ETH zürich

Departement Informatik

Programming and Problem-Solving

Object Orientation

Dennis Komm

Spring 2021 – May 20, 2021

Classes and Objects

Pyhon Classes

Classes – Technical

A class is an entity with a **name** that contains **data** and **functionality**

Classes – Technical

A class is an entity with a **name** that contains **data** and **functionality**

- A class defines a new **data type**
- *Data:* stored variables, called **attributes**
- *Functionality:* consists of functions, called **methods**
- Classes are (often) separate `.py` files with the same name

Classes – Technical

A class is an entity with a **name** that contains **data** and **functionality**

- A class defines a new **data type**
- *Data:* stored variables, called **attributes**
- *Functionality:* consists of functions, called **methods**
- Classes are (often) separate `.py` files with the same name

Name

- attribute1
- attribute2
- ...

- method1
- method2
- ...

Classes – Conceptual

Classes facilitate to **bundle** the data that **belongs together** contentwise

Classes – Conceptual

Classes facilitate to **bundle** the data that **belongs together** contentwise

Classes provide **functionality** that allows to perform **queries** based on the data or **operations** on the data

Classes – Conceptual

Classes facilitate to **bundle** the data that **belongs together** contentwise

Classes provide **functionality** that allows to perform **queries** based on the data or **operations** on the data

Example

- Coherent measurements
- Functions to read out data
- Functions to modify data

Example – Earthquake Catalog

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			

...

Example – Earthquake Catalog

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			

...

Class for Measurement – First Try

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

Class for Measurement – First Try

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 Class `Measurement`

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

    latitude = 0
    longitude = 0

    magnitude = 0
```

Class for Measurement – First Try

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 Class `Measurement`

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

    latitude = 0
    longitude = 0

    magnitude = 0
```

Name of the class / data type

Class for Measurement – First Try

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 Class Measurement

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

    latitude = 0
    longitude = 0

    magnitude = 0
```

Attributes according to CSV header

Class for Measurement – First Try

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 Class `Measurement`

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

    latitude = 0
    longitude = 0

    magnitude = 0
```

Measurement

- `date` (Empty string λ)
- `time` (Empty string λ)
- `latitude` (Number 0)
- `longitude` (Number 0)
- `magnitude` (Number 0)

Classes and Objects

Python Objects

Objects – Instances of Classes

Classes describe the structure of objects, like a **blueprint**
⇨ Comparable with the **header** of the CSV file

Objects – Instances of Classes

Classes describe the structure of objects, like a **blueprint**
⇨ Comparable with the **header** of the CSV file

Objects are instantiated according to the blueprint and will contain values
⇨ Comparable with the individual **data rows** in the CSV file

Objects – Instances of Classes

Classes describe the structure of objects, like a **blueprint**
⇨ Comparable with the **header** of the CSV file

Objects are instantiated according to the blueprint and will contain values
⇨ Comparable with the individual **data rows** in the CSV file

Example

- Variables to store parameters of measurement
- Function to display measurements lucidly
- Function to compare measurements

Object Instantiation

Objects are instances of classes

Object Instantiation

Objects are instances of classes

```
w = Measurement()
```

Object Instantiation

Objects are instances of classes

```
w = Measurement()
```

Instantiation of an object of type “Measurement”

Object Instantiation

Objects are instances of classes

```
w = Measurement()
```

Instantiation of an object of type “Measurement”

Measurement w

date	λ
time	λ
latitude	0
longitude	0
magnitude	0

Object Instantiation

Objects are instances of classes

```
w = Measurement()

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

Measurement w

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

Object Instantiation

Objects are instances of classes

```
w = Measurement()

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

Dot notation; `instance.attribute`

Measurement `w`

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

Object Instantiation

Objects are instances of classes

```
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

Object Instantiation

Objects are instances of classes

```
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

Object Instantiation

Objects are instances of classes

```
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

Object Instantiation

Objects are instances of classes

