Iterierbare Datenstrukturen

Felix Döring, Felix Wittwer, Anton Obersteiner

Python-Kurs

15. November 2021

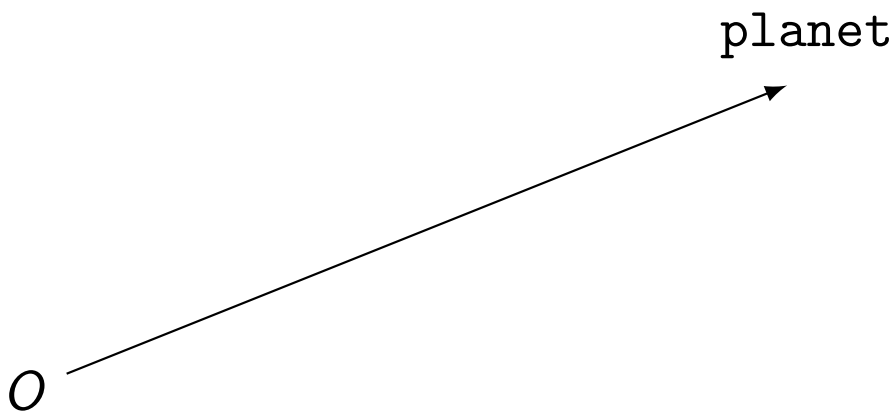
Gliederung

Objekte und Vererbung

- Was sind Klassen, was Objekte, was ist Vererbung?

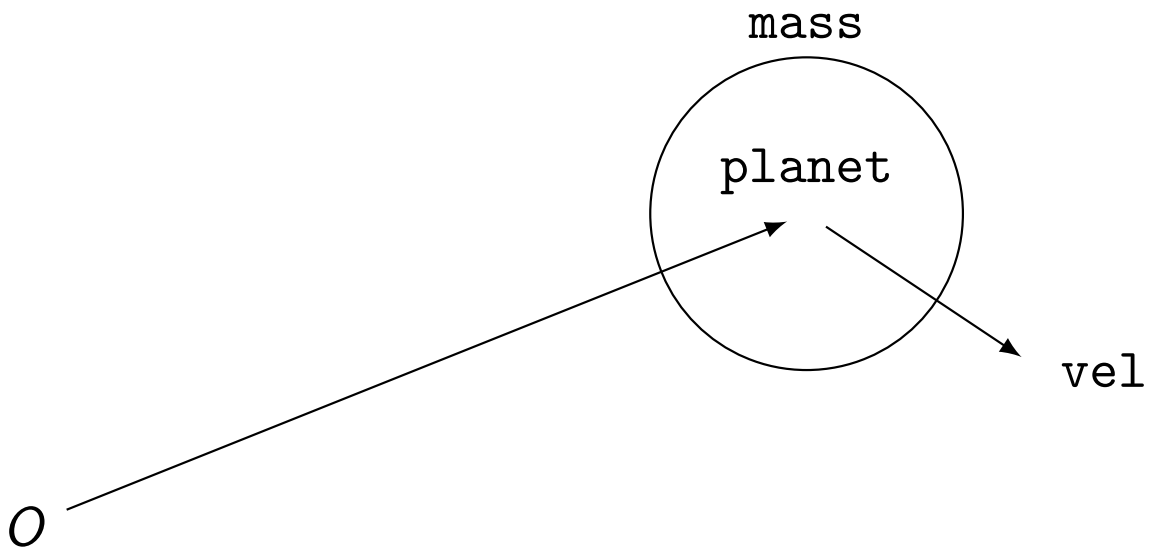
Objekte und Vererbung

- Was sind Klassen, was Objekte, was ist Vererbung?
- Was braucht unser Planet noch? `class Planet(Vector):`



Objekte und Vererbung

- Was sind Klassen, was Objekte, was ist Vererbung?
- Was braucht unser Planet noch? `class Planet(Vector):`



