

Informatik II Woche 5



Klassen, Programmierkonzepte, Funktionelle Programmierung

Website: n.ethz.ch/~kvaratharaja/

Die Slides basieren auf den offiziellen Übungssides der Kurswebsite: <https://lec.inf.ethz.ch/mavt/informatik2/2025/>

Heute

1. Klassen

2. Programmierkonzepte

3. Funktionale Programmierung

4. Inclass-Exercise

5. Hausaufgaben

Pandas: Spalten umbenennen

Die "rename" Funktion verwenden:

```
data = data.rename(columns={  
    "noah": "Noah", "liam": "Liam", "emma": "Emma", "mia": "Mia"})
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-




	Noah	Liam	Emma	Mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

Pandas: Alle Spalten umbenennen

Column (Spalten) Namen direkt setzen

```
data.columns = ["Noah", "Liam", "Emma", "Mia"]
```

	noah	liam	emma	mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-



	Noah	Liam	Emma	Mia
Item				
Food	270	140	188	200
Insurance	72	310	88	72
Fun	410	-	60	130
Salary	-	1510	-	-
Tuition	720	820	720	720
Rent	-	430	390	-

Matplotlib: Color and Marker

```
import matplotlib.pyplot as plt
```

```
X = range(1, 9)
```

```
Y = [1, 2, 3, 1, 2, 3, 1, 2]
```

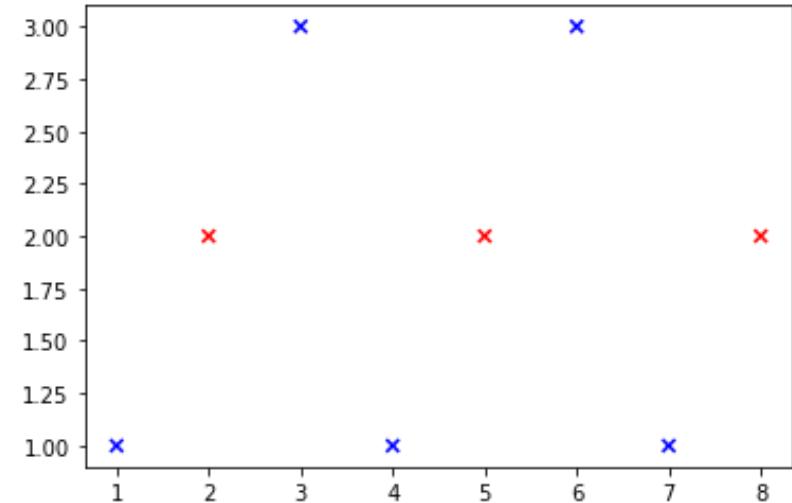
```
fig = plt.figure()
```

```
ax = fig.add_subplot()
```

```
cols = ["red" if y == 2 else "blue" for y in Y]
```

```
ax.scatter(X, Y, marker = "x", c = cols)
```

```
plt.show()
```



1. Klassen

Klassen und Objekte, Magische Methoden , Vererbung

Klassen und Objekte

- Mit steigender Komplexität ist es wichtig, Programme zu strukturieren
- Klasse: Datentypdefinition mit Attributen und Methoden.
- Objekte: Instanzen einer bestimmten Klasse.

Student

- **name**
- **grade**

- **sleep**
- **eat**
- **study**

Klasse implementieren in Python

```
class Student:
    def __init__(self):
        self.name = ''
    def sleep(self, hours):
        print(self.name, 'slept for', hours, 'hours')
    def eat(self, food):
        print(self.name, 'ate', food)
    def study(self, hours, subject):
        print(self.name, 'studied', subject, 'for', hours, 'hours')
```

Konstruktor — [def __init__(self):
self.name = ''] — public attributes/
member variables

Methoden — [def sleep(self, hours):
print(self.name, 'slept for', hours, 'hours')
def eat(self, food):
print(self.name, 'ate', food)
def study(self, hours, subject):
print(self.name, 'studied', subject, 'for', hours, 'hours')]

Klasse implementieren in Python