```
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

Class for Measurement – Second Try

Measurement

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

Class for Measurement – Second Try

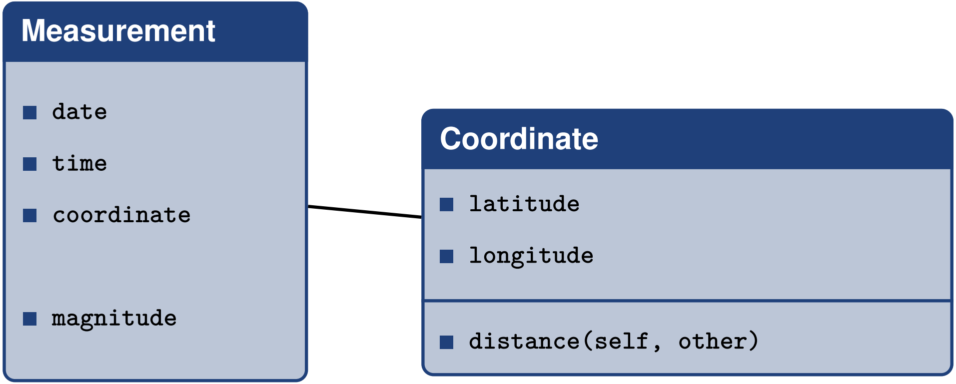
Measurement

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

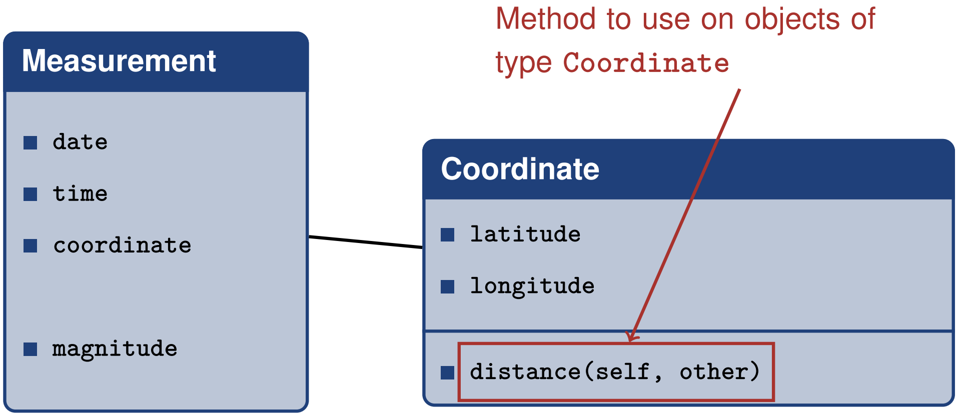
Better structuring

- Latitude and longitude belong in their own data type `Coordinate`
- Object of type `Measurement` has an attribute of type `Coordinate`
- "Composition"

Class for Measurement – Second Try



Class for Measurement – Second Try



Methods

- Methods are function that are defined within a class

Methods

- Methods are function that are defined within a class
- The first parameter is always `self`, which allows to refer to the current instance

Methods

- Methods are function that are defined within a class
- The first parameter is always `self`, which allows to refer to the current instance
- Again **dot notation**; Call analogously to `append()` for lists

Methods

- Methods are function that are defined within a class
- The first parameter is always `self`, which allows to refer to the current instance
- Again **dot notation**; Call analogously to `append()` for lists
- Pre-defined functions with special functionality

Methods

- Methods are function that are defined within a class
- The first parameter is always `self`, which allows to refer to the current instance
- Again **dot notation**; Call analogously to `append()` for lists
- Pre-defined functions with special functionality
- Function `__str__` defines what happens when instance is given to `print()`

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

Methods in Classes

```
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))
```

Methods in Classes

```python
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))
```

First parameter is always `self`

Methods in Classes

```
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))
```

