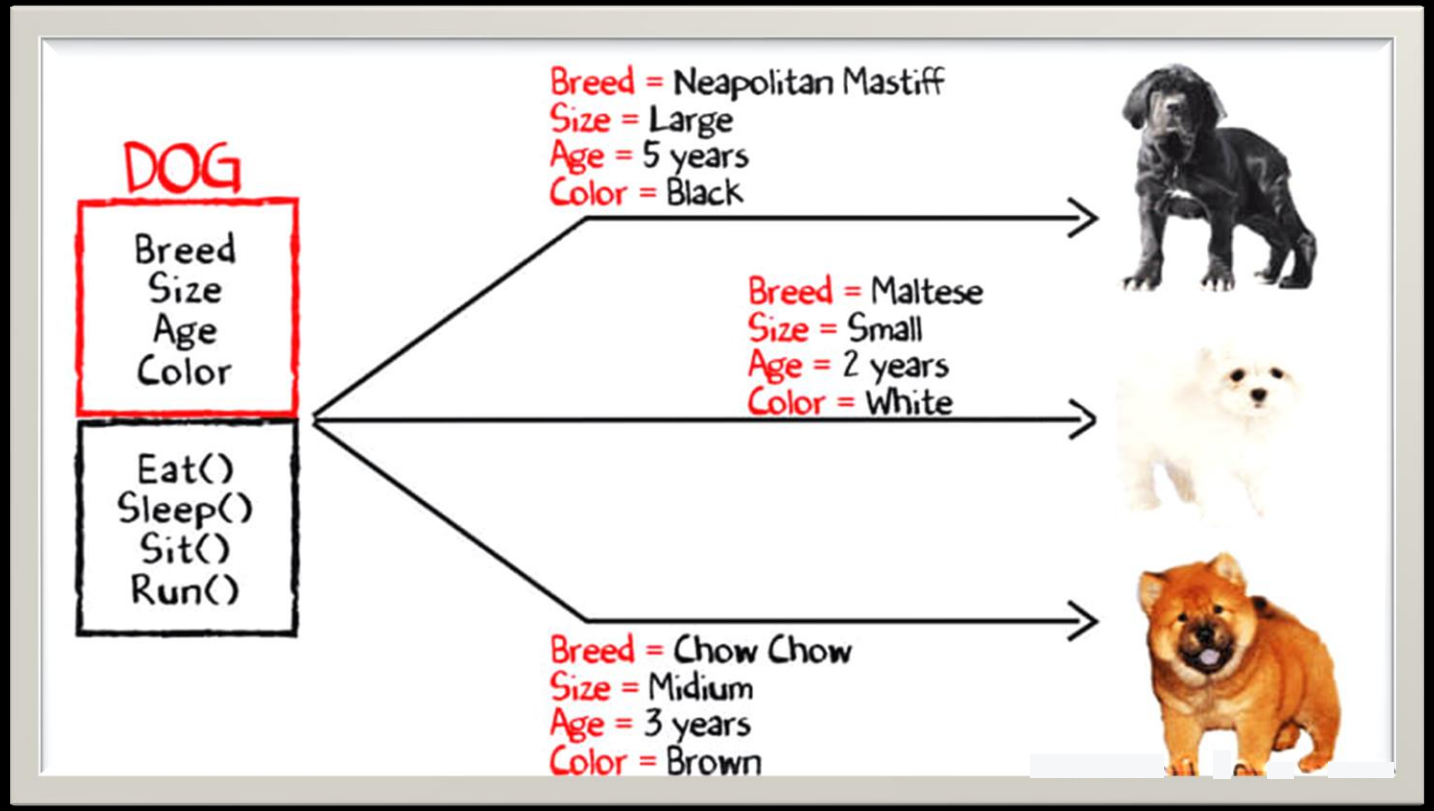
# Informatik Q1 Abels

# Klasse vs. Objekt

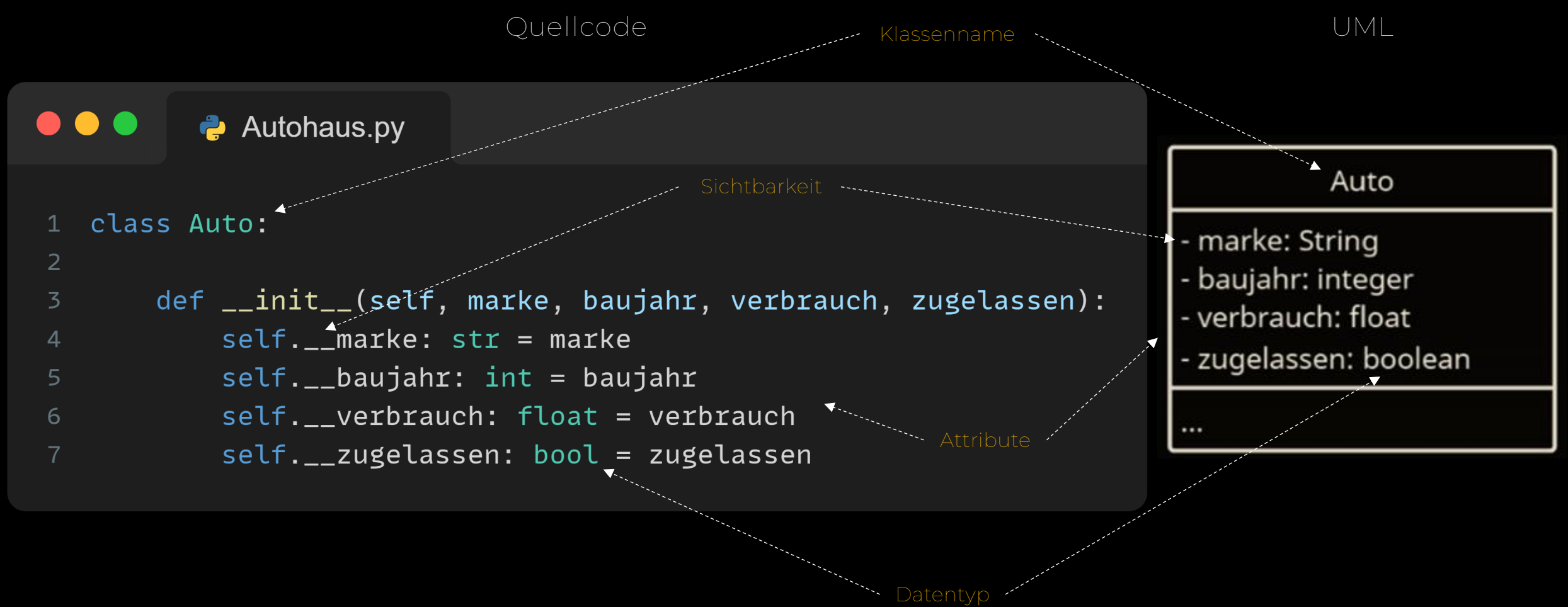
# Klasse vs. Objekt




Klassen sind Vorlagen, aus denen zur Laufzeit Instanzen erzeugt werden.

Diese Intanzen heißen Objekte.

# Klasse


Quellcode
Klassenname
UML
Autohaus.py
Sichtbarkeit
1 class Auto:
2
3 def __init__(self, marke, baujahr, verbrauch, zugelassen):
4 self.__marke: str = marke
5 self.__baujahr: int = baujahr
6 self.__verbrauch: float = verbrauch
7 self.__zugelassen: bool = zugelassen
Attribute
Datentyp
Auto
- marke: String
- baujahr: integer
- verbrauch: float
- zugelassen: boolean
...

# Sichtbarkeit

bzw. Datenkapselung

| Typ | Zugriff | Quellcode | UML |
|---|---|---|---|
| public | Überall | name | + |
| protected | In der Klasse und den Subklassen | _name | # |
| private | In der Klasse | __name | - |