```
class Student:
    def __init__(self):
        self.name = ''

    def sleep(self, hours):
        print(self.name, 'slept for', hours, 'hours')

    def eat(self, food):
        print(self.name, 'ate', food)

    def study(self, hours, subject):
        print(self.name, 'studied', subject, 'for',
hours, 'hours')
```

```
s = Student()
s.name = 'John'
s.sleep(7)
```

Output:

John slept for 7 hours

Übung 1.1: Liste von Objekten

```
names = ['Anne', 'Ben', 'Charles', 'David', 'Elena', 'Fiona']
```

Erstelle eine Liste **students** von **Student**-Objekten mit den entsprechenden Namen von **names**.

Lade dafür das Jupyter Notebook in meiner Polybox runter.

Übung 1.2: Liste von Objekten

```
names = ['Anne', 'Ben', 'Charles', 'David', 'Elena', 'Fiona']
```

Lasse alle Studierenden in **students** 9 Stunden schlafen, 6 Stunden Informatik lernen und dann eine Pizza essen.

Quiz 1

Was ist die Ausgabe?

```
class Dummy:
    def __init__(self, n):
        self.n = n

    def update(self, m):
        self.n += m
```

```
d = Dummy(2)
d.update(3)
print(d.n)
```

Quiz 2

Was ist die Ausgabe?

```
class Dummy:  
    def __init__(self, n):  
        self.n = n  
  
    def method1(self, n):  
        return self.n * 2
```

```
d = Dummy(4)  
res = d.method1(3)  
print(res)
```

Übung 2: Definition einer Klasse

- Erstellen Sie eine Klasse **Rectangle** mit den folgenden Attributen und Methoden.
- Erstellen Sie ein Objekt mit Breite 4 und Höhe 5 und verwenden Sie die Methode **area**, um seine Fläche zu berechnen.

Rectangle

- **width**
- **height**

- **area**

Magische Methoden

- Die sogenannten magischen Methoden definieren Standardoperatoren, die eine Klasse definieren kann
- Dafür muss lediglich die entsprechende Methode implementiert werden

Magische Methoden

Operation	Meaning	Magical Method
<	less than	__lt__
<=	less than or equal	__le__
>	greater than	__gt__
>=	greater than or equal	__ge__
==	equal to	__eq__
!=	not equal to	__ne__

Magische Methoden

Operation	Meaning	Magical Method
+, +=	Addition	<code>__add__</code> , <code>__iadd__</code>
-	Subtraction	<code>__sub__</code>
*	Multiplication	<code>__mul__</code>
/	Division	<code>__truediv__</code>
//	Integer division	<code>__floordiv__</code>
%	Modulo	<code>__modulo__</code>

Beispiel: Output

- Die Standardausgabe für eigene Klassen ist nicht sehr hilfreich.

```
class Rectangle:  
    ...  
  
rect = Rectangle(m,n)  
print(rect)
```

<__main__.Rectangle object at 0x0000020B02DF17C0>

Beispiel: Output

- Um dies zu verändern, können wir die magische Methode `__str__` implementieren.

```
class Rectangle:
    def __str__(self):
        return f'Rectangle: {self.width} x {self.height}'

rect = Rectangle(m,n)
print(rect)
```

Rectangle: 4 x 5

Quiz 3

- Was ist die Ausgabe?

```
class Rectangle:  
    ...  
    def __add__(self, other):  
        width = other.width + self.width  
        height = other.height + self.height  
        return Rectangle(width, height)
```

```
a = Rectangle(4,5)  
b = Rectangle(2,3)  
c = a + b  
print(c)
```

Quiz 4

- Was ist die Ausgabe?