Enables to access the current object from within a method of that class

Methods in Classes

```
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 formula

Classes and Objects

Constructors

Constructors

Creating a `Coordinate` needs three steps

Constructors

Creating a `Coordinate` needs three steps

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

Constructors

Creating a `Coordinate` needs three steps

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

Constructors facilitate to easily set the initial values of a newly created object

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

Constructors

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)              # Conversion from degree measure
        self.longitude = radians(deg_longitude)            # to radians measure

    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))
```

Constructors

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)           # Conversion from degree measure
        self.longitude = radians(deg_longitude)         # to radians measure

    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)))
```

Constructors

```
from math import *

class Coordinate:

    def __init__(self, deg_latitude, deg_longitude):
        self.latitude = radians(deg_latitude)           # Conversion from degree measure
        self.longitude = radians(deg_longitude)         # to radians measure

    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))
```

Is executed when object is initialized; parameter values are passed to this function

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

Managing an Earthquake Database

Managing an Earthquake Database

1. Implement data structure to represent earthquakes
2. Read in CSV file, create objects from the lines, insert them into a dictionary
3. Implement user interface to query data

Managing an Earthquake Database

1. Implement data structure to represent earthquakes
2. Read in CSV file, create objects from the lines, insert them into a dictionary
3. Implement user interface to query data

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

Managing an Earthquake Database

1. Implement data structure to represent earthquakes
2. Read in CSV file, create objects from the lines, insert them into a dictionary
3. Implement user interface to query data

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

Of interest are

- Index 0: Keys for dictionary; is converted to natural number
- Index 1: Date and time; is split at space
- Index 4: Longitude; is converted to floating-point number
- Index 5: Latitude; is converted to floating-point number
- Index 9: Magnitude on Richter scale; is converted to floating-point number

Managing an Earthquake Database

1. Implement data structure to represent earthquakes

Managing an Earthquake Database

1. Implement data structure to represent earthquakes

```
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)
```

Managing an Earthquake Database

2. Read in CSV, create objects from the lines, insert them into a dictionary

Managing an Earthquake Database

2. Read in CSV, create objects from the lines, insert them into a dictionary

```
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
```

Managing an Earthquake Database

3. Implement user interface to query data

Managing an Earthquake Database

3. Implement user interface to query data

```
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])
```

Managing a Student Database

Exercise – Managing a Student Database

Write a class the represents students with attributes

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

- Enable the user to create student objects using `input()`
- Save them into a dictionary
- Output every student using a `for ... in` loop

Exercise – Managing a Student Database

```python
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."
```

```python
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

Lists and Dictonaries

Complexity on lists and dictionaries with n elements

Lists

Access with `[]`	$\mathcal{O}(1)$
Insertion with `append()`	$\mathcal{O}(1)$
Insertion with `insert()`	$\mathcal{O}(n)$
Removal with `pop(0)`	$\mathcal{O}(1)$
Removal with `pop()`	$\mathcal{O}(1)$
Find minimum	$\mathcal{O}(n)$

Dictionaries

Access with `[]`	$\mathcal{O}(1)$
Insertion with `[]`	$\mathcal{O}(1)$
Find minimum	$\mathcal{O}(n)$

Heaps

Design data structure for special usage

⇨ Minimum can be computed efficiently (Using lists and dictionaries $\mathcal{O}(n)$)

Heaps

Design data structure for special usage
⇨ Minimum can be computed efficiently (Using lists and dictionaries $\mathcal{O}(n)$)

Data structure with the following operations

Insertion	$\mathcal{O}(\log n)$
Get minimum	$\mathcal{O}(1)$
Pop minimum	$\mathcal{O}(\log n)$

