ETH zürich

Departement Informatik

Programmieren und Problemlösen

Objektorientierung

Dennis Komm

Frühling 2021 – 20. Mai 2021

Klassen und Objekte

Python-Klassen

Klassen – Technisch

Eine Klasse ist eine Einheit mit einem **Namen**, die **Daten** und **Funktionalität** beinhaltet

Klassen – Technisch

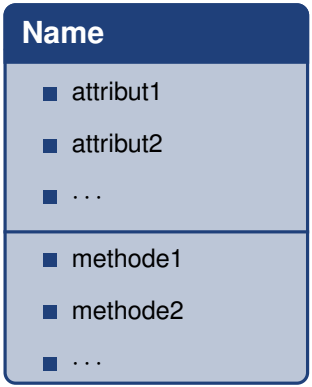
Eine Klasse ist eine Einheit mit einem **Namen**, die **Daten** und **Funktionalität** beinhaltet

- Eine Klasse definiert einen neuen **Datentyp**
- *Daten:* Gespeichert in Variablen, genannt **Attribute**
- *Funktionalität:* in Form von Funktionen, genannt **Methoden**
- Klassen sind ofmals separate `.py`-Dateien mit gleichem Namen

Klassen – Technisch

Eine Klasse ist eine Einheit mit einem **Namen**, die **Daten** und **Funktionalität** beinhaltet

- Eine Klasse definiert einen neuen **Datentyp**
- *Daten:* Gespeichert in Variablen, genannt **Attribute**
- *Funktionalität:* in Form von Funktionen, genannt **Methoden**
- Klassen sind ofmals separate `.py`-Dateien mit gleichem Namen



Klassen – Konzeptuell

Klassen erlauben es, Daten, die inhaltlich **zusammengehören**, zu einem Datentyp **zusammenzufassen**

Klassen – Konzeptuell

Klassen erlauben es, Daten, die inhaltlich **zusammengehören**, zu einem Datentyp **zusammenzufassen**

Klassen bieten Funktionalitäten an, welche **Abfragen** basierend auf den Daten oder **Operationen** auf den Daten ermöglichen

Klassen – Konzeptuell

Klassen erlauben es, Daten, die inhaltlich **zusammengehören**, zu einem Datentyp **zusammenzufassen**

Klassen bieten Funktionalitäten an, welche **Abfragen** basierend auf den Daten oder **Operationen** auf den Daten ermöglichen

Beispiel

- Zusammenhängende Messdaten
- Funktionen, um Daten auszulesen
- Funktionen, um Daten zu bearbeiten

Beispiel – Erdbebendaten

Schweizerischer Erdbebendienst
Service Sismologique Suisse
Servizio Sismico Svizzero
Swiss Seismological Service

SED > *Earthquake catalog* > Query the catalogue

Earthquake catalog

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml	Io	Ix	epicentral area
»	2001/01/01	00:03:47.8	certain	earthquake	45.53	6.75	RENASS/BCSF (2009)	5.	1.52	0.9			SSE BEAUFORT(73)
»	2001/01/01	00:20:01.5	uncertain	earthquake	47.51	9.48	LED (2009)	10.	2.17	1.99			
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3			
»	2001/01/07	18:55:18.3	certain	earthquake	48.05	9.03	LED (2009)	15.	1.82	1.41			
»	2001/01/07	20:55:36.5	certain	earthquake	46.564	10.29	SED (ECOS-09)	5.	1.94	1.6			

Beispiel – Erdbebendaten

Schweizerischer Erdbebendienst
Service Sismologique Suisse
Servizio Sismico Svizzero
Swiss Seismological Service

SED > *Earthquake catalog* > Query the catalogue

Earthquake catalog

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml	Io	Ix	epicentral area
»	2001/01/01	00:03:47.8	certain	earthquake	45.53	6.75	RENASS/BCSF (2009)	5.	1.52	0.9			SSE BEAUFORT(73)
»	2001/01/01	00:20:01.5	uncertain	earthquake	47.51	9.48	LED (2009)	10.	2.17	1.99			
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3			
»	2001/01/07	18:55:18.3	certain	earthquake	48.05	9.03	LED (2009)	15.	1.82	1.41			
»	2001/01/07	20:55:36.5	certain	earthquake	46.564	10.29	SED (ECOS-09)	5.	1.94	1.6			

Klasse für Messwert – Erster Versuch

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3

Klasse für Messwert – Erster Versuch

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3

Python-Klasse `Measurement`

```
class Measurement:
    date = ""
    time = ""

    latitude = 0
    longitude = 0

    magnitude = 0
```

Klasse für Messwert – Erster Versuch

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3

Python-Klasse `Measurement`

```
class Measurement:
    date = ""
    time = ""

    latitude = 0
    longitude = 0

    magnitude = 0
```

Name der Klasse / des Datentyps

Klasse für Messwert – Erster Versuch

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3

Python-Klasse `Measurement`

```
class Measurement:
    date = ""
    time = ""

    latitude = 0
    longitude = 0

    magnitude = 0
```

Attribute gemäss CSV-Kopfdaten

Klasse für Messwert – Erster Versuch

link	date	time	appraisal	event type	lat [°N]	lon [°E]	source agency	depth	Mw	Ml
»	2001/01/03	11:11:20.4	certain	earthquake	46.446	9.982	SED (ECOS-09)	4.	2.36	2.3