```
class Rectangle:
    def __add__(self, other):
        width = other.width + self.width
        height = other.height + self.height
        return Rectangle(width, height)
```

```
a = Rectangle(4,5)
b = Rectangle(2,3)
c = a + b
print(c)
```

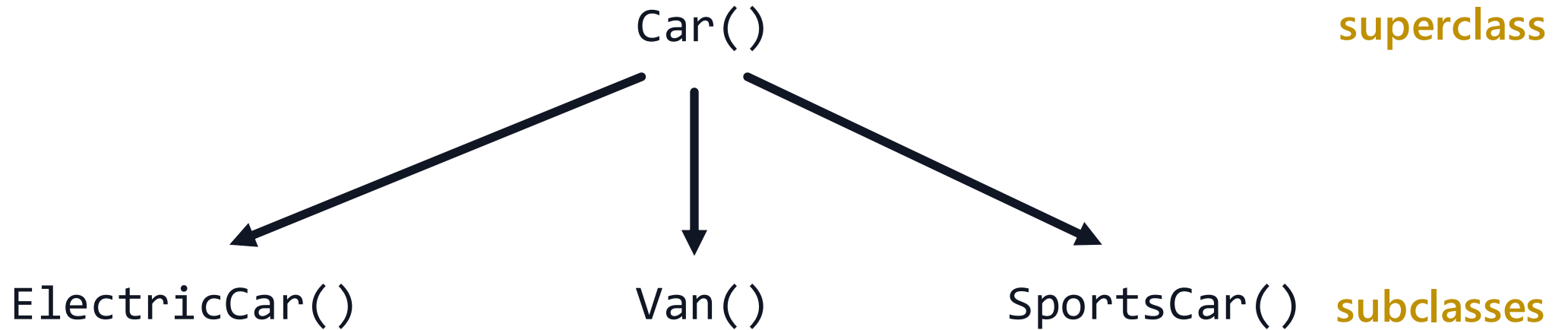
Vererbung: Motivation

- Vererbung drückt eine Beziehung von Klassen aus.
- Eine Subklasse (erbende Klasse) beinhaltet dabei die Attribute und Methoden der Elternklasse (Basisklasse)

Beispiel

- Basisklasse: **Precipitation**
- Attribute: **amount**
- Methoden: **__str__**, **alarm**
- Subklassen: **Rain**, **Snow**

Vererbung



- Erstellt viele verschiedene Klassen desselben „Typs“ durch Vererbung
- Unterklassen erhalten alle Attribute und Methoden der Oberklasse

Beispiel

- Beziehung: Ein Quadrat **ist** ein Rechteck

```
class Sqaure(Rectangle):  
    def __init__(self, n):  
        Rectangle.__init__(self, width = n, height = n)  
  
s = Square(2)  
print(s)
```

Rectangle: 2 x 2

Quiz 5: Vererbung

Was ist der Output von folgendem Code

```
a = Square(n = 4)
b = Rectangle(4,2)
c = a + b
print(c, type(c))
```

Quiz 6: Vererbung

Was ist der Output von folgendem Code?

```
a = Square(4)
b = Square(2)
c = a + b
print(c, type(c))
```

Quiz 7: Vererbung

Was ist der Output von folgendem Code?

```
class Sqaure(Rectangle):  
    ...  
    def __add__(self, other):  
        if isinstance(other, Square):  
            return Square(self.width + other.width)  
        else:  
            return other + self  
  
a = Square(4)  
b = Rectangle(3,2)  
c = a + b  
print(c, type(c))
```

Quiz 8: Vererbung

Was ist der Output von folgendem Code?

```
class Sqaure(Rectangle):  
    ...  
    def __add__(self, other):  
        if isinstance(other, Square):  
            return Square(self.width + other.width)  
        else:  
            return other + self  
  
a = Square(n = 4)  
b = Square(n = 2)  
c = a + b  
print(c, type(c))
```

Rückblick Ex.2: Behandlung von Exceptions

- Wir können **Try**- und **Except**-Klauseln verwenden, um Exceptions zu behandeln.
- Eine try-Anweisung kann mehr als eine Except-Klausel haben, um Handler für verschiedene Exceptions anzugeben.
- Es wird jedoch höchstens ein Handler ausgeführt.

```
a = [1, 2, 3]
try:
    print("Second element = %d" %(a[1]))
    # Throws error since there are only 3 elements in array
    print("Fourth element = %d" %(a[3]))
except IndexError:
    print("An error occurred")
```

Output:

Second element = 2
An error occurred

Vererbung: Custom Exceptions

- Man kann seine eigenen Exceptions erstellen, indem man eine neue Klasse erstellt.
- Neue Exceptions müssen von der Klasse Exception abgeleitet werden.