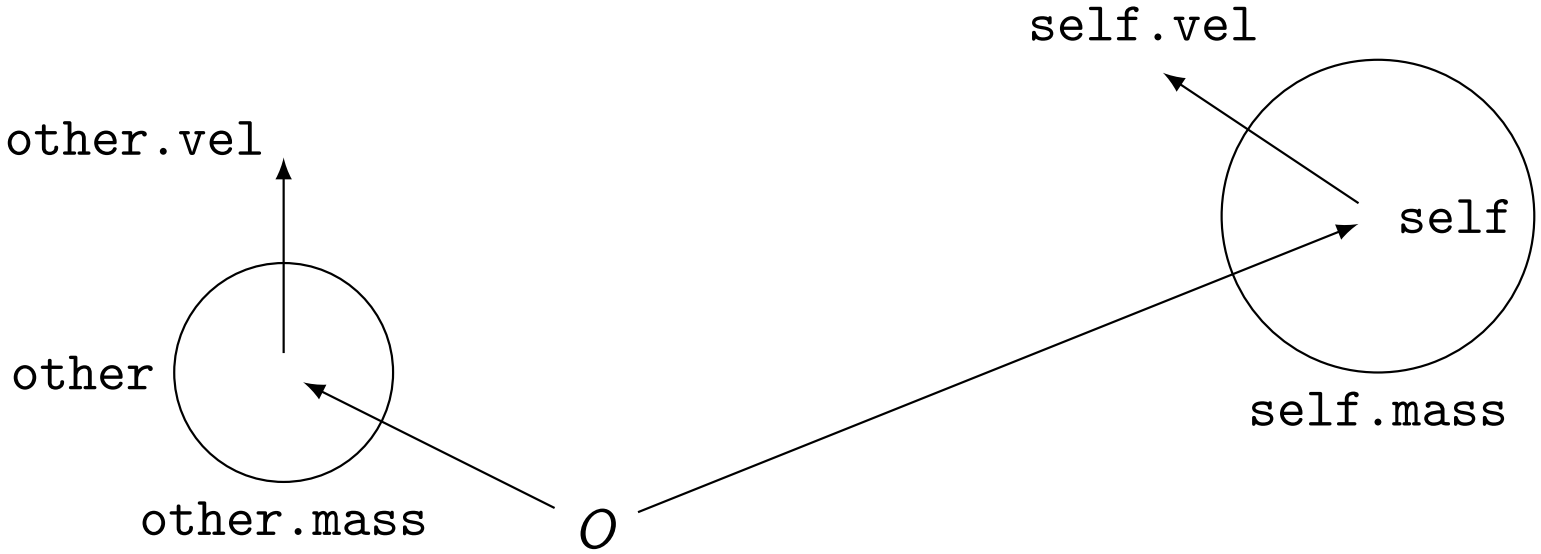
Planeten-Eigenschaften

```
dt = .1 #time step
class Planet(...):
    """
    Planet(mass:float, Vector(x, y)) -> a mass at a position
        vel: velocity (initially (0, 0))
    planet.update() -> move self by vel * dt
    planet.accel(force) -> accelerate (vel by force / mass)
    planet.attract(other) -> accelerate planet towards other
    """

    def __init__(self, mass, pos):
        super(Planet, self).__init__(pos.x, pos.y)
        ...
    def update(self): ...
    def accel(self, force): ...
    def attract(self, other): ...
```

github.com/AntonObersteiner/python-lessons/tree/master/latex/slides/resources/02_grundlagen/planet_question.py