Python-Klasse `Measurement`

```
class Measurement:
    date = ""
    time = ""

    latitude = 0
    longitude = 0

    magnitude = 0
```

Measurement

- `date` (Leerer String λ)
- `time` (Leerer String λ)
- `latitude` (Zahl 0)
- `longitude` (Zahl 0)
- `magnitude` (Zahl 0)

Klassen und Objekte

Python-Objekte

Objekte – Instanzen von Klassen

Klassen beschreiben den Aufbau von Objekten, eine Art **Bauplan**

⇨ Vergleichbar mit den **Kopfdaten** aus der CSV-Datei

Objekte – Instanzen von Klassen

Klassen beschreiben den Aufbau von Objekten, eine Art **Bauplan**
⇨ Vergleichbar mit den **Kopfdaten** aus der CSV-Datei

Objekte werden instanziert nach Bauplan und enthalten nun Werte
⇨ Vergleichbar mit den einzelnen **Datenzeilen** aus der CSV-Datei

Objekte – Instanzen von Klassen

Klassen beschreiben den Aufbau von Objekten, eine Art **Bauplan**
⇨ Vergleichbar mit den **Kopfdaten** aus der CSV-Datei

Objekte werden instanziert nach Bauplan und enthalten nun Werte
⇨ Vergleichbar mit den einzelnen **Datenzeilen** aus der CSV-Datei

Beispiel

- Variablen für Parameter der Messungen
- Funktion, um Messung übersichtlich darzustellen
- Funktion, um Messungen zu vergleichen

Objektinstanzierung

Objekte sind Instanzen von Klassen

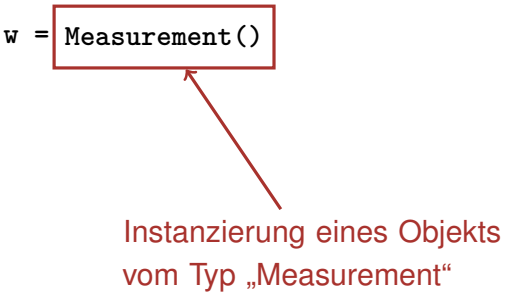
Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()
```

Objektinstanzierung

Objekte sind Instanzen von Klassen



Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()
```

Instanzierung eines Objekts vom Typ „Measurement“

Measurement `w`

date	λ
time	λ
latitude	0
longitude	0
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
```

Measurement `w`

date	2001/01/03
time	λ
latitude	0
longitude	0
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
```

Punktoperator; `instanz.attribut`

Measurement `w`

date	2001/01/03
time	λ
latitude	0
longitude	0
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
w.time = "11:11:20.4"
```

Measurement `w`

date	2001/01/03
time	11:11:20.4
latitude	0
longitude	0
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
w.time = "11:11:20.4"
w.latitude = 46.446
```

Measurement w

date	2001/01/03
time	11:11:20.4
latitude	46.446
longitude	0
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
w.time = "11:11:20.4"
w.latitude = 46.446
w.longitude = 9.982
```

Measurement `w`

date	2001/01/03
time	11:11:20.4
latitude	46.446
longitude	9.982
magnitude	0

Objektinstanzierung

Objekte sind Instanzen von Klassen

```
w = Measurement()