```
class MyError(Exception):  
    # Constructor or Initializer  
    def __init__(self, value):  
        self.value = value  
    # __str__ is to print() the value  
    def __str__(self):  
        return(repr(self.value))  
  
try:  
    raise(MyError(3*2))  
# Value of Exception is stored in error  
except MyError as error:  
    print("A New Exception occurred: ", error.value)
```

Output:

A New Exception occurred: 6

2. Programmierkonzepte

Key Differences Python/ C++

Python	C++
Interpreted	Compiled
Dynamically typed	Statically typed
Object-oriented, functional	Object-oriented, functional

Kompilieren vs Interpretieren

Kompilierte Programmiersprachen (z.B C / C++) ← **Schneller**

- Programmcode wird von einem Compiler (z.B. GCC) in Assemblercode kompiliert;
- Assemblercode wird ausgeführt;
- Der Programmcode sollte rekompiliert werden sofern Änderungen vorgenommen wurden.

Interpretierte Programmiersprachen (z.B. Python, JavaScript)

- Der Interpreter liest den Programmcode und führt ihn direkt aus;
- Die Interpretation wird bei jeder Ausführung des Programmcodes wiederholt;
- Kleinere Änderungen können schnell und einfach vorgenommen werden.

Kompiliert/Interpretiert

Kompiliert: Der gesamte Code wird vom Compiler in Maschinencode übersetzt.



Interpretiert: der Code wird direkt (Zeile für Zeile) vom Interpreter ausgeführt



Kompilieren vs Interpretieren: Ein Beispiel

Führe ein C++-Programm aus, nachdem es modifiziert wurde.

```
g++ program.cc -o program #compile the program
./program #execute the program
... #modify program.cc
g++ program.cc -o program # compile the program again
./program # execute the program
```

Führe ein Python-Programm aus, nachdem es modifiziert wurde.

```
python program.py #interpret and execute the program
... #modify program.py
python program.py #interpret and execute the program
```

Statically/Dynamically typed

Statically Typed: type checking takes place at compile time.

- Types of variables can't be changed

```
int n = 5; //n stays int forever  
std::string s = "hello"; //s stays string forever
```

Dynamically Typed: type checking takes place at runtime

- Types of variables can be changed throughout the code

```
n = 5 #n is an integer here  
n = "hello" #n is a string now
```

Statische vs Dynamische Typisierung

Statische Typisierung (C / C++)

- Jedes Element hat einen vom Programmierer definierten Typ;
- Ob die gewählten Typen korrekt zusammenpassen, wird bei der Kompilation überprüft, sodass Kompilierungsfehler erzeugt werden, wenn dies nicht der Fall ist.

Dynamische Typisierung (Python)

- Elemente haben im Vorhinein keinen festgelegten Typ;
- Der Typ wird erst zur Laufzeit gewählt;
- Der Typ ist zur Laufzeit veränderlich;
- Abhängig vom Typ während der Ausführung kann es zu Laufzeit-Fehlern kommen;
- Fehler sind schwieriger zu beheben, da sie nicht immer auftreten.

Dynamische Typisierung: Ein Beispiel

Für Python nicht zwingend erforderlich

Welche Outputs werden generiert?

```
def add_integers(l: list[int]) -> int:  
    return sum(l[::2])
```

```
print(add_integers([1, 2, 3, 4, 5]))  
print(add_integers([1, 2, 'a', 4, 5]))
```

Generische Programmierung

Schreibe Code (Funktionen, Klassen), der so allgemein wie nur möglich einsetzbar ist.

Python unterstützt die generische Programmierung.

```
def add(x,y):  
    return x + y
```

Die Funktion **add** ist anwendbar für alle Typen, für die die "+"-Operation definiert ist.

3. Funktionale Programmierung

Funktionale Programmierung

Funktionen können als Argumente an andere Funktionen weitergegeben werden.

```
def increase_one(n):  
    return n + 1  
def applyToAll(function, container):  
    return [function(elem) for elem in container]  
  
number = [1, 2, 3, 4, 5]  
# increase_one is passed as an argument  
print(applyToAll(increase_one, number))
```

Output: ?

Funktionale Programmierung: map

map: Eine eingebaute Python-Funktion, die eine Funktion auf jedes Element eines Iterable (z.B. list oder set) anwendet und ein neues Iterable mit den Resultaten zurückgibt.

