# Python – Lektion 6
# Klassen

# Rückblick

- Dateien lesen und schreiben
  - Datei mit der `open()`-Funktion öffnen:

```
# Lesen
with open("dokument.txt") as f:
# Lesen
with open("dokument.txt", "r") as f:
# Schreiben
with open("dokument.txt", "w") as f:
# Anhängen
with open("dokument.txt", "a") as f:
# Binär lesen
with open("dokument.txt", "rb") as f:
# Binär schreiben
with open("dokument.txt", "wb") as f:
```

  - Weitere Parameter findet man in der Hilfe[1]

[1] `https://docs.python.org/3/library/functions.html#open`

# Rückblick

- Dateien lesen und schreiben
  - Datei lesen:

```
inhalt = f.read()           # gesamte Datei lesen
inhalt = f.read(n)          # n Zeichen lesen
zeilen = f.readlines()      # Liste aller Zeilen
```

  - Datei schreiben:

```
f.write("hello")            # String schreiben
f.writelines(["1\n", "2\n"]) # Liste von Strings
```

  - Datei schliessen:

```
f.close()
```

# Rückblick

- Pathlib Modul
  - Pfadobjekte erzeugen

```
# Pfad auf das aktuelle Arbeitszverzeichnis
my_cwd = pathlib.Path.cwd()
# Pfad auf das aktuelle Homeverzeichnis
my_cwd = pathlib.Path.home()
# Pfad aus einem String erzeugen
my_cwd = pathlib.Path(r"D:\Documents\")
# Pfad mit / Operator zusammensetzen
my_cwd = pathlib.Path.cwd() / "FS2021" / "Python"
# Pfad mit joinpath() Methode zusammensetzen
my_cwd = pathlib.Path.cwd().joinpath("FS2021",
    "Python")
```

# Rückblick

- Pathlib Modul
  - Pfadkomponenten extrahieren

```
# Extrahierung des Dateinamens
my_path.name
# Extrahierung des übergeordneten Pfades
my_path.parent
# Extrahierung des Dateinamens ohne den Dateityp
my_path.stem
# Extrahierung des Dateityps
my_path.suffix
# Extrahierung des Drives ohne Verzeichnisse
my_path.anchor
```

# Rückblick

- Pathlib Modul
  - Weitere nützliche Methoden

```
# Datei öffnen und lesen
my_file.read_text()
# Datei öffnen und schreiben
my_file.write_text(text)
# Verzeichnis erstellen
my_path.mkdir()
# Verzeichnis löschen
my_path.rmdir()
# Datei verschieben
old_path.replace(new_path)
# Prüfen, ob ein Pfad/Datei bereits existiert
new_path.exists()
# Prüfen, ob der Pfad auf eine Datei zeigt
my_path.is_file()
# Prüfen, ob der Pfad auf einen Ordner zeigt.
my_path.is_dir()
```

# Rückblick

- Listen im Detail
  - Elemente hinzufügen:

```
liste.append("x")
liste.insert(2,"y")
liste += [3, 4]
liste.extend([5, 6])
```

  - Elemente ersetzen:

```
liste[1] = "B"
liste[3:] = ["C", "D"]
```

  - Elemente entfernen:

```
element = liste.pop()
element = liste.pop(0)
element = liste.remove("D")
```

  - Elemente sortieren:

```
sortiert = sorted(liste)
```

# Rückblick

- Dictionaries im Detail
  - Auf Werte/Schlüssel zugreifen:

```
title = book["Title"]
title = book.get("Title")
author = book.get("Author", "not provided")
all_values = book.values()
all_keys = book.keys()
all_items = book.items()
```

  - Elemente hinzufügen:

```
book["Title"] = "Numerisches Python"
book.setdefault("Erscheinungsjahr", 2017)
```

