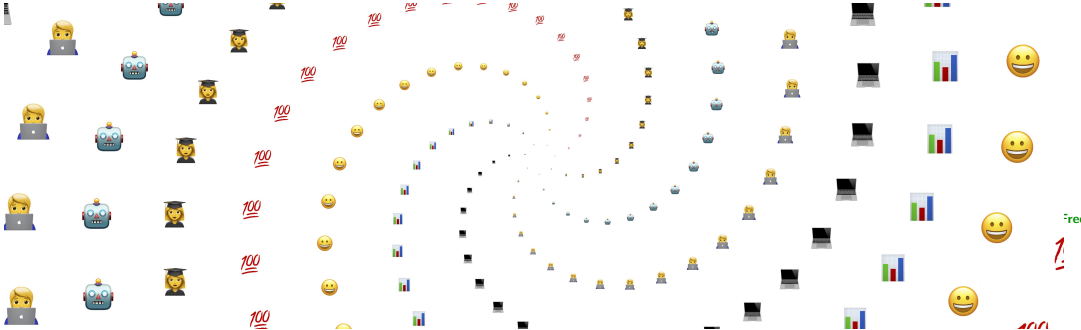
Programmieren: Klassen

Klassen & Objektorientierte Programmierung

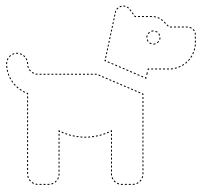
Cyril Blum

Fachschaft Informatik
Kantonsschule im Lee



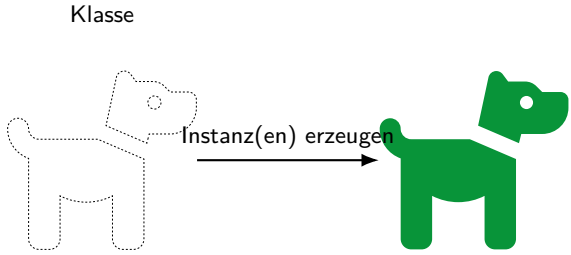
Klassen in Python

Klasse



```
class Hund:
```

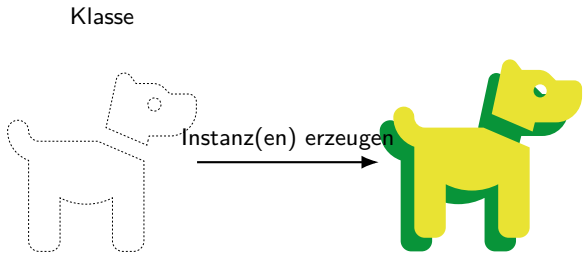
Klassen in Python




```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```



Klassen in Python

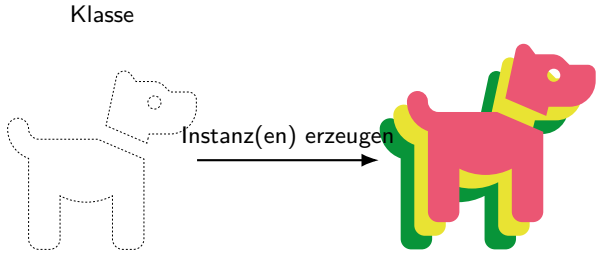



```
class Hund:
```

```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```



Klassen in Python



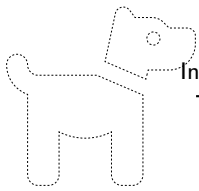

```
class Hund:
```

```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```



Klassen in Python

Klasse



Instanz(en) erzeugen

Objekte / Instanzen



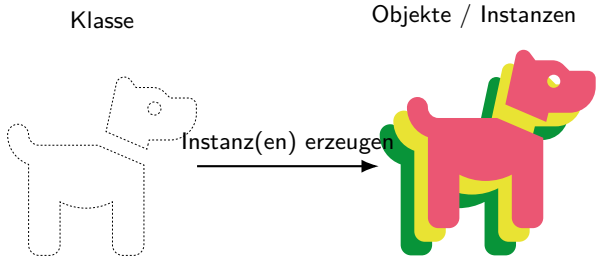
```
class Hund:
```