```
def increase_one(n):  
    return n + 1  
  
new_number = map(increase_one, [1,2,3,4,5])  
print(new_number)  
print(list(new_number))
```

Output: ?

Funktionale Programmierung: map

Die als Input übergebene Funktion kann mehrere Argumente besitzen. In diesem Fall muss **map** die selbe Anzahl an Iterables als Argumente übergeben werden. Die Länge des zurückgegebenen Iterables ist gleich der Länge des **KÜRZESTEN** Input-Iterables.

```
def add(x, y):  
    return x + y  
  
number1 = [1, 2, 3, 4, 5, 6] # length: 6  
number2 = [7, 8, 9, 10, 11] # length: 5  
new_number = map(add, number1, number2)  
print(list(new_number))
```

Output: ?

Funktionelle Programmierung: filter

filter: Eine eingebaute Python-Funktion, die aufgrund einer gegebenen Funktion Elemente aus einem Iterable (z.B. list oder set) filtert und diese Elemente in einem neuen Iterable zurückgibt.

```
def is_even(n):  
    return n % 2 == 0  
even_number = filter(is_even, [1, 2, 3, 4, 5])  
print(even_number)  
print(list(even_number))
```

Output: ?

Funktionelle Programmierung: reduce

reduce: Eine eingebaute Python-Funktion im **functools**-Modul, das eine gegebene Funktion in kumulativer Weise auf die Elemente eines gegebenen Iterables anwendet und einen einzelnen Wert zurückgibt.

```
from functools import reduce
def add (x, y):
    return x + y

Total = reduce(add, [1,2,3,4,5])
print(Total)
```

Output: ?

Lambda Expressions

- Funktion ohne expliziten Namen.
- Meistens simpel und kurz. Notation:

```
lambda <arguments> : <expression>
```

- Example:

```
lambda : print('Hello World') # no argument  
lambda x,y : x + y # two arguments x and y, return x + y
```

Lambda Expression

```
from functools import reduce  
n = [1,2,3,4,5]  
  
total = reduce(lambda x, y : x + y, [1, 2, 3, 4, 5])  
print(total)
```

Output: ?

lambda <arguments> : <expression>

Quiz 1

What is the output of the following code snippet?

```
add = lambda x, y : x + y
multiply = lambda a, b : a * b
mean_3 = lambda f, g, h : (f + g + h)/3

add(mean_3(1,2,3), multiply(2,5))
```

A. 13.0

B. error

C. 12.0

D. 15.0

`lambda <arguments> : <expression>`

Quiz 2

- What is the output of the following code snippet?

```
apply = lambda x, y : x(y)  
  
print(apply(lambda a : a + 2, 5))
```

A. a

B. error

C. 2

D. 7

4. In-Class Exercise

In-class Exercise 5: Departemente

CodeExpert in-class task:

<https://expert.ethz.ch/enrolled/SS25/mavt2/codeExamples>

Define classes, create objects, and use magical methods.

- Define classes (task 1 & 2);
- Define magical methods (task 3 & 5);
- Create objects (task 4);
- Use magical methods (task 6).

5. Hausaufgaben

Exercise 4: Classes and Programming Concepts

Auf <https://expert.ethz.ch/mycourses/SS25/mavt2/exercises>

- Immobilien
- Erdbeben-Datenbank
- Student and Faculty (inheritance)

Fällig bis Montag 24.03.2025, 20:00 CET

NO HARDCODING

Feedback?

Zu schnell? Zu langsam? Weniger Theorie, mehr Aufgaben?
Dankbar für Feedback am besten mir direkt sagen oder Mail
schreiben

Credits

Die Slide(-templates) stammen ursprünglich von Julian Lotzer und Daniel Steinhauser, vielen Dank!

→ Checkt ihre Websites ab für zusätzliches Material in Informatik I, Informatik II und Stochastik & Machine Learning.

- <https://n.ethz.ch/~jlotzer/>
- <https://n.ethz.ch/~dsteinhauser/>