w.date = "2001/01/03"
w.time = "11:11:20.4"
w.latitude = 46.446
w.longitude = 9.982
w.magnitude = 2.36
```

Measurement `w`

date	2001/01/03
time	11:11:20.4
latitude	46.446
longitude	9.982
magnitude	2.36

Klasse für Messwert – Zweiter Versuch

Measurement

- `date`
- `time`
- `latitude`
- `longitude`
- `magnitude`

Klasse für Messwert – Zweiter Versuch

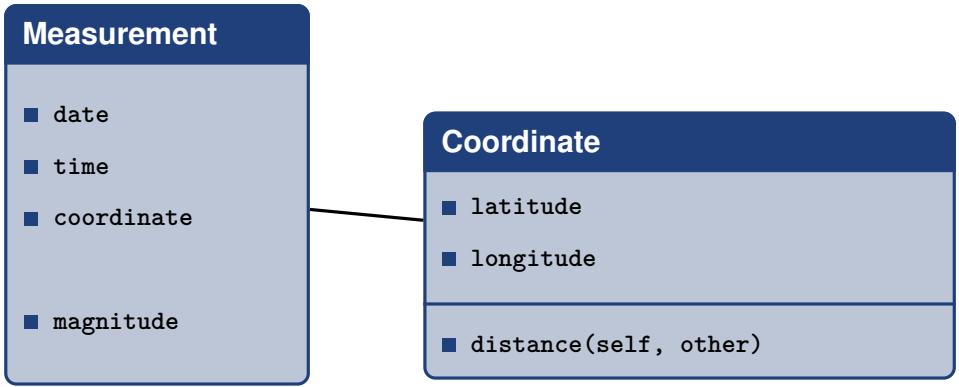
Measurement

- `date`
- `time`
- `latitude`
- `longitude`
- `magnitude`

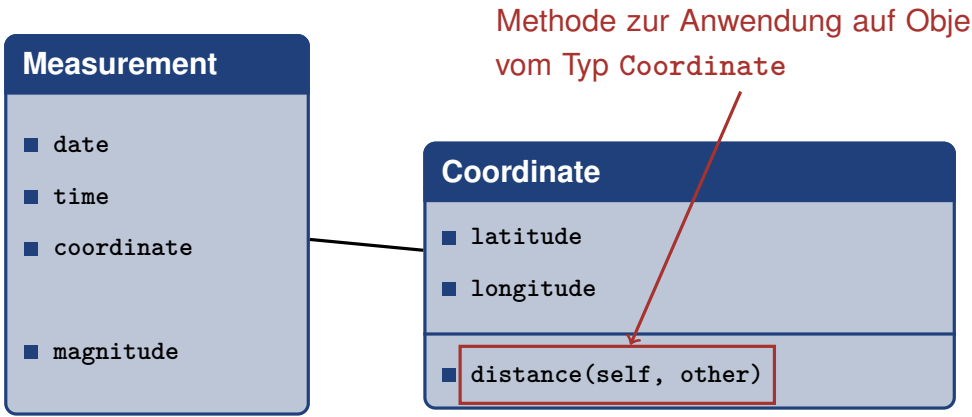
Bessere Strukturierung

- Breitengrad und Längengrad gehören in einen Datentyp `Coordinate`
- Objekt vom Typ `Measurement` hat ein Attribut vom Typ `Coordinate`
- „Komposition“

Klasse für Messwert – Zweiter Versuch



Klasse für Messwert – Zweiter Versuch



Methoden

- Methoden sind Funktionen, die in einer Klasse definiert sind

Methoden

- Methoden sind Funktionen, die in einer Klasse definiert sind
- Der erste Parameter einer Methode ist immer `self`, womit auf die aktuelle Instanz zugegriffen werden kann

Methoden

- Methoden sind Funktionen, die in einer Klasse definiert sind
- Der erste Parameter einer Methode ist immer `self`, womit auf die aktuelle Instanz zugegriffen werden kann
- Wieder **Punktoperator**; Aufruf analog zu `append()` bei Listen

Methoden

- Methoden sind Funktionen, die in einer Klasse definiert sind
- Der erste Parameter einer Methode ist immer `self`, womit auf die aktuelle Instanz zugegriffen werden kann
- Wieder **Punktoperator**; Aufruf analog zu `append()` bei Listen
- Vordefinierte Funktionen mit spezieller Funktionalität

Methoden

- Methoden sind Funktionen, die in einer Klasse definiert sind
- Der erste Parameter einer Methode ist immer `self`, womit auf die aktuelle Instanz zugegriffen werden kann
- Wieder **Punktoperator**; Aufruf analog zu `append()` bei Listen
- Vordefinierte Funktionen mit spezieller Funktionalität
- Funktion `__str__` definiert, was ausgegeben wird, wenn eine Instanz an `print()` übergeben wird

```
class Coordinate:
    def __str__(self):
        return "Dies ist eine Koordinate"
```

Methoden in Klassen

```
from math import *

class Coordinate:

    latitude = 0
    longitude = 0

    def __str__(self):
        return "Dies ist eine Koordinate"

    # Computes the distance to the provided coordinate 'other' in kilometers
    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Methoden in Klassen

```
from math import *

class Coordinate:

    latitude = 0
    longitude = 0

    def __str__(self):
        return "Dies ist eine Koordinate"

    # Computes the distance to the provided coordinate 'other' in kilometers
    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Erster Parameter ist immer `self`

Methoden in Klassen

```
from math import *

class Coordinate:

    latitude = 0
    longitude = 0

    def __str__(self):
        return "Dies ist eine Koordinate"

    # Computes the distance to the provided coordinate 'other' in kilometers
    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Erlaubt, innerhalb Methoden einer Klasse, auf das aktuelle Objekt zuzugreifen

Methoden in Klassen

```
from math import *

class Coordinate:

    latitude = 0
    longitude = 0

    def __str__(self):
        return "Dies ist eine Koordinate"

    # Computes the distance to the provided coordinate 'other' in kilometers
    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Haversine-Formel

Klassen und Objekte

Konstruktoren

Konstruktoren

Erstellen einer `Coordinate` erfordert drei Schritte

Konstruktoren

Erstellen einer `Coordinate` erfordert drei Schritte

```
k = Coordinate()
k.latitude = 45.97
k.longitude = 7.65
```

Konstruktoren

Erstellen einer `Coordinate` erfordert drei Schritte

```
k = Coordinate()
k.latitude = 45.97
k.longitude = 7.65
```

Konstruktoren erlauben es, einfacher die Initialwerte der Felder eines neu erstellten Objektes zu setzen

```
k = Coordinate(45.97, 7.65)
```

Konstruktoren

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)              # Umrechnung von Gradmass
        self.longitude = radians(deg_longitude)            # in Bogenmass

    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Konstruktoren

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)              # Umrechnung von Gradmass
        self.longitude = radians(deg_longitude)            # in Bogenmass

    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

```
zurich = Coordinate(47.36667, 8.55)
brisbane = Coordinate(-27.46794, 153.02809)
print(int(zurich.distance(brisbane)))
```

Konstruktoren

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)
        self.longitude = radians(deg_longitude)

    def distance(self, other):
        dlat = self.latitude - other.latitude
        dlon = self.longitude - other.longitude
        Hav = sin(dlat / 2)**2 + cos(self.latitude) * cos(other.latitude) * sin(dlon / 2)**2
        return 6373 * 2 * atan2(sqrt(Hav), sqrt(1 - Hav))
```