Heaps

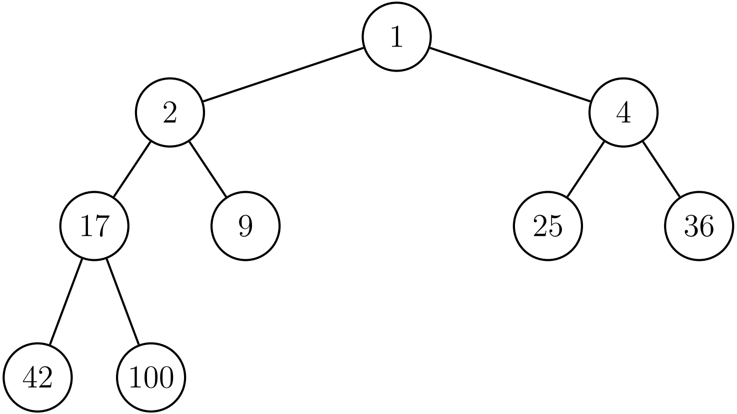
Design data structure for special usage
⇨ Minimum can be computed efficiently (Using lists and dictionaries $\mathcal{O}(n)$)

Data structure with the following operations

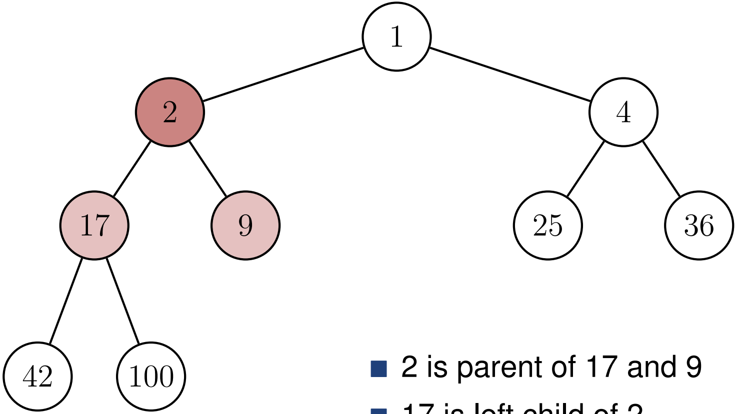
Insertion	$\mathcal{O}(\log n)$
Get minimum	$\mathcal{O}(1)$
Pop minimum	$\mathcal{O}(\log n)$

- Use a "tree"
- Embed this tree into list
- Root (first element of list) contains smallest element
- After removing an element, tree needs to be rearranged
- When inserting element, tree needs to be rearranged as well

Heaps

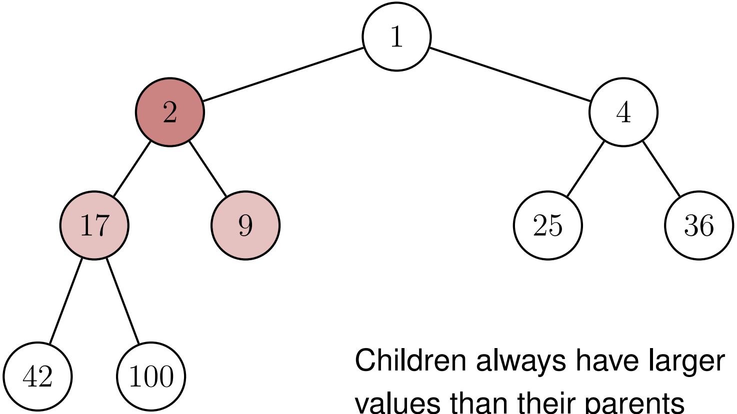


Heaps



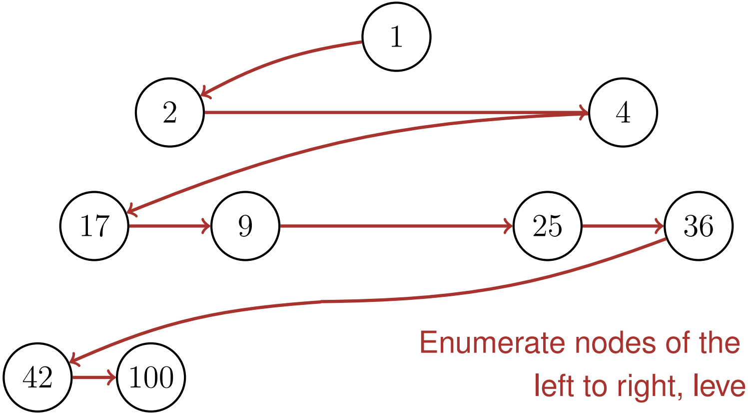
- 2 is parent of 17 and 9
- 17 is left child of 2
- 9 is right child of 2