# Objekt

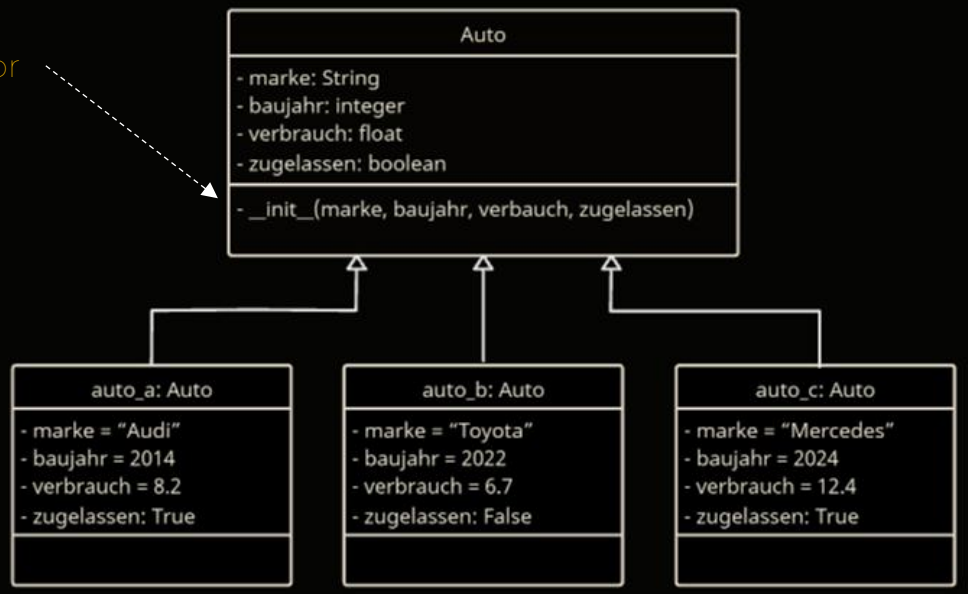
Autohaus.py

```python
class Auto:

    def __init__(self, marke, baujahr, verbrauch, zugelassen):
        self.__marke: str = marke
        self.__baujahr: int = baujahr
        self.__verbrauch: float = verbrauch
        self.__zugelassen: bool = zugelassen

auto_a = Auto("Audi", 2014, 8.2, True)
auto_b = Auto("Toyota", 2022, 6.7, False)
auto_c = Auto("Mercedes", 2024, 12.4, True)
```

Konstruktor

Variable vom Typ "Auto" = Objekt der Klasse „Auto“

Aufruf des Konstruktors



# Getter

Autohaus.py

```python
class Auto:

    def __init__(self, marke, baujahr, verbrauch, zugelassen):
        self.__marke: str = marke
        self.__baujahr: int = baujahr
        self.__verbrauch: float = verbrauch
        self.__zugelassen: bool = zugelassen

    def get_marke(self) -> str:
        return self.__marke

    def get_baujahr(self) -> int:
        return self.__baujahr

    def get_verbrauch(self) -> float:
        return self.__verbrauch

    def get_zugelassen(self) -> bool:
        return self.__zugelassen

auto_a = Auto("Audi", 2014, 8.2, True)
auto_b = Auto("Toyota", 2022, 6.7, False)
auto_c = Auto("Mercedes", 2024, 12.4, True)

print(auto_a.get_marke())
print(auto_b.get_baujahr())

if auto_c.get_zugelassen():
    print(f"Der {auto_c.get_marke()} ist zugelassen.")
else:
    print(f"Der {auto_c.get_marke()} ist noch nicht zugelassen.")
```

get-Methoden

Datentyp der Rückgabe der Methode

Ausgabe der Marke von auto_a

Einbettung der Attribute in Algorithmen

# Setter

Autohaus.py

```python
class Auto:

    def __init__(self, marke, baujahr, verbrauch, zugelassen):
        self.__marke: str = marke
        self.__baujahr: int = baujahr
        self.__verbrauch: float = verbrauch
        self.__zugelassen: bool = zugelassen

    def get_marke(self) -> str:
        return self.__marke

    def get_baujahr(self) -> int:
        return self.__baujahr

    def get_verbrauch(self) -> float:
        return self.__verbrauch

    def get_zugelassen(self) -> bool:
        return self.__zugelassen

    def set_marke(self, marke: str):
        self.__marke = marke

    def set_baujahr(self, baujahr: int):
        self.__baujahr = baujahr

    def set_verbrauch(self, verbrauch: float):
        self.__verbrauch = verbrauch

    def set_zugelassen(self, zugelassen: bool):
        self.__zugelassen = zugelassen

auto_a = Auto("Audi", 2014, 8.2, True)
auto_b = Auto("Toyota", 2022, 6.7, False)
auto_c = Auto("Mercedes", 2024, 12.4, True)

auto_a.set_marke("Volvo")
auto_b.set_baujahr(2023)

if auto_c.get_zugelassen():
    print(f"Der {auto_c.get_marke()} ist bereits zugelassen.")
else:
    auto_c.set_zugelassen(True)
```