```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```



FreeFlower

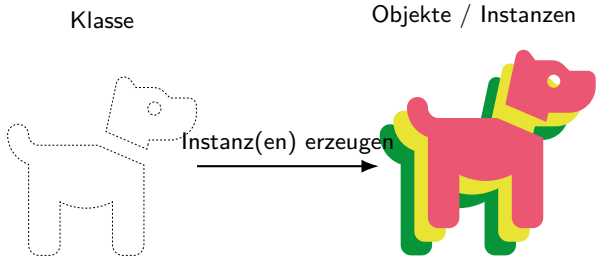
Klassen in Python




```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```

- Eigenschaften / Attribute: `name`, `groesse`, `farbe`, etc.

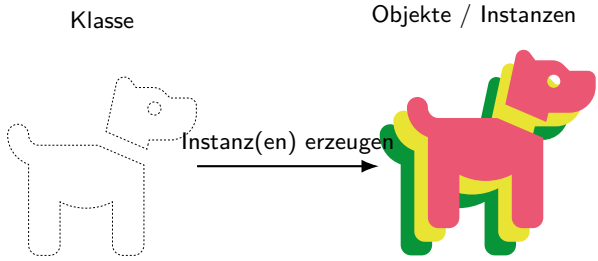
Klassen in Python




```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```

- Eigenschaften / Attribute: `name`, `groesse`, `farbe`, etc.
- Methoden: `.bellen()`, `.sitzen()`, etc.

Klassen in Python




```
hund1 = Hund("Thommy", 510, "Grün")
hund2 = Hund("Bello", 380, "Gelb")
hund3 = Hund("Rocky", 315, "Rot")
# ...
```

- Eigenschaften / Attribute: `name`, `groesse`, `farbe`, etc.
- Methoden: `.bellen()`, `.sitzen()`, etc.
- **Wird in Games ständig verwendet, um Objekte zu „spawnen“!**

Klassen in Python

- Eine Klasse ist ein Bauplan für Objekte.



Klassen in Python

- Eine Klasse ist ein Bauplan für Objekte.
- Sie definiert Eigenschaften (Attribute) und Verhalten (Methoden) von Objekten.



Klassen in Python

- Eine Klasse ist ein Bauplan für Objekte.
- Sie definiert Eigenschaften (Attribute) und Verhalten (Methoden) von Objekten.
- Klassen ermöglichen die Erstellung von benutzerdefinierten Datentypen.



Klassen in Python

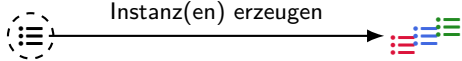
- Eine Klasse ist ein Bauplan für Objekte.
- Sie definiert Eigenschaften (Attribute) und Verhalten (Methoden) von Objekten.
- Klassen ermöglichen die Erstellung von benutzerdefinierten Datentypen.
- Wir kennen Klassen bereits von anderen Typen: **Listen, Zahlen, Zeichenketten!**



Listen als Klassen

Klasse

Objekte / Instanzen




```
class list:
```

Attribute
Elemente
Länge
...

Methoden
append()
remove()
sort()
pop()
count()
...

```
liste1 = [1,2,3]
liste2 = ["a","b"]
liste3 = [True, False]
# ...
```

Attribute
Elemente : [1,2,3]
Länge : 3
...

Methoden
append()
remove()
sort()
pop()
count()
...

FreeFlower

Beispiel: Personen

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

    def vorstellen(self):
        print(f"Hallo, ich bin {self.name} und {self.alter} Jahre alt.")

    def geburtstag_feiern(self):
        self.alter += 1
        print(f"{self.name} ist jetzt {self.alter} Jahre alt.")



# Objekte der Klasse Person erstellen
anna = Person("Anna", 16)
jan = Person("Jan", 17)

# Methoden aufrufen
anna.vorstellen()  # Ausgabe: Hallo, ich bin Anna und 16 Jahre alt.
jan.vorstellen()   # Ausgabe: Hallo, ich bin Jan und 17 Jahre alt.

# Geburtstag feiern
anna.geburtstag_feiern()  # Ausgabe: Anna ist jetzt 17 Jahre alt.
```

Beispiel: Personen

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

    def vorstellen(self):
        print(f"Hallo, ich bin {self.name} und {self.alter} Jahre alt.")

    def geburtstag_feiern(self):
        self.alter += 1
        print(f"{self.name} ist jetzt {self.alter} Jahre alt.")



# Objekte der Klasse Person erstellen
anna = Person("Anna", 16)
jan = Person("Jan", 17)

# Methoden aufrufen
anna.vorstellen()  # Ausgabe: Hallo, ich bin Anna und 16 Jahre alt.
jan.vorstellen()   # Ausgabe: Hallo, ich bin Jan und 17 Jahre alt.

# Geburtstag feiern
anna.geburtstag_feiern()  # Ausgabe: Anna ist jetzt 17 Jahre alt.
```

Instanzen erstellen (initialisieren)

Beispiel: Personen

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

    def vorstellen(self):
        print(f"Hallo, ich bin {self.name} und {self.alter} Jahre alt.")

    def geburtstag_feiern(self):
        self.alter += 1
        print(f"{self.name} ist jetzt {self.alter} Jahre alt.")



