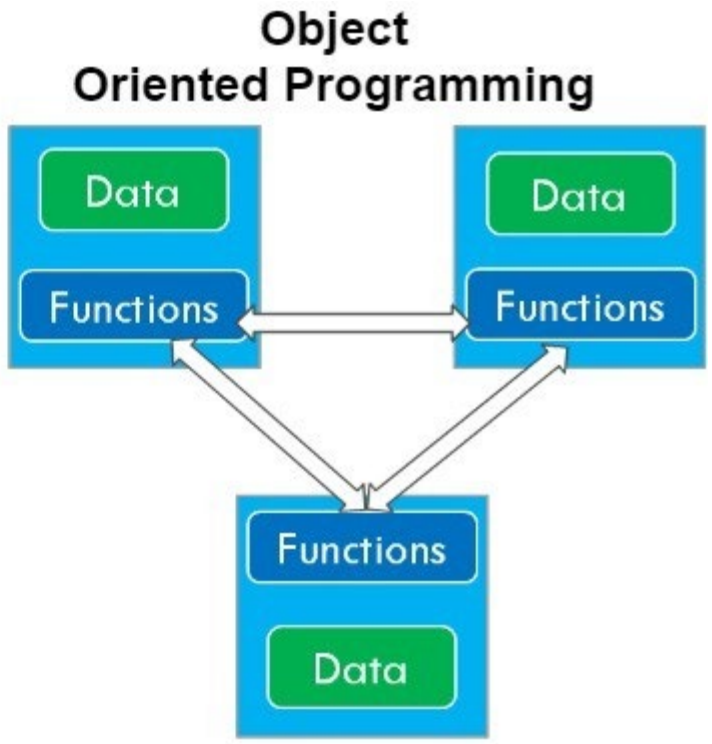
Data Encapsulation

Inheritance

Polymorphism

Message Passing





Objects Everywhere

Everything in Python is really an object.

We've seen hints of this already…

```
"hello1".upper()
list3.append("a")
dict2.keys()
```

New types of Objects can be easily defined.

In fact, programming in Python is typically done in an object-oriented fashion.

```
from bs4 import BeautifulSoup
import requests
url = input("Enter a URL: ")
r  = requests.get("http://" +url)
data = r.text
soup = BeautifulSoup(data)
for link in soup.find_all('a'):
    print(link.get('href'))
```

Can you spot all the objects?

Object-oriented Terminology

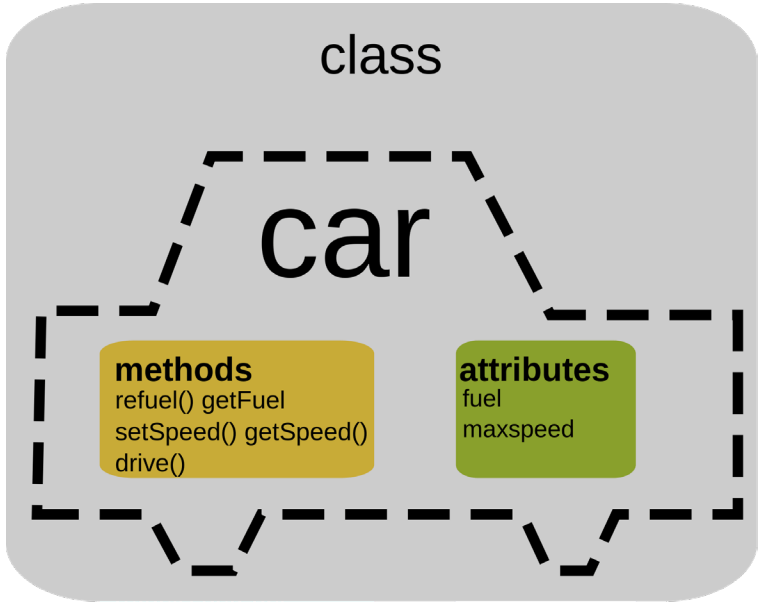
OOP allows representation of real-life objects as software objects

Object: A single software unit that combines attributes and methods

Attribute: A "characteristic" of an object; like a variable associated with a kind of object

Method: A "behavior" of an object; like a function associated with a kind of object

Class: Code that defines the attributes and methods of a kind of object



(A class is a collection of
variables and
functions working with these variables)