set-Methoden

Änderung der Marke von auto_a

Einbettung in Algorithmen

| Auto |
| --- |
| - marke: String<br>- baujahr: integer<br>- verbrauch: float<br>- zugelassen: boolean |
| - __init__(marke, baujahr, verbauch, zugelassen)<br>+ get_marke(): String<br>+ get_baujahr(): integer<br>+ get_verbrauch(): float<br>+ get_zugelassen(): boolean<br>+ set_marke(marke: String)<br>+ set_baujahr(baujahr: integer)<br>+ set_verbrauch(verbrauch: float)<br>+ set_zugelassen(zugelassen: boolean) |

# Methoden

Autohaus.py

```python
class Auto:

    def __init__(self, marke, baujahr, verbrauch, zugelassen):
        self.__marke: str = marke
        self.__baujahr: int = baujahr
        self.__verbrauch: float = verbrauch
        self.__zugelassen: bool = zugelassen

    def get_marke(self) -> str:
        return self.__marke

    def get_baujahr(self) -> int:
        return self.__baujahr

    def get_verbrauch(self) -> float:
        return self.__verbrauch

    def get_zugelassen(self) -> bool:
        return self.__zugelassen

    def set_marke(self, marke: str):
        self.__marke = marke

    def set_baujahr(self, baujahr: int):
        self.__baujahr = baujahr

    def set_verbrauch(self, verbrauch: float):
        self.__verbrauch = verbrauch

    def set_zugelassen(self, zugelassen: bool):
        self.__zugelassen = zugelassen

    def bewerten(self) -> str:
        if self.__baujahr < 2015:
            return "Das Auto ist alt."
        else:
            return "Das Auto ist neu."

auto_a = Auto("Audi", 2014, 8.2, True)

print(auto_a.bewerten())
```

eigene Methoden

Aufruf eigener Methoden

| Auto |
| --- |
| - marke: String<br>- baujahr: integer<br>- verbrauch: float<br>- zugelassen: boolean |
| - __init__(marke, baujahr, verbauch, zugelassen)<br>+ get_marke(): String<br>+ get_baujahr(): integer<br>+ get_verbrauch(): float<br>+ get_zugelassen(): boolean<br>+ set_marke(marke: String)<br>+ set_baujahr(baujahr: integer)<br>+ set_verbrauch(verbrauch: float)<br>+ set_zugelassen(zugelassen: boolean)<br>+ bewerten(): String |

# Übung 1

Schreibe ein Programm **Verkehr**. In dem Programm ist eine Klasse **Bus** defeiniert. Ein Bus hat einen Fahrer und eine Anzahl an Gästen. Beide Attribute sollen sowohl gelesen (Getter) als auch geändert (Setter) werden können. Außerdem können in einen Bus n Personen zusteigen (Methode) bzw. aussteigen (Methode).

a) Entwirf ein UML-Diagramm für die Klasse **Bus**.

b) Erweitere das UML-Diagramm um zwei Objekte `bus_1` und `bus_2`.