# Objekte der Klasse Person erstellen
anna = Person("Anna", 16)
jan = Person("Jan", 17)

# Methoden aufrufen
anna.vorstellen()  # Ausgabe: Hallo, ich bin Anna und 16 Jahre alt.
jan.vorstellen()   # Ausgabe: Hallo, ich bin Jan und 17 Jahre alt.

# Geburtstag feiern
anna.geburtstag_feiern()  # Ausgabe: Anna ist jetzt 17 Jahre alt.
```

Instanzen erstellen (initialisieren)

Methode

Beispiel: Personen

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

    def vorstellen(self):
        print(f"Hallo, ich bin {self.name} und {self.alter} Jahre alt.")

    def geburtstag_feiern(self):
        self.alter += 1
        print(f"{self.name} ist jetzt {self.alter} Jahre alt.")



# Objekte der Klasse Person erstellen
anna = Person("Anna", 16)
jan = Person("Jan", 17)

# Methoden aufrufen
anna.vorstellen()  # Ausgabe: Hallo, ich bin Anna und 16 Jahre alt.
jan.vorstellen()   # Ausgabe: Hallo, ich bin Jan und 17 Jahre alt.

# Geburtstag feiern
anna.geburtstag_feiern()  # Ausgabe: Anna ist jetzt 17 Jahre alt.
```

Instanzen erstellen (initialisieren)

Methode

Methode

Aufgaben: Grundlagen von Klassen

- Definition 7.1, Beispiele 7.1–7.2
  - Code im Skript studieren und zwei weitere Instanzen von `Person` erstellen.

  Aufgabe 7.1
- (Klasse `Buch` mit `titel`, `autor`, `seitenzahl` und Methode `info()`)
- Aufgabe 7.2

  (Erweiterung der Klasse `Bankkonto` um die Methode `ueberweisen()`)



Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:



Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:
  - `int`, `float` (Zahlen)



Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:
  - `int`, `float` (Zahlen)
  - `str` (Zeichenketten)



FreeFlower

Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:
  - `int`, `float` (Zahlen)
  - `str` (Zeichenketten)
  - `list`, `dict`, `set`, `tuple` (Datenstrukturen)



Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:
  - `int`, `float` (Zahlen)
  - `str` (Zeichenketten)
  - `list`, `dict`, `set`, `tuple` (Datenstrukturen)
- **Automatische Typ-Erkennung** („Python-Magie“):

```
x = 5       # Python erkennt automatisch: Objekt der Klasse int
x = int(5)  # Aequivalente, explizite Schreibweise
```



Vordefinierte Klassen in Python

- Jede Variable in Python gehört zu einer **vordefinierten Klasse**:
  - `int`, `float` (Zahlen)
  - `str` (Zeichenketten)
  - `list`, `dict`, `set`, `tuple` (Datenstrukturen)
- **Automatische Typ-Erkennung** („Python-Magie“):

```
x = 5       # Python erkennt automatisch: Objekt der Klasse int
x = int(5)  # Aequivalente, explizite Schreibweise
```

- Mit der Funktion `dir()` werden alle verfügbaren Methoden und Attribute einer Klasse oder Instanz angezeigt:

```
print(dir(list))     # Zeigt alle Methoden der Klasse 'list'
print(dir("hallo"))  # Zeigt Methoden einer String-Instanz
```



Klassenattribute vs. Instanzattribute

- **Instanzattribute** (`self.attribut`): Gelten individuell für ein einzelnes Objekt (z.B. `self.name`, `self.alter`).



Klassenattribute vs. Instanzattribute

- **Instanzattribute** (`self.attribut`): Gelten individuell für ein einzelnes Objekt (z.B. `self.name`, `self.alter`).
- **Klassenattribute**: Werden direkt in der Klasse definiert und sind für **alle Instanzen gleich**.



Klassenattribute vs. Instanzattribute

- **Instanzattribute** (`self.attribut`): Gelten individuell für ein einzelnes Objekt (z.B. `self.name`, `self.alter`).
- **Klassenattribute**: Werden direkt in der Klasse definiert und sind für **alle Instanzen gleich**.