Heaps



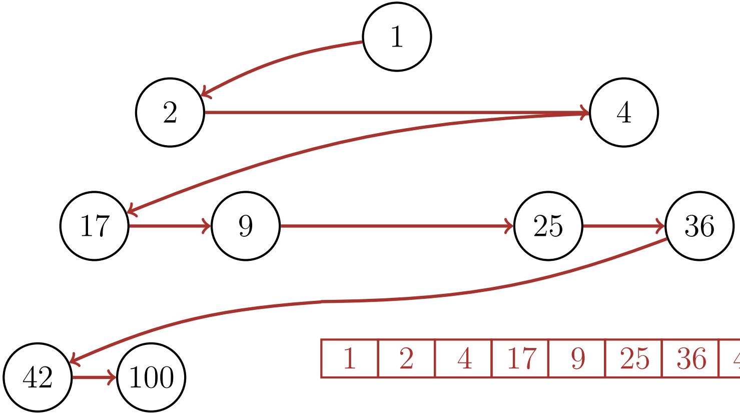
Children always have larger values than their parents

Heaps



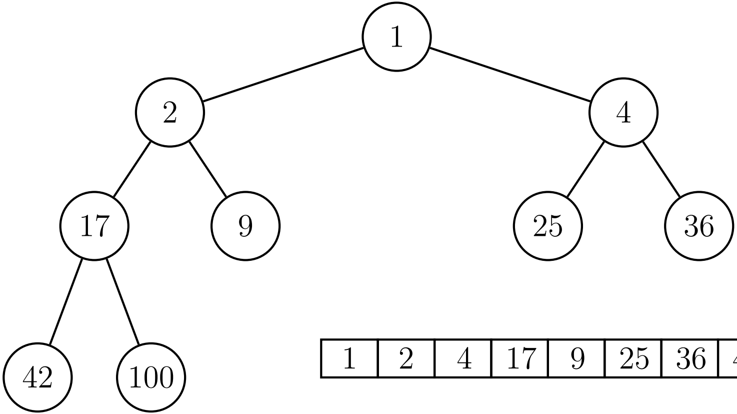
Enumerate nodes of the tree from left to right, level by level, and write values into a list in that order