Wird ausgeführt, wenn Objekt erzeugt wird; Parameterwerte werden an diese Funktion übergeben

```
# Umrechnung von Gradmass
# in Bogenmass
```

```
zurich = Coordinate(47.36667, 8.55)
brisbane = Coordinate(-27.46794, 153.02809)
print(int(zurich.distance(brisbane)))
```

Erdbeben-Datenbank verwalten

Erdbeben-Datenbank verwalten

1. Erstelle Datenstruktur, um Erdbeben zu repräsentieren
2. Lies CSV-Datei ein, erstelle aus Zeilen Objekte, füge diese in Dictionary ein
3. Erstelle User-Interface, mit dem Daten abgefragt werden können

Erdbeben-Datenbank verwalten

1. Erstelle Datenstruktur, um Erdbeben zu repräsentieren
2. Lies CSV-Datei ein, erstelle aus Zeilen Objekte, füge diese in Dictionary ein
3. Erstelle User-Interface, mit dem Daten abgefragt werden können

```
30274940.00000; 2001/01/20 15:49:10; certain; earthquake; 45.856; 8.142; "SED (ECOS-09)"; 13.; 2.56; 2.6;
```

Erdbeben-Datenbank verwalten

1. Erstelle Datenstruktur, um Erdbeben zu repräsentieren
2. Lies CSV-Datei ein, erstelle aus Zeilen Objekte, füge diese in Dictionary ein
3. Erstelle User-Interface, mit dem Daten abgefragt werden können

```
30274940.00000; 2001/01/20 15:49:10; certain; earthquake; 45.856; 8.142; "SED (ECOS-09)"; 13.; 2.56; 2.6;
```

Interessant sind

- Index 0: Schlüssel für Dictionary; wird in natürliche Zahl umgewandelt
- Index 1: Datum und Uhrzeit; wird am Leerzeichen unterteilt
- Index 4: Längengrad; wird in Kommazahl umgewandelt
- Index 5: Breitengrad; wird in Kommazahl umgewandelt
- Index 9: Stärke auf Richter-Skala; wird in Kommazahl umgewandelt

Erdbeben-Datenbank verwalten

1. Erstelle Datenstruktur, um Erdbeben zu repräsentieren

Erdbeben-Datenbank verwalten

1. Erstelle Datenstruktur, um Erdbeben zu repräsentieren

```
class Coordinate:
    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)
        self.longitude = radians(deg_longitude)
    def __str__(self):
        return str(self.latitude) + ", " + str(self.longitude)

class Measurement:
    def __init__(self, date, time, magnitude, coordinate):
       self.date = date
       self.time = time
       self.magnitude = magnitude
       self.coordinate = coordinate
    def __str__(self):
       return "Erdbeben der Stärke " + str(self.magnitude) + ", gemessen am " \
         + str(self.date) + " um " + str(self.time) + " an Position " + str(self.coordinate)
```

Erdbeben-Datenbank verwalten

2. Lies CSV ein, erstelle aus Zeilen Objekte, füge diese in Dictionary ein

Erdbeben-Datenbank verwalten

2. Lies CSV ein, erstelle aus Zeilen Objekte, füge diese in Dictionary ein

```
def read_measurements(filename):

    # Datei Zeile für Zeile einlesen
    with open(filename) as file:
        lines = file.read().splitlines()
    measurements = {}

    # Alle Zeilen nacheinander verarbeiten
    for i in range(1, len(lines)):
        tmp = lines[i].split(";")
        tmp_coord = Coordinate(float(tmp[4]), float(tmp[5]))
        tmp_date_time = tmp[1].split(" ")
        tmp_magnitude = float(tmp[9])
        tmp_meas = Measurement(tmp_date_time[1], tmp_date_time[2], tmp_magnitude, tmp_coord)
        measurements[int(float(tmp[0]))] = tmp_meas

    return measurements
```

Erdbeben-Datenbank verwalten

3. Erstelle User-Interface, mit dem Daten abgefragt werden können

Erdbeben-Datenbank verwalten

3. Erstelle User-Interface, mit dem Daten abgefragt werden können

```
earthquakes = read_measurements("earthquakes.csv")

while True:
    user_input = input("Geben Sie eine Erdbeben-ID ein (Abbrechen mit exit): ")
    if user_input == "exit":
        print("Programm beendet.")
        break
    else:
        quake_id = int(user_input)
        if quake_id not in earthquakes:
            print("Erdbeben-ID nicht gefunden.")
        else:
            print(earthquakes[quake_id])
```

Studierenden-Datenbank verwalten

Aufgabe – Studierenden-Datenbank verwalten

Schreiben Sie eine Klasse, die Studierende repräsentiert mit Attributen

- `student_id`
- `name`
- `grade`

- Erlauben Sie der Nutzerin, mit `input()` Studierende-Objekte anzulegen
- Speichern Sie diese in einem Dictionary
- Geben Sie alle Studierenden mit einer `for ... in`-Schleife aus

Aufgabe – Studierenden-Datenbank verwalten

```
class Student:

  def __init__(self, s_id, name, grade):
    self.s_id = s_id
    self.name = name
    self.grade = grade

  def __str__(self):
    return "Die / Der Studierende "
      + str(self.name)
      + " (" + str(self.s_id)
      + ") hat die Note "
      + str(self.grade)
      + " erhalten."
```

```
students = {}

while True:
  user_input = input("Weitere Daten eingeben? [J/N]")
  if user_input == "J":
    tmp_id = int(input(" ID: "))
    tmp_name = input(" Name: ")
    tmp_grade = float(input(" Note: "))
    tmp_student = Student(tmp_id, tmp_name, tmp_grade)
    students[tmp_id] = tmp_student
  elif user_input == "N":
    print("Programm beendet.")
    break
  else:
    print("Ungültige Eingabe.")

for id in students:
  print(students[id])
```