Planeten-Eigenschaften



$$\dot{s} = v$$

$$\dot{v} = a$$

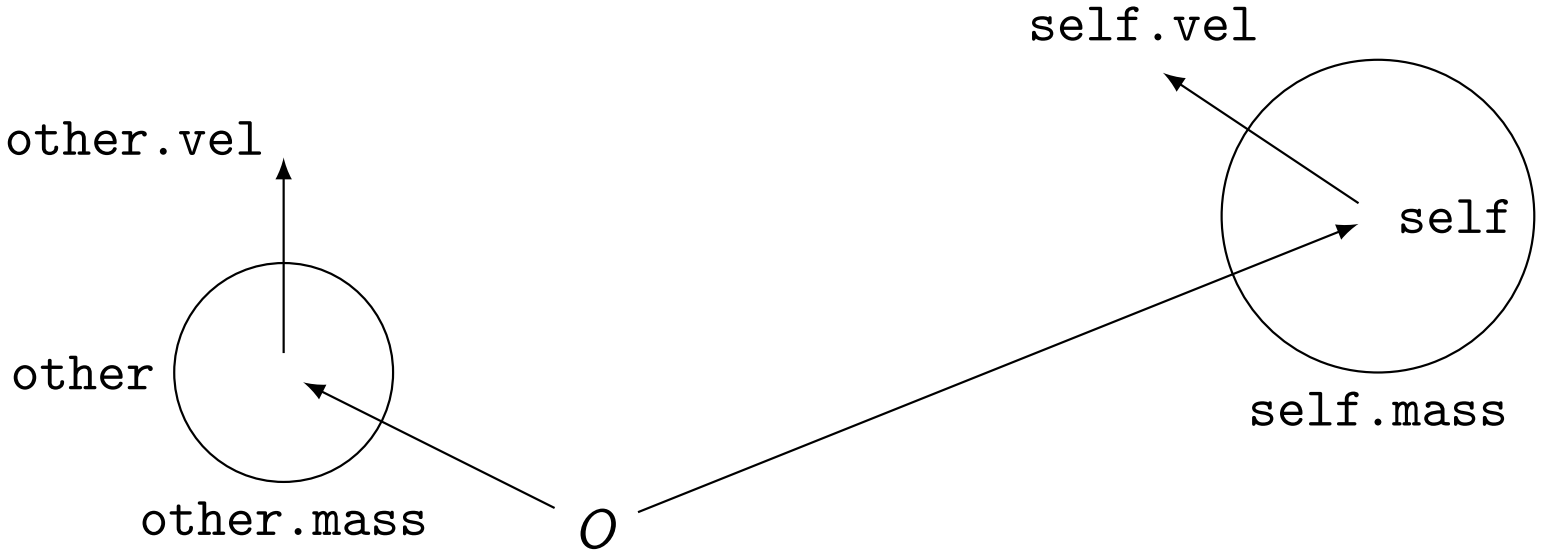
$$F = m \cdot a$$

$$d = s_2 - s_1$$

$$|F_G| =$$

$$F_G =$$

Planeten-Eigenschaften



$\dot{s} = v$ `self += vel * dt` Planet(Vector)