Klassenattribute vs. Instanzattribute

- **Instanzattribute** (`self.attribut`): Gelten individuell für ein einzelnes Objekt (z.B. `self.name`, `self.alter`).
- **Klassenattribute**: Werden direkt in der Klasse definiert und sind für **alle Instanzen gleich**.

```
class Schueler:
    schule = "Kantonsschule im Lee" # Klassenattribut fuer alle

    def __init__(self, name, klasse):
        self.name = name            # Instanzattribut
        self.klasse = klasse        # Instanzattribut

s1 = Schueler("Lisa", "3a")
s2 = Schueler("Tim", "4b")
print(s1.schule) # Ausgabe: Kantonsschule im Lee

Schueler.schule = "KLW" # Aenderung wirkt auf alle Instanzen!
print(s2.schule) # Ausgabe: KLW
```

FreeFlower

Instanzzähler & `@classmethod`

- Klassenattribute eignen sich ideal zum **Zählen von Instanzen**.
- Mit `@classmethod` definieren wir Methoden, die sich auf die Klasse (`cls`) beziehen.

```
class Produkt:
    anzahl_produkte = 0 # Klassenattribut

    def __init__(self, name, preis):
        self.name = name
        self.preis = preis
        Produkt.anzahl_produkte += 1 # Bei jeder neuen Instanz +1

    @classmethod
    def get_anzahl_produkte(cls):
        print("Anzahl Produkte:", cls.anzahl_produkte)

p1 = Produkt("Laptop", 1200)
p2 = Produkt("Maus", 30)
Produkt.get_anzahl_produkte() # Ausgabe: Anzahl Produkte: 2
```



Aufgaben: Klassenattribute

- Kapitel 7.2 & 7.3: Vordefinierte Klassen & Klassenattribute
- Aufgabe 7.3

  (Klasse `Auto` mit Instanzattributen `marke`, `modell`, `baujahr`, Klassenattribut `anzahl_autos` und Klassenmethode `get_statistik()`)



Vererbung (Inheritance)

Definition Vererbung

Bei der **Vererbung** erbt eine Kindklasse (Subklasse) Attribute und Methoden von einer Elternklasse (Superklasse). Sie kann eigene Attribute/Methoden hinzufügen oder bestehende Methoden überschreiben.



Vererbung (Inheritance)

Definition Vererbung

Bei der **Vererbung** erbt eine Kindklasse (Subklasse) Attribute und Methoden von einer Elternklasse (Superklasse). Sie kann eigene Attribute/Methoden hinzufügen oder bestehende Methoden überschreiben.

- Syntax für Vererbung: `class Kindklasse(Elternklasse):`



Vererbung (Inheritance)

Definition Vererbung

Bei der **Vererbung** erbt eine Kindklasse (Subklasse) Attribute und Methoden von einer Elternklasse (Superklasse). Sie kann eigene Attribute/Methoden hinzufügen oder bestehende Methoden überschreiben.

- Syntax für Vererbung: `class Kindklasse(Elternklasse):`
- Mit `super().__init__(...)` wird der Konstruktor der Elternklasse aufgerufen.



Vererbung (Inheritance)

Definition Vererbung

Bei der **Vererbung** erbt eine Kindklasse (Subklasse) Attribute und Methoden von einer Elternklasse (Superklasse). Sie kann eigene Attribute/Methoden hinzufügen oder bestehende Methoden überschreiben.

- Syntax für Vererbung: `class Kindklasse(Elternklasse):`
- Mit `super().__init__(...)` wird der Konstruktor der Elternklasse aufgerufen.
- Vermeidet Code-Duplizierung und ermöglicht modulare Programmierstrukturen.



Beispiel: Vererbung

```python
class Fahrzeug:
    def __init__(self, marke, modell):
        self.marke = marke
        self.modell = modell
    def info(self):
        return f"{self.marke} {self.modell}"

class ElektroAuto(Fahrzeug): # ElektroAuto erbt von Fahrzeug
    def __init__(self, marke, modell, batterie_kapazitaet):
        super().__init__(marke, modell) # Eltern-Konstruktor
        self.batterie_kapazitaet = batterie_kapazitaet

    def info(self): # Methode der Elternklasse ueberschreiben
        basis = super().info()
        return f"{basis}, Kapazitaet: {self.batterie_kapazitaet} kWh"

e_car = ElektroAuto("Tesla", "Model 3", 75)
print(e_car.info()) # Tesla Model 3, Kapazitaet: 75 kWh
```

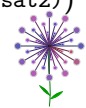


Aufgaben: Vererbung

- Kapitel 7.4.1: Vererbung
- Aufgabe 7.4

  (Elternklasse `Person`, Kindklasse `Schueler` mit zusätzlichem Attribut `schulklasse`, überschriebener Methode `vorstellen()` und Zuweisungsmethode)
- Aufgabe 7.5

  (Basisklasse `Bankkonto`, Unterklasse `Sparkonto` mit `zinsen_gutschreiben(zinssatz)`)



Polymorphismus („Viele Formen“)

Definition Polymorphismus

Verschiedene Klassen können Methoden mit **demselben Namen** besitzen, aber jeweils eine eigene, spezifische Implementierung haben (*poly* = viel, *morphos* = Form).