Heaps

Listen und Dictionarys

Komplexität von Operationen auf Listen und Dictionarys mit n Elementen

Listen

Zugriff mit `[]`	$\mathcal{O}(1)$
Einfügen mit `append()`	$\mathcal{O}(1)$
Einfügen mit `insert()`	$\mathcal{O}(n)$
Löschen mit `pop(0)`	$\mathcal{O}(1)$
Löschen mit `pop()`	$\mathcal{O}(1)$
Minimum finden	$\mathcal{O}(n)$

Dictionarys

Zugriff mit `[]`	$\mathcal{O}(1)$
Einfügen mit `[]`	$\mathcal{O}(1)$
Minimum finden	$\mathcal{O}(n)$

Heaps

Gesucht wird eine Datenstruktur für eine besondere Anwendung

⇨ Minimum soll schnell berechnet werden (Bei Listen und Dictionarys $\mathcal{O}(n)$)

Heaps

Gesucht wird eine Datenstruktur für eine besondere Anwendung
⇨ Minimum soll schnell berechnet werden (Bei Listen und Dictionarys $\mathcal{O}(n)$)

Datenstruktur mit folgenden Operationen	
Einfügen	$\mathcal{O}(\log n)$
Minimum abrufen	$\mathcal{O}(1)$
Minimum entfernen	$\mathcal{O}(\log n)$

Heaps

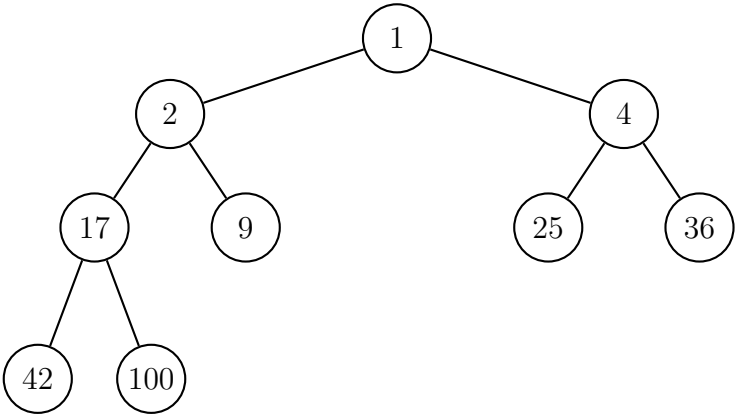
Gesucht wird eine Datenstruktur für eine besondere Anwendung
⇨ Minimum soll schnell berechnet werden (Bei Listen und Dictionarys $\mathcal{O}(n)$)

Datenstruktur mit folgenden Operationen

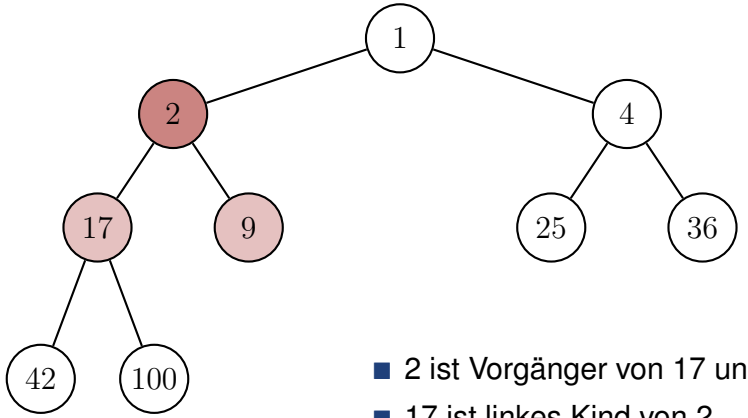
Einfügen	$\mathcal{O}(\log n)$
Minimum abrufen	$\mathcal{O}(1)$
Minimum entfernen	$\mathcal{O}(\log n)$

- Verwende „Baum“
- Bette diesen geschickt in Liste ein
- Wurzel (erstes Listen-Element) enthält kleinstes Element
- Nach Entfernen der Wurzel muss Baum umgebaut werden
- Beim Einfügen muss Baum ebenfalls umgebaut werden