Heaps



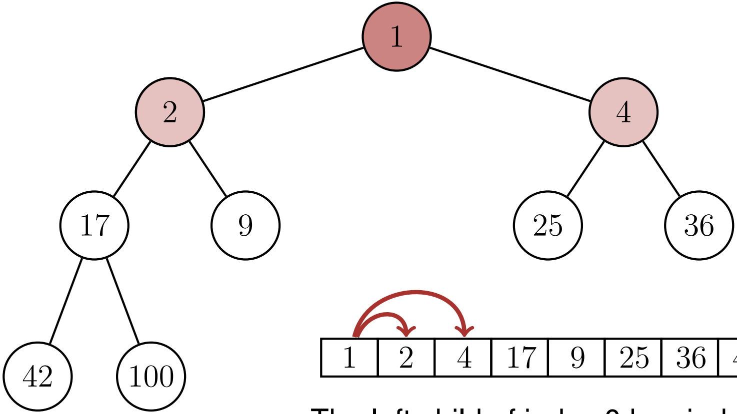
1	2	4	17	9	25	36	42	100

Heaps



1	2	4	17	9	25	36	42	100

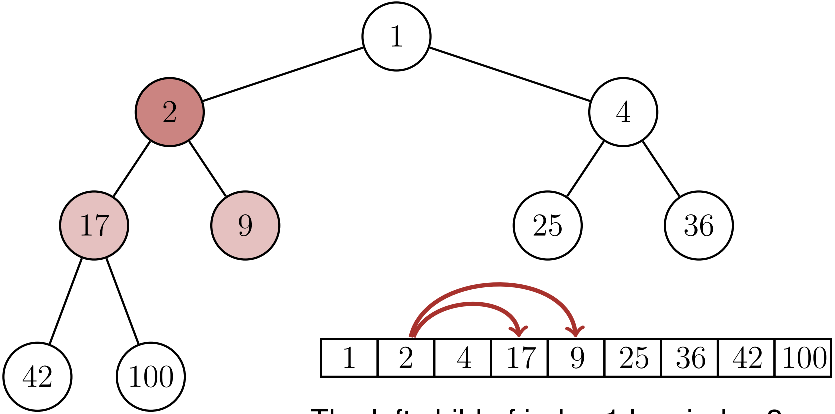
Heaps



1	2	4	17	9	25	36	42	100

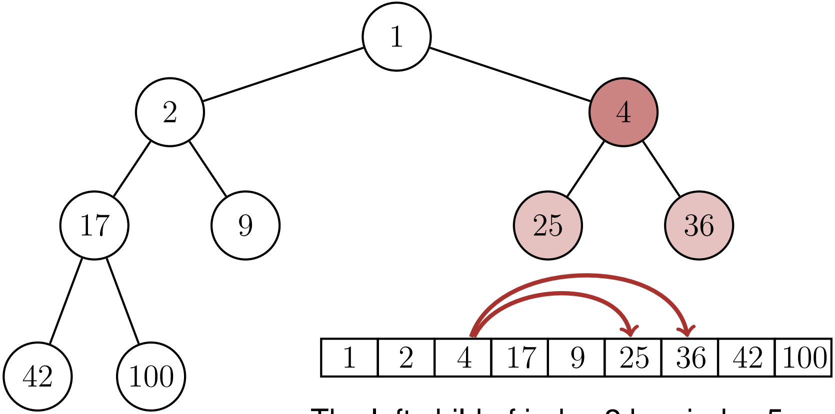
- The left child of index 0 has index 1
- The right child of index 0 has index 2

Heaps



- The left child of index 1 has index 3
- The right child of index 1 has index 4

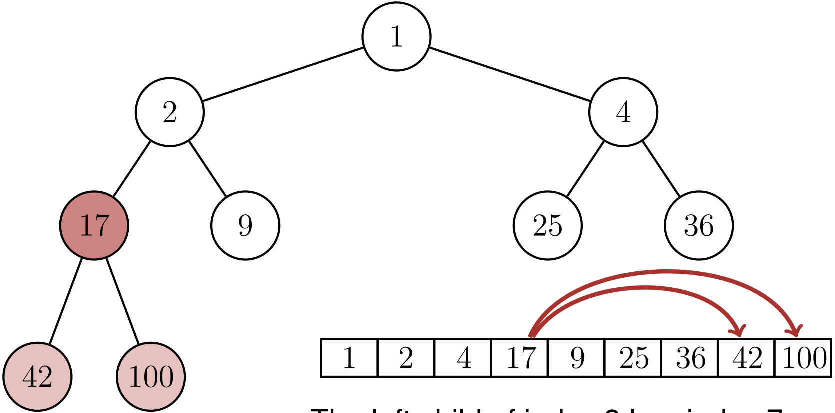
Heaps



1	2	4	17	9	25	36	42	100

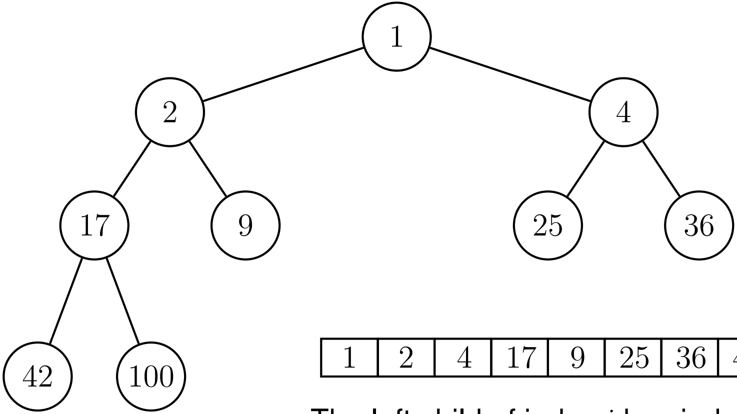
- The left child of index 2 has index 5
- The right child of index 2 has index 6