  - Elemente entfernen:

```
book.pop("Author")
book.popitem()
book.clear()
```

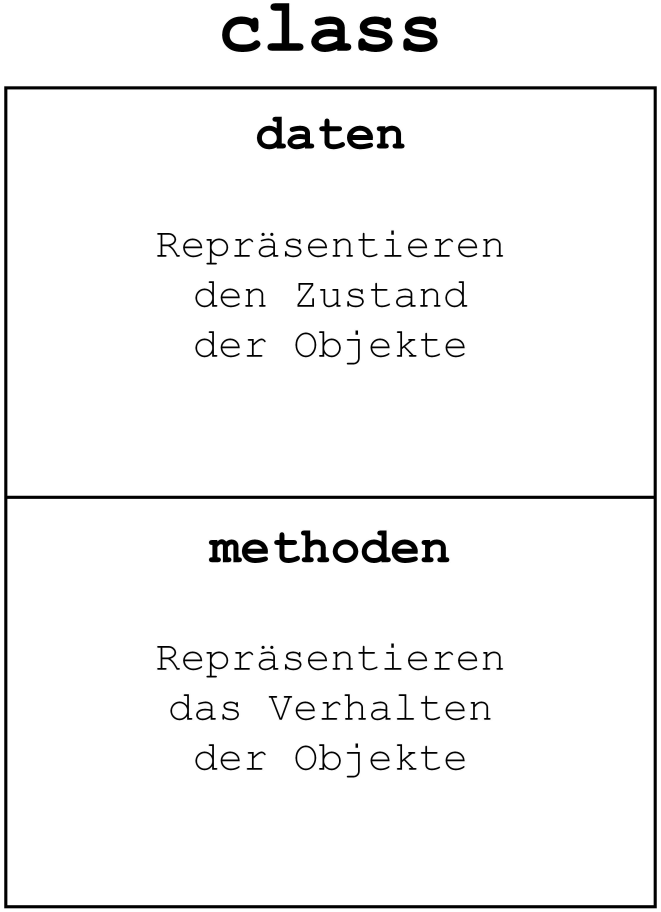
# Heutige Themen

- Klassen definieren
- Datenabstraktion
- Klassen testen

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.



# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Attribute und Methoden.


```
# Instanzierung eines Objekts
katze1 = Katze()
```

.

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.


```
# Das kennen Sie bereits:
liste1 = list()
```

.

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.


```
# Das kennen Sie bereits:
liste1 = list()
```

```
# Methode aufrufen
liste1.append(3)
```

.

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.


```
# Instanzierung erstes Objekt
katze1 = Katze()

# Daten anpassen
katze1.name = "Garfield"
katze1.fellfarbe = "orange"
katze1.gewicht = 25


# Methode aufrufen
katze1.schlafen()


.
```

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.


```
# Instanzierung zweites Objekt
katze2 = Katze()

# Daten anpassen
katze2.name = "Tom"
katze2.fellfarbe = "grau"
katze2.gewicht = 16


# Methode aufrufen
katze2.schlafen()


.
```

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.


```
class Katze:
    def __init__(self):
        self.name = ""
        self.fellfarbe = ""
        self.gewicht = 0

    def fressen(self, futter):
        self.gewicht += futter

    def schlafen(self):
        print("ZzzZzzzZ")
```

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.

```
class Katze:
    def __init__(self):
        self.name = ""
        self.fellfarbe = ""
        self.gewicht = 0

    def fressen(self, futter):
        self.gewicht += futter

    def schlafen(self):
        print("ZzzZzzzZ")
```

```
# Objekt erzeugen
katze1 = Katze()

# Der Aufruf...
katze1.fressen(5)
# ist äquivalent zu ...
Katze.fressen(katze1,5)
# Der Aufruf...
katze1.schlafen()
# ist äquivalent zu ...
Katze.schlafen(katze1)
.
```

# Klassen

Eine Klasse ...

- ... ist eine formale Beschreibung der Struktur eines Objektes.
- ... hat Daten und Methoden.

`http://localhost:8888/notebooks/klassen.ipynb`

# Datenabstraktion

- Datenabstraktion = Datenkapselung + Geheimnisprinzip
- Datenkapselung (Zugriff kontrollieren)
  - Getter- und Setter-Methoden
- Geheimnisprinzip (interne Information verstecken)
  - public
  - protected
  - private

`http://localhost:8888/notebooks/datenabstraktion.ipynb`

# Klassen testen

- Klassen werden in separate Pythondateien gespeichert
- Testcode in die gleiche Datei integrieren
- Testcode in eine `if`-Anweisung platzieren:

```
1 if __name__ == '__main__':
2     Testcode
```

# Eigenes Modul importieren

- Klasse aus einer separaten Pythondatei importieren

`http://localhost:8888/notebooks/modul_importieren.ipynb`