Polymorphismus („Viele Formen“)

Definition Polymorphismus

Verschiedene Klassen können Methoden mit **demselben Namen** besitzen, aber jeweils eine eigene, spezifische Implementierung haben (*poly* = viel, *morphos* = Form).

```
class Tier:
    def geraeusch(self):
        return "Ein Tier macht ein Geraeusch"

class Hund(Tier):
    def geraeusch(self):
        return "Wuff!"

class Katze(Tier):
    def geraeusch(self):
        return "Miau!"

tiere = [Tier(), Hund(), Katze()]
for t in tiere:
    print(t.geraeusch()) # Jedes Objekt ruft seine eigene Methode auf!
```

FreeFlower

Praktisches Beispiel: Bibliothekssystem

- In komplexen Anwendungen arbeiten **mehrere Klassen zusammen**:
  - `Buch`: Repräsentiert ein Buch (Titel, Autor, ISBN, Status)
  - `Bibliotheksmitglied`: Repräsentiert einen Ausleihenden
  - `Bibliothek`: Verwaltungsklasse für Sammlungen von Büchern und Mitgliedern



Praktisches Beispiel: Bibliothekssystem

- In komplexen Anwendungen arbeiten **mehrere Klassen zusammen**:
  - `Buch`: Repräsentiert ein Buch (Titel, Autor, ISBN, Status)
  - `Bibliotheksmitglied`: Repräsentiert einen Ausleihenden
  - `Bibliothek`: Verwaltungsklasse für Sammlungen von Büchern und Mitgliedern

```
class Bibliothek:
    def __init__(self, name):
        self.name = name
        self.buecher = []
        self.mitglieder = []

    def buch_hinzufuegen(self, buch):
        self.buecher.append(buch)
        print(f"Buch '{buch.titel}' zur Bibliothek hinzugefuegt.")
```



Aufgaben: Bibliothekssystem & Challenge

- Kapitel 7.5: Bibliothekssystem
- Aufgabe 7.6

  (Klasse `Zeitschrift` erbt von `Buch` mit zusätzlicher `ausgabe` und überschriebener `info()`-Methode)
- Challenge 7.7

  (Vollständiges Lagerverwaltungssystem mit Klassen `Produkt`, `Elektronikprodukt`, `Lebensmittel`, `Lager`, `Bestellung`)



Zusammenfassung: Klassen & OOP

- **Klasse**: Bauplan für eigene Datentypen mit Attributen und Methoden.



Zusammenfassung: Klassen & OOP

- **Klasse**: Bauplan für eigene Datentypen mit Attributen und Methoden.
- **Instanz / Objekt**: Konkretes Exemplar (`self`).



Zusammenfassung: Klassen & OOP

- **Klasse**: Bauplan für eigene Datentypen mit Attributen und Methoden.
- **Instanz / Objekt**: Konkretes Exemplar (`self`).
- **Klassenattribute & `@classmethod`**: Für alle Instanzen einer Klasse gemeinsam.



Zusammenfassung: Klassen & OOP

- **Klasse**: Bauplan für eigene Datentypen mit Attributen und Methoden.
- **Instanz / Objekt**: Konkretes Exemplar (`self`).
- **Klassenattribute & `@classmethod`**: Für alle Instanzen einer Klasse gemeinsam.
- **Vererbung**: Kindklassen erben Eigenschaften von Elternklassen (`super()`).



Zusammenfassung: Klassen & OOP

- **Klasse**: Bauplan für eigene Datentypen mit Attributen und Methoden.
- **Instanz / Objekt**: Konkretes Exemplar (`self`).
- **Klassenattribute & `@classmethod`**: Für alle Instanzen einer Klasse gemeinsam.
- **Vererbung**: Kindklassen erben Eigenschaften von Elternklassen (`super()`).
- **Polymorphismus**: Gleicher Methodenname, unterschiedliches Verhalten.