Heaps



- The left child of index 3 has index 7
- The right child of index 3 has index 8

Heaps



1	2	4	17	9	25	36	42	100

- The left child of index i has index $2i + 1$
- The right child of index i has index $2i + 2$

Heaps

Create class `Heap` with functions

- `add(self, x)` Insert element in $\mathcal{O}(\log n)$
- `getmin(self)` Output element in $\mathcal{O}(1)$
- `popmin(self)` Remove minimum in $\mathcal{O}(\log n)$

Heaps

Create class `Heap` with functions

- `add(self, x)` — Insert element in $\mathcal{O}(\log n)$
- `getmin(self)` — Output element in $\mathcal{O}(1)$
- `popmin(self)` — Remove minimum in $\mathcal{O}(\log n)$

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

Heaps – Initialization

- Constructor creates list

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

Heaps – Initialization

- Constructor creates list

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

- Create helper functions; the underscore at the beginning indicates that they are for “internal use” only

```
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 – Insert Element

`add(self, x)` – Insert element `x`

Heaps – Insert Element

`add(self, x)` – Insert element `x`

- Append `x` at the end
- Now consider last position of heap
- If this element is smaller than its parent, swap them
- Now consider position of parent and repeat

Heaps – Insert Element

`add(self, x)` – Insert element `x`

- Append `x` at the end
- Now consider last position of heap
- If this element is smaller than its parent, swap them
- Now consider position of parent and repeat

`getmin(self)` – Return smallest element

Heaps – Insert Element

`add(self, x)` – Insert element `x`

- Append `x` at the end
- Now consider last position of heap
- If this element is smaller than its parent, swap them
- Now consider position of parent and repeat

`getmin(self)` – Return smallest element

- Return the first element of list `data`

Heaps – Insert Element

```
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 – Insert Element

```
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 – Remove Minimum

`pop_min(self)` – Remove smallest element

Heaps – Remove Minimum

`pop_min(self)` – Remove smallest element

- The element is located at the root, that is, the first position of the heap
- We cannot simply remove it and leave the remainder

Heaps – Remove Minimum

`pop_min(self)` – Remove smallest element

- The element is located at the root, that is, the first position of the heap
- We cannot simply remove it and leave the remainder

- Overwrite first element with last element and remove the latter using `data.pop()`
- Now there is a wrong element located at the root
- Reorder tree from top to bottom
- To this end, swap root with larger child
- Now look at position of child and repeat

Heaps – Remove Minimum

```python
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

Sorting with Heaps

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$
- `add()` only considers one node per level

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$
- **`add()`** only considers one node per level
- **`pop_min()`** considers only two nodes per level

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$
- `add()` only considers one node per level
- `pop_min()` considers only two nodes per level
- Both functions have a complexity in $\mathcal{O}(\log n)$

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$
- **`add()`** only considers one node per level
- **`pop_min()`** considers only two nodes per level
- Both functions have a complexity in $\mathcal{O}(\log n)$

- With this, n elements can be inserted in $\mathcal{O}(n \log n)$
- Then, the respective minimum can be extracted n times in $\mathcal{O}(n \log n)$

Heaps – Complexity

- Suppose there are n elements in the heap
- Then the heap has roughly height $\log n$
- **`add()`** only considers one node per level
- **`pop_min()`** considers only two nodes per level
- Both functions have a complexity in $\mathcal{O}(\log n)$

- With this, n elements can be inserted in $\mathcal{O}(n \log n)$
- Then, the respective minimum can be extracted n times in $\mathcal{O}(n \log n)$
- **Heapsort:** With this strategy we can sort in $\mathcal{O}(n \log n)$

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
```

Thanks for your attention