Example: Object-oriented Terminology

Class - a template - **Dog**

Method - A defined capability of a class - **bark**()

Attribute - A bit of data in a class – **color** of the dog

Object or **Instance** - A particular instance of a class - **Lassie**

Keywords in Python

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Define your classes

Class - a template - Dog

Method - A defined capability of a class - bark()

Attribute - A bit of data in a class – color of the dog

Object or **Instance** - A particular instance of a class - Lassie

```
class Dog:
   pass
```

Type Definition:
Code that defines the attributes and methods of a kind of object

```
lassie = Dog( )
rocky = Dog( )
```

Instantiation
To create an object. A single object is called an **Instance**

Adding to Our Dog Class

```
class Dog:
   def __init__(self, name, color):
      self.name = name
      self.color = color
            print (“a new dog is born”)
```

Constructor

```
lassie = Dog("Lassie", "white")
rocky = Dog("Rocky", "black")
```

Instantiation
(calls the constructor)

Create new object with class name followed by set of parentheses

`Dog()` creates new object of class Dog

Can assign a newly instantiated object to a variable of any name

`lassie = Dog(...)` assigns new Dog object to lassie

Avoid using variable that's same name as the class name in lowercase letters

Constructors and Methods

Constructor: A special method that is automatically invoked right after a new object is created

An **__init__** method can take any number of arguments.

Like other functions or methods, the arguments can be defined with default values, making them optional to the caller.

However, the first argument **self** in is special…

```
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def bark(self):
        print ("I am", self.color, self.name)
```

Constructor

Method

self

The first argument of every method is a reference to the current instance of the class

By convention, we name this argument **self**

In **__init__,** self refers to the object currently being created; so, in other class methods, it refers to the instance whose method was called

Although you must specify **self** explicitly when defining the method, you don't include it when calling the method.

Python passes it for you automatically

```
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def bark(self):
        print ("I am", self.color, self.name)
```

Constructor

Method

```
lassie = Dog("Lassie", "white")
lassie.bark()
```

Instantiation

Invoking/Calling a Method

Every Dog object has method **bark()**

```
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def bark(self):
        print ("I am", self.color, self.name)
```

Constructor

Method

```
lassie = Dog("Lassie", "white")
lassie.bark()
```

Instantiation

lassie.bark() invokes bark method of Dog object lassie

```
rocky = Dog("Rocky", „black")
rocky.bark()
```

Instantiation

rocky.bark() invokes bark method of Dog object rocky

Dog:Lassie

name:Lassie
color:white

Dog:Rocky

name:Rocky
color:black

Accessing attributes

Every Dog object has attributes **name**, and **color**

```
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color

    def bark(self):
        print ("I am", self.color, self.name)
```

Constructor

Method

```
lassie = Dog("Lassie", "white")
lassie.bark()
```

Instantiation

```
rocky = Dog("Rocky", „black")
rocky.bark()
```

Instantiation

```
print (lassie.name)
print (lassie.color)
```

Dog:Lassie
name:Lassie
color:white

Dog:Rocky
name:Rocky
color:black

Assessing attributes using methods: bark()

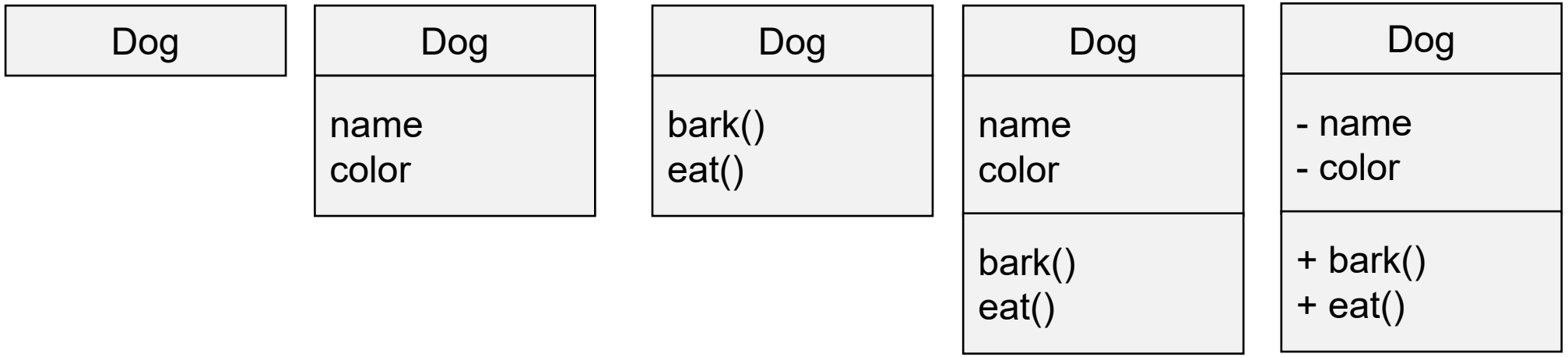
- Uses a Dog object's name attribute
- Receives reference to the object itself into self