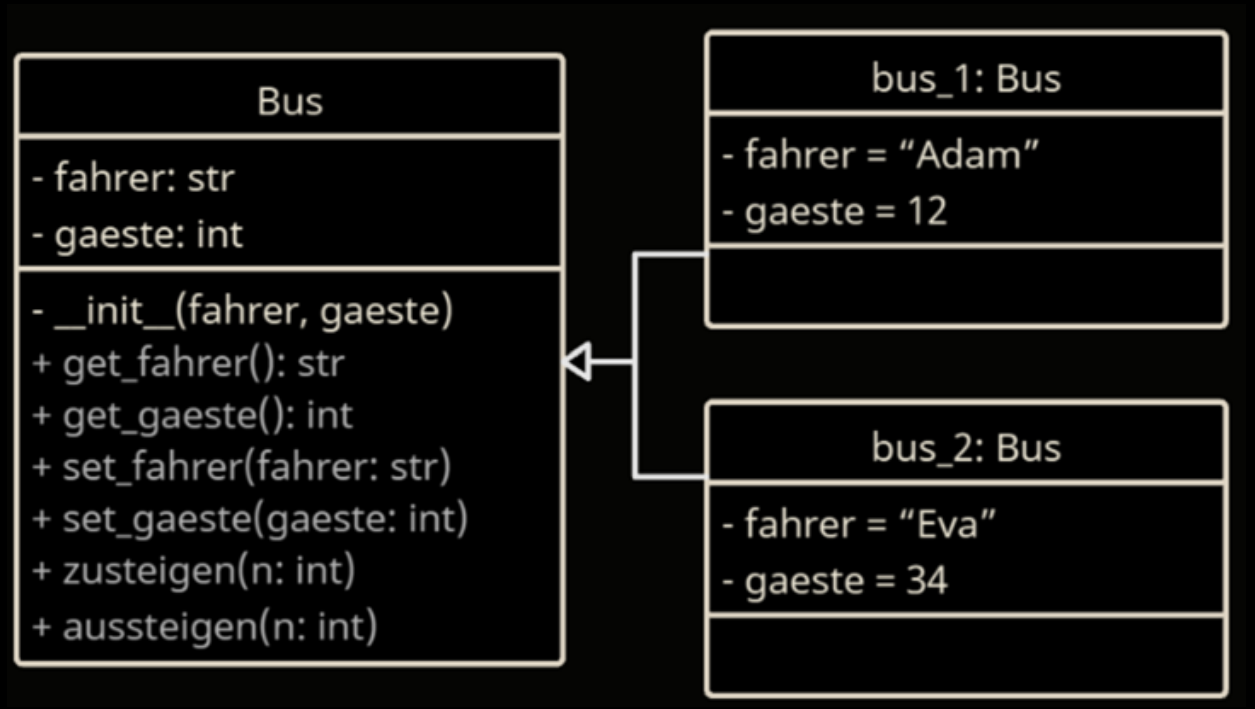
c) Implementiere die Klasse **Bus**.

d) Erzeuge im Verkehr die beiden Objekte `bus_1` und `bus_2` mit beliebigen Daten.

e) Gib den Fahrer des Busses mit den meisten Gästen aus.

f) Befördere 5 Gäste vom Bus mit mehr Gästen zum Bus mit weniger Gästen.

# Übung 1




Verkehr.py

```python
# c)

class Bus:

    def __init__(self, fahrer, gaeste):
        self.__fahrer: str = fahrer
        self.__gaeste: int = gaeste

    def get_fahrer(self) -> str:
        return self.__fahrer

    def get_gaeste(self) -> int:
        return self.__gaeste

    def set_fahrer(self, fahrer: str):
        self.__fahrer = fahrer

    def set_gaeste(self, gaeste: int):
        self.__gaeste = gaeste

    def zusteigen(self, n: int):
        self.__gaeste += n

    def aussteigen(self, n: int):
        self.__gaeste -= n

# d)

bus_1 = Bus("Adam", 12)
bus_2 = Bus("Eva", 34)

# e)

if bus_1.get_gaeste() > bus_2.get_gaeste():
    print(bus_1.get_fahrer())
else:
    print(bus_2.get_fahrer())

# f)

if bus_1.get_gaeste() > bus_2.get_gaeste():
    bus_1.aussteigen(5)
    bus_2.zusteigen(5)
else:
    bus_2.aussteigen(5)
    bus_1.zusteigen(5)
```

# Tagebucheintrag

## Klasse vs. Objekt

# Wochenübung

| Dreieck |
| --- |
| - a: float<br>- b: float<br>- c: float |
| - __init__(a, b, c)<br>+ get_a(): float<br>+ get_b(): float<br>+ get_c(): float<br>+ set_a(a: float)<br>+ set_b(b: float)<br>+ set_c(c: float)<br>+ area(): float |

Schreibe ein Programm **`Geometrie`**.

a) Implementiere eine Klasse **`Dreieck`** nach folgendem UML:

b) Implementiere die Funktion **`area()`**. (Formel recherchieren!)

c) Teste die Funktion anhand eines Dreiecks `d_1` mit beliebigen Werten.

d) Implementiere eine Klasse **`Punkt`** mit den Attributen `x: float` und `y: float` sowie der Methode `distance(Punkt p) -> float`.

e) Teste die Funktion anhand zweier Punkte `p_1` und `p_2` mit beliebigen Werten.

f) Berechne die Fläche des Dreiecks ABC mit A(1|2), B(2|6) und C(7|0.5).