$\dot{v} = a$

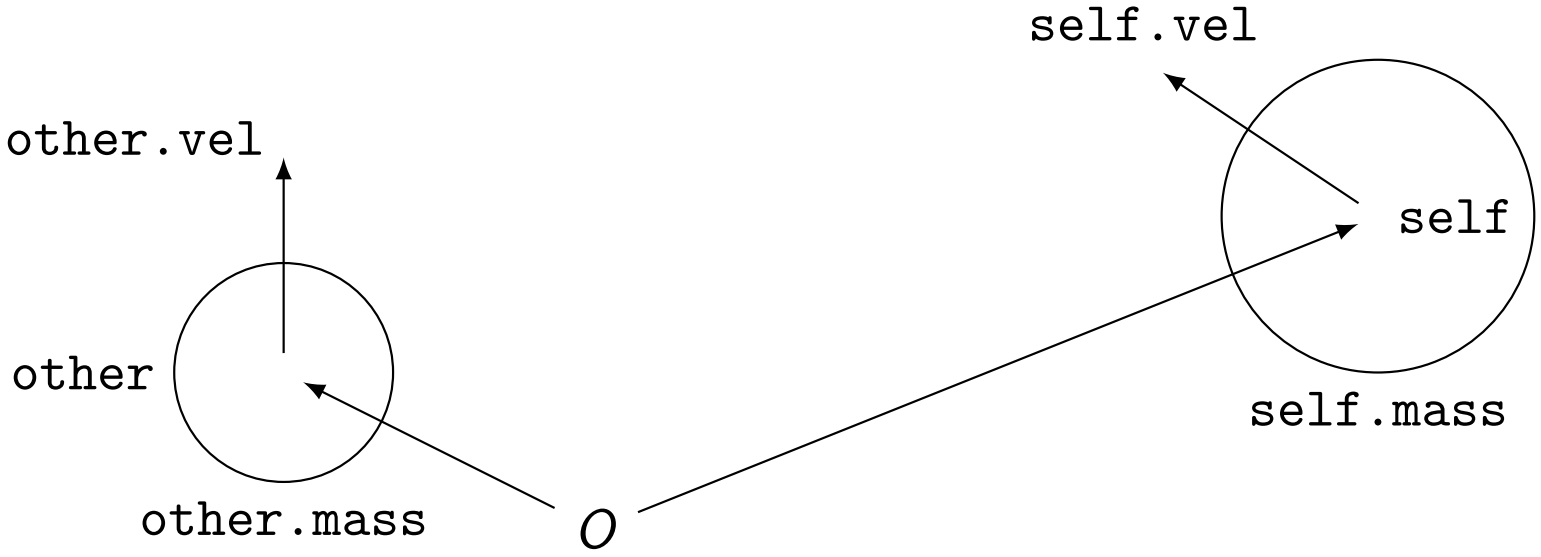
$F = m \cdot a$

$d = s_2 - s_1$

$|F_G| =$

$F_G =$

Planeten-Eigenschaften



$\dot{s} = v$ `self += vel * dt` Planet(Vector)

$\dot{v} = a$ `vel += acc * dt`

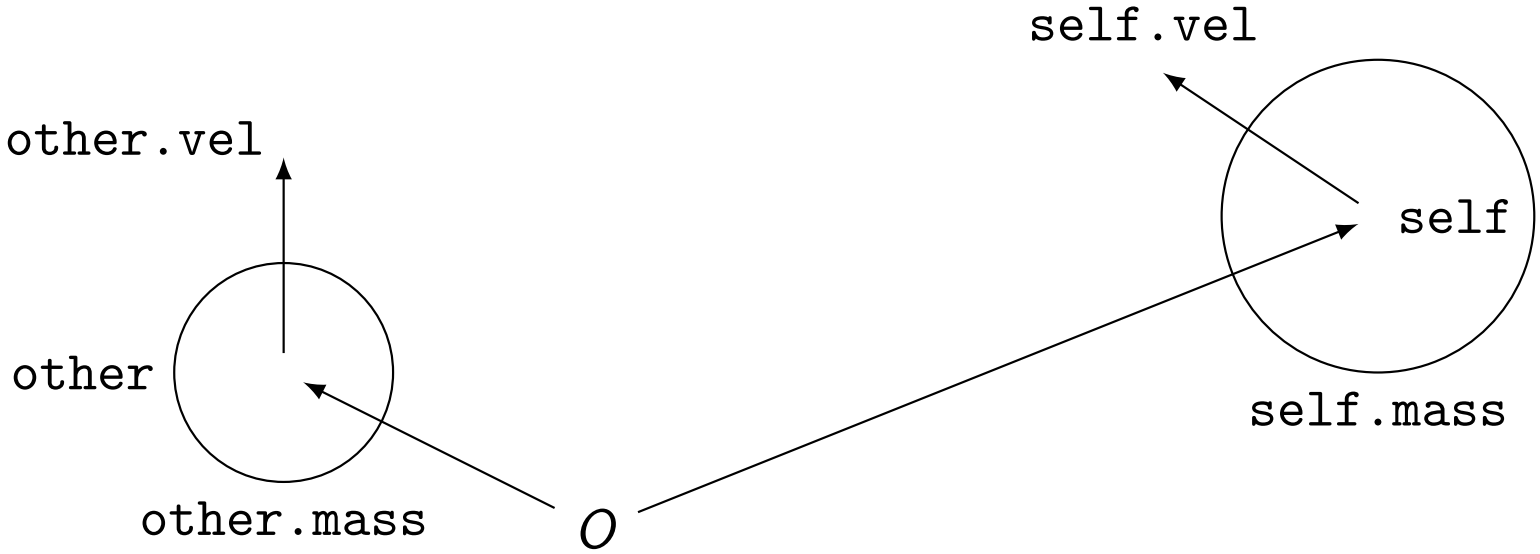
$F = m \cdot a$

$d = s_2 - s_1$

$|F_G| =$

$F_G =$

Planeten-Eigenschaften