Heaps

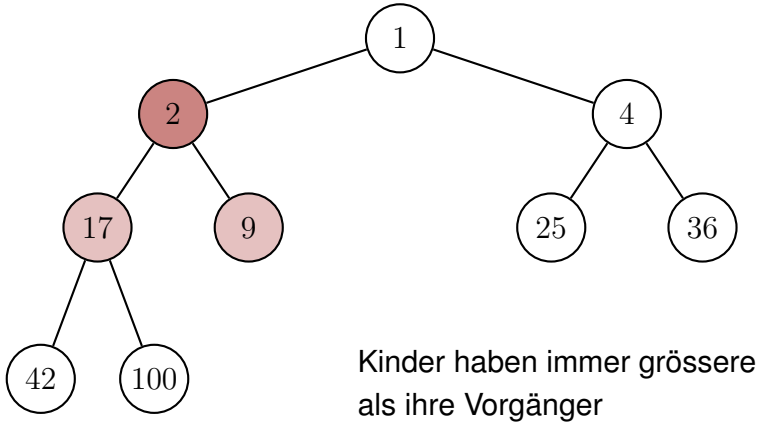


Heaps



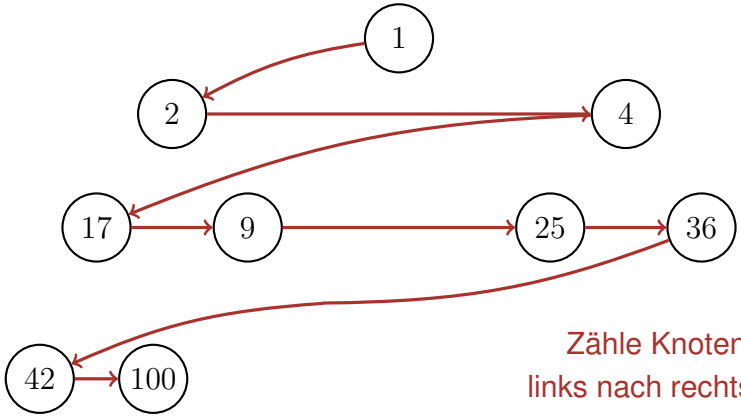
- 2 ist Vorgänger von 17 und 9
- 17 ist linkes Kind von 2
- 9 ist rechtes Kind von 2

Heaps



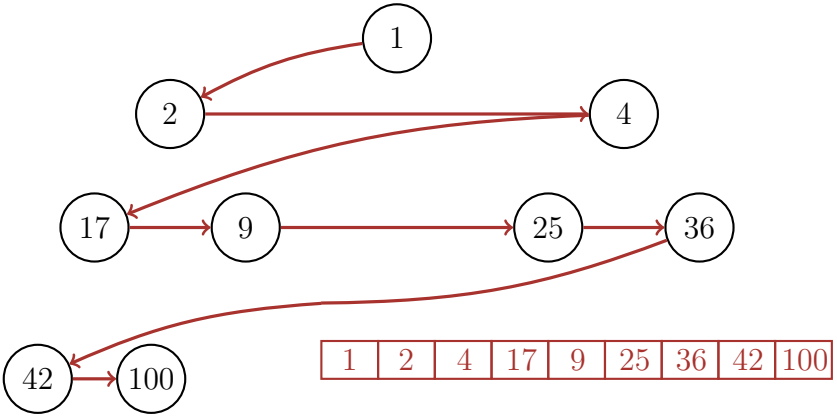
Kinder haben immer grössere Werte als ihre Vorgänger

Heaps

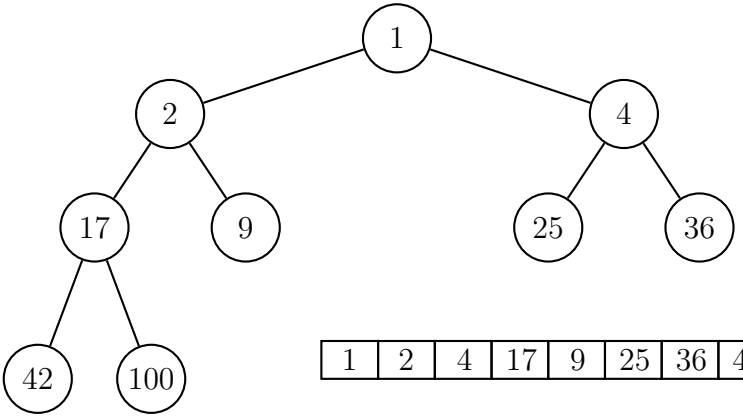


Zähle Knoten des Baums von
links nach rechts, Level für Level
durch und schreibe die Werte so in eine Liste