Accessing Attributes Directly

UML Class Diagrams

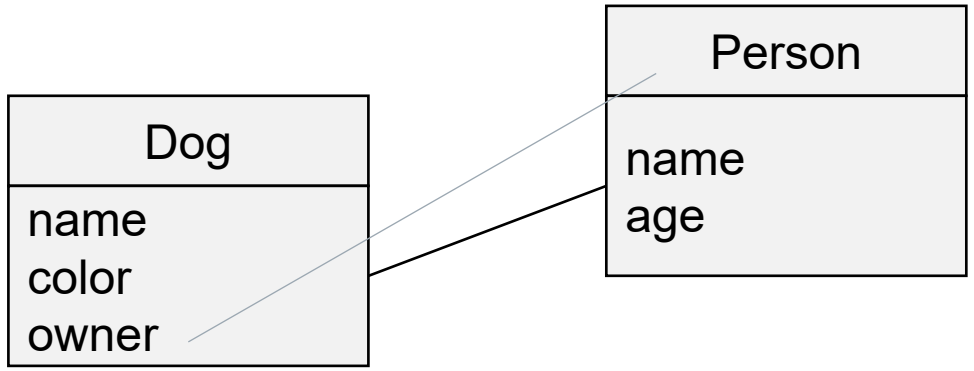
A class is simply represented as a box with the name of the class inside

The diagram may also show the attributes and operations



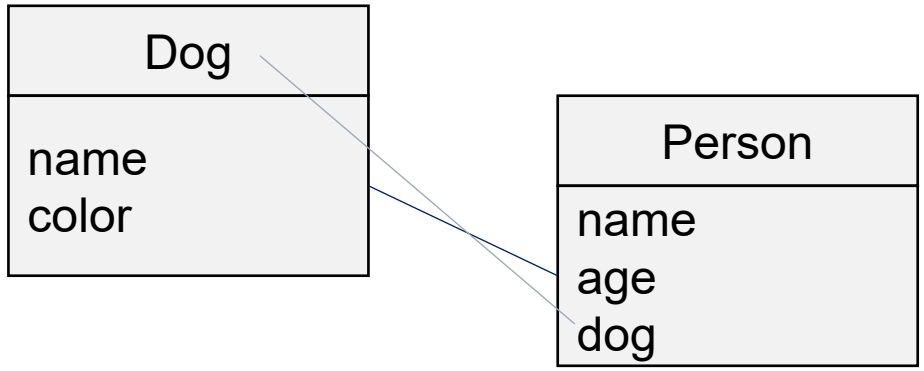
Interactions between Classes

Dog has a reference to its owner!




```
p = Person("Mr Smith")
d = Dog("Lassie")
d.owner = p
```

Owner has a reference to his/her dog!




```
p = Person("Mr Smith")
d = Dog("Lassie")
p.dog = d
```