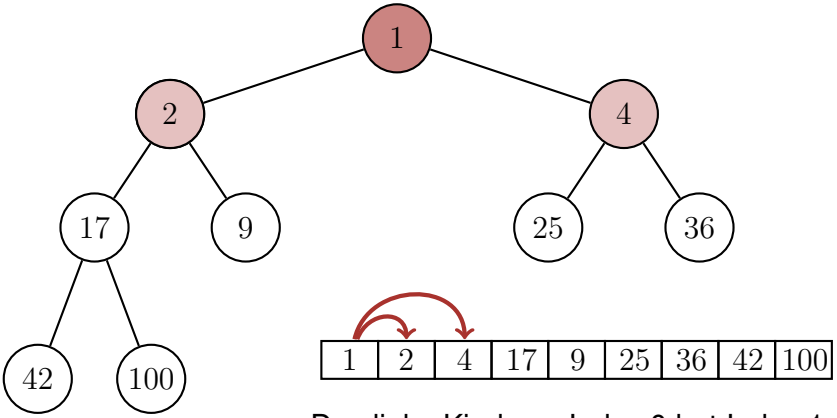
Heaps



Heaps

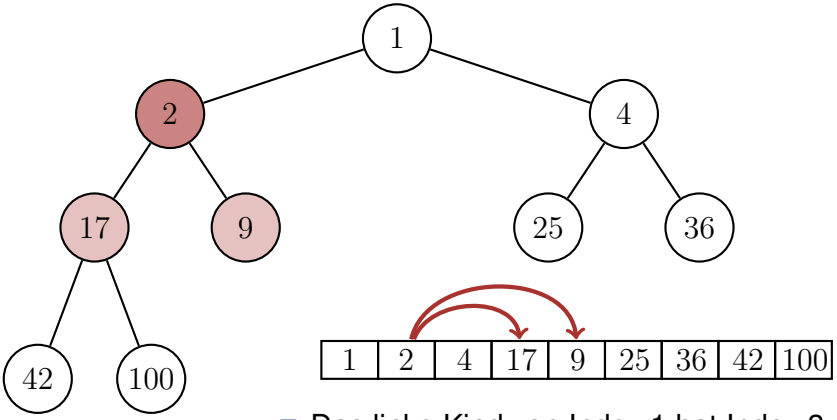


Heaps



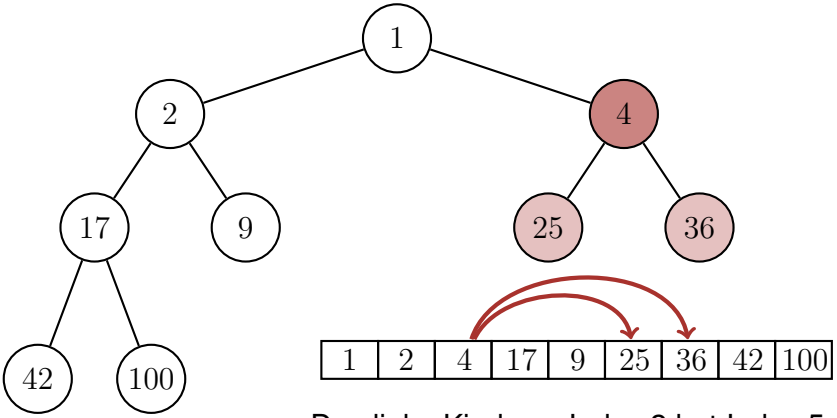
- Das linke Kind von Index 0 hat Index 1
- Das rechte Kind von Index 0 hat Index 2

Heaps



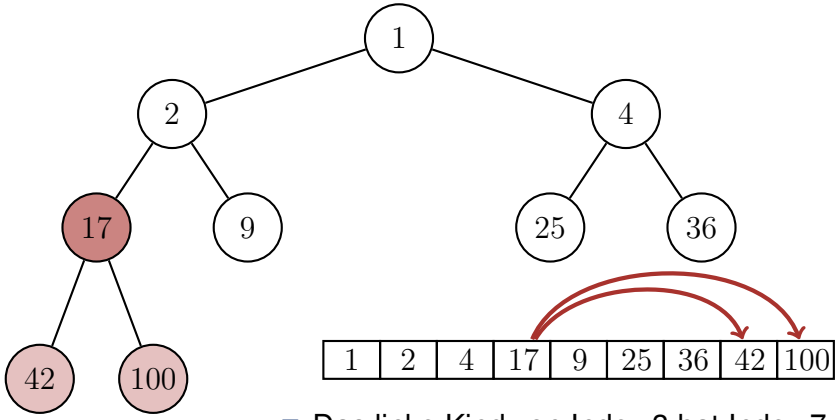
- Das linke Kind von Index 1 hat Index 3
- Das rechte Kind von Index 1 hat Index 4

Heaps



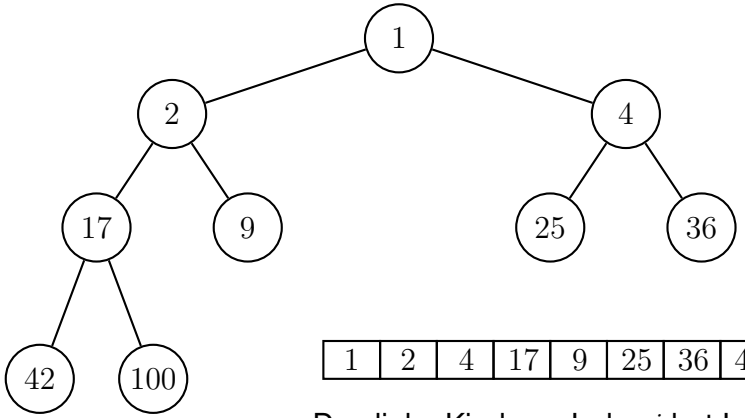
- Das linke Kind von Index 2 hat Index 5
- Das rechte Kind von Index 2 hat Index 6

Heaps



- Das linke Kind von Index 3 hat Index 7
- Das rechte Kind von Index 3 hat Index 8

Heaps



1	2	4	17	9	25	36	42	100

- Das linke Kind von Index i hat Index $2i + 1$
- Das rechte Kind von Index i hat Index $2i + 2$

Heaps

Erstelle Klasse `Heap` mit Funktionen

- `add(self, x)` — Element einfügen in $\mathcal{O}(\log n)$
- `getmin(self)` — Minimum ausgeben in $\mathcal{O}(1)$
- `popmin(self)` — Minimum entfernen in $\mathcal{O}(\log n)$

Heaps

Erstelle Klasse `Heap` mit Funktionen

- `add(self, x)` — Element einfügen in $\mathcal{O}(\log n)$
- `getmin(self)` — Minimum ausgeben in $\mathcal{O}(1)$
- `popmin(self)` — Minimum entfernen in $\mathcal{O}(\log n)$

```
class Heap:
  ...
  def add(self, x):
    ...
  def getmin(self):
    ...
  def popmin(self):
    ...
```

Heaps – Initialisierung

- Konstruktor erstellt Liste

```
def __init__(self):
    self.data = []
```

Heaps – Initialisierung

- Konstruktor erstellt Liste

```
def __init__(self):
    self.data = []
```

- Erstelle Hilfsfunktionen; der Unterstrich am Anfang sagt aus, dass diese nur für den „internen Gebrauch“ gedacht sind

```
def _swap(self, i, j):
    self.data[i], self.data[j] = self.data[j], self.data[i]
```

```
def _parent(self, i):
    return (i-1) // 2
```

```
def _left_child(self, i):
    return 2 * i + 1
```

```
def _right_child(self, i):
    return 2 * i + 2
```

Heaps – Element einfügen

`add(self, x)` – Füge Element `x` ein

Heaps – Element einfügen

`add(self, x)` – Füge Element `x` ein

- Füge `x` hinten an
- Schaue nun auf die letzte Stelle des Heaps
- Ist das Element kleiner als sein Vorgänger, vertausche beide
- Schaue nun an die Stelle des Vorgängers und wiederhole

Heaps – Element einfügen

`add(self, x)` – Füge Element `x` ein

- Füge `x` hinten an
- Schaue nun auf die letzte Stelle des Heaps
- Ist das Element kleiner als sein Vorgänger, vertausche beide
- Schaue nun an die Stelle des Vorgängers und wiederhole

`getmin(self)` – Gib kleinsten Element zurück

Heaps – Element einfügen

`add(self, x)` – Füge Element `x` ein

- Füge `x` hinten an
- Schaue nun auf die letzte Stelle des Heaps
- Ist das Element kleiner als sein Vorgänger, vertausche beide
- Schaue nun an die Stelle des Vorgängers und wiederhole

`getmin(self)` – Gib kleinsten Element zurück

- Gib das erste Element der Liste `data` zurück

Heaps – Element einfügen

```
def add(self, x):
    self.data.append(x)
    a = len(self.data) - 1
    while a > 0 and self.data[a] < self.data[self._parent(a)]:
        self._swap(a, self._parent(a))
        a = self._parent(a)
```

Heaps – Element einfügen

```
def add(self, x):
    self.data.append(x)
    a = len(self.data) - 1
    while a > 0 and self.data[a] < self.data[self._parent(a)]:
        self._swap(a, self._parent(a))
        a = self._parent(a)
```

```
def getmin(self):
    return self.data[0]
```

Heaps – Minimum entfernen

`pop_min(self)` – Entferne kleinstes Element

Heaps – Minimum entfernen

`pop_min(self)` – Entferne kleinstes Element

- Das Element steht an der Wurzel, also an der ersten Stelle des Heaps
- Nicht einfach entfernen und den Rest unverändert lassen

Heaps – Minimum entfernen

`pop_min(self)` – Entferne kleinstes Element

- Das Element steht an der Wurzel, also an der ersten Stelle des Heaps
- Nicht einfach entfernen und den Rest unverändert lassen

- Überschreibe erstes Element mit letztem und lösche dieses mit `data.pop()`
- Nun steht ein falsches Element in der Wurzel
- Ordne den Baum neu von oben nach unten
- Tausche dazu Wurzel mit kleinerem Kind
- Fahre mit diesem Kind fort und wiederhole

Heaps – Minimum entfernen

```
def popmin(self):
    self.data[0] = self.data[-1]
    self.data.pop()
    a = 0
    while True:
        m = a
        if self._left_child(a) < len(self.data) and \
            self.data[self._left_child(a)] < self.data[m]:
            m = self._left_child(a)
        if self._right_child(a) < len(self.data) and \
            self.data[self._right_child(a)] < self.data[m]:
            m = self._right_child(a)
        if m > a:
            self._swap(a, m)
            a = m
        else:
            return
```

Heapsort

Sortieren mit Heaps

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level
- `add()` betrachtet nur einen Knoten pro Level

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level
- `add()` betrachtet nur einen Knoten pro Level
- `pop_min()` betrachtet nur zwei Knoten pro Level

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level
- `add()` betrachtet nur einen Knoten pro Level
- `pop_min()` betrachtet nur zwei Knoten pro Level
- Beide Funktionen haben Komplexität in $\mathcal{O}(\log n)$

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level
- `add()` betrachtet nur einen Knoten pro Level
- `pop_min()` betrachtet nur zwei Knoten pro Level
- Beide Funktionen haben Komplexität in $\mathcal{O}(\log n)$

- Damit werden n Elemente eingefügt in $\mathcal{O}(n \log n)$
- Danach kann das jeweilige Minimum n Mal extrahiert werden in $\mathcal{O}(n \log n)$

Heaps – Komplexität

- Seien n Elemente im Heap
- Dann besitzt der Heap ungefähr $\log n$ Level
- `add()` betrachtet nur einen Knoten pro Level
- `pop_min()` betrachtet nur zwei Knoten pro Level
- Beide Funktionen haben Komplexität in $\mathcal{O}(\log n)$

- Damit werden n Elemente eingefügt in $\mathcal{O}(n \log n)$
- Danach kann das jeweilige Minimum n Mal extrahiert werden in $\mathcal{O}(n \log n)$
- **Heapsort:** Mit dieser Strategie können wir in $\mathcal{O}(n \log n)$ sortieren

Heapsort

```
def heapsort(data):
    tmp = Heap()
    sorted_data = []
    for element in data:
        tmp.add(element)
    for i in range(len(data)):
        sorted_data.append(tmp.getmin())
        tmp.popmin()
    return sorted_data
```

Danke für die Aufmerksamkeit