Methods

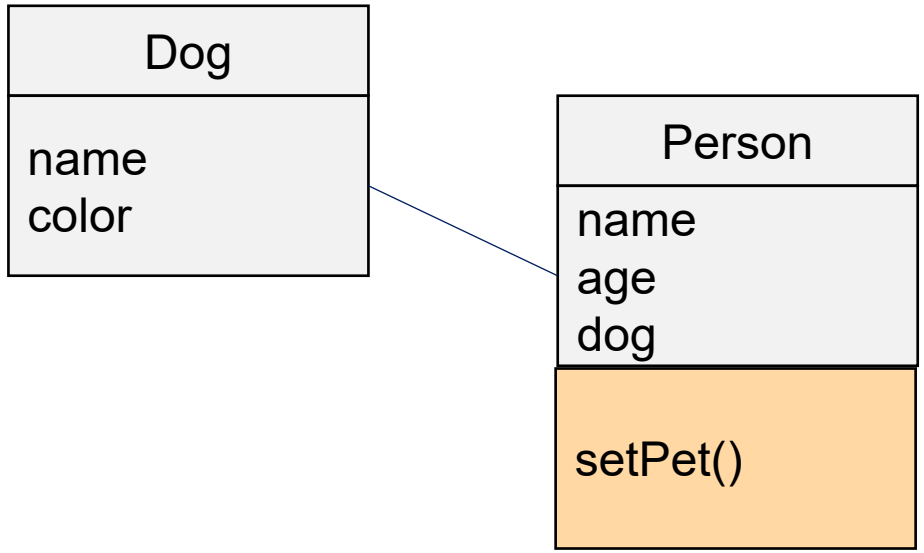
```
class Dog:
    def __init__(self, name):
        self.name = name
```

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

    def setPet(self, d):
        self.dog = d
```

```
p = Person ("Mr. Smith")
d = Dog("Lassie")
p.setPet(d)

print (d)
print(p.dog)
```




```
<__main__.Dog object at 0x000002AC4030E978>
<__main__.Dog object at 0x000002AC4030E978>
```

Try at Home

Task 1.1 Playing with objects

- Define classes required for a Zoo!
- Zoo has many animals – each animal has a name, age and weight.
- The animals can make noises and eat, and sleep.
- Zoo has animal care takers. Each caretaker has a name. Care takers feed animals.
- Create a list of 10 animals.
- Create one care taker, who feeds all the animals.
- Write code for the classes Animal and Person, so that the instantiation works as expected.

Test code

```
animals = [
          Animal("Tiger32", 12, 121),
          Animal("Lion42", 4, 131),
          Animal("Zebra 12", 12, 11),
          Animal("Bison 23", 4, 121),
          # add more
          ]

caretaker = Person ("Joe")

for animal in animals:
          caretaker.feed(animal, "Apple")
```

output

```
Joe just fed Tiger32: Apple
Joe just fed Lion42: Apple
Joe just fed Zebra 12: Apple
Joe just fed Bison 23: Apple
```

Try at Home

Task 1.2 Playing with objects

Define the classes (Student, Exam, University) so that following Excerpt of code from a Student Management System works as expected.

Test code

```
s1= Student ("Sandy", "24.01.1992") # name, dob
s2= Student ("Spili", "14.10.1993") # name, dob
s3= Student ("Waile", "04.06.1994") # name, dob

imc = University ("FH Krems")

imc.enroll(s1)
imc.enroll(s2)
imc.enroll(s3)

e1 = Exam("Programming II")
e2 = Exam("Software Eng")
e3 = Exam("Creativity")

# assign a random value as grade
s1.takeExam (e1)
s2.takeExam (e1)
s3.takeExam (e1)

s1.takeExam (e2)
s2.takeExam (e2)

s1.takeExam (e3)
s3.takeExam (e3)

# print statistics
imc.stats()
```

output

```
Sandy took 3 exams
        Got 4 in Programming II
        Got 1 in Software Eng
        Got 2 in Creativity
Spili took 2 exams
        Got 3 in Programming II
        Got 1 in Software Eng
Waile took 2 exams
        Got 3 in Programming II
        Got 2 in Creativity
```

Try at Home

Task 1.3 Playing with objects

Extend the class structure from Task 1.2, so that the new code excerpt works as expected.

Add a new method called stats() in the Exam class.

Test code

```
...
# code same as in Task 1.3

s1.takeExam (e1)
s2.takeExam (e1)
s3.takeExam (e1)

s1.takeExam (e2)
s2.takeExam (e2)

s1.takeExam (e3)
s3.takeExam (e3)

# print statistics
imc.stats()

# new code below

e1.stats()
e2.stats()
e3.stats()
```

New Output

```
Sandy took 3 exams
        Got 1 in Programming II
        Got 3 in Software Eng
        Got 1 in Creativity
Spili took 2 exams
        Got 5 in Programming II
        Got 4 in Software Eng
Waile took 2 exams
        Got 5 in Programming II
        Got 4 in Creativity
Programming II exam was taken by 3 students. Average score = 3.6666666666666665
Software Eng exam was taken by 2 students. Average score = 3.5
Creativity exam was taken by 2 students. Average score = 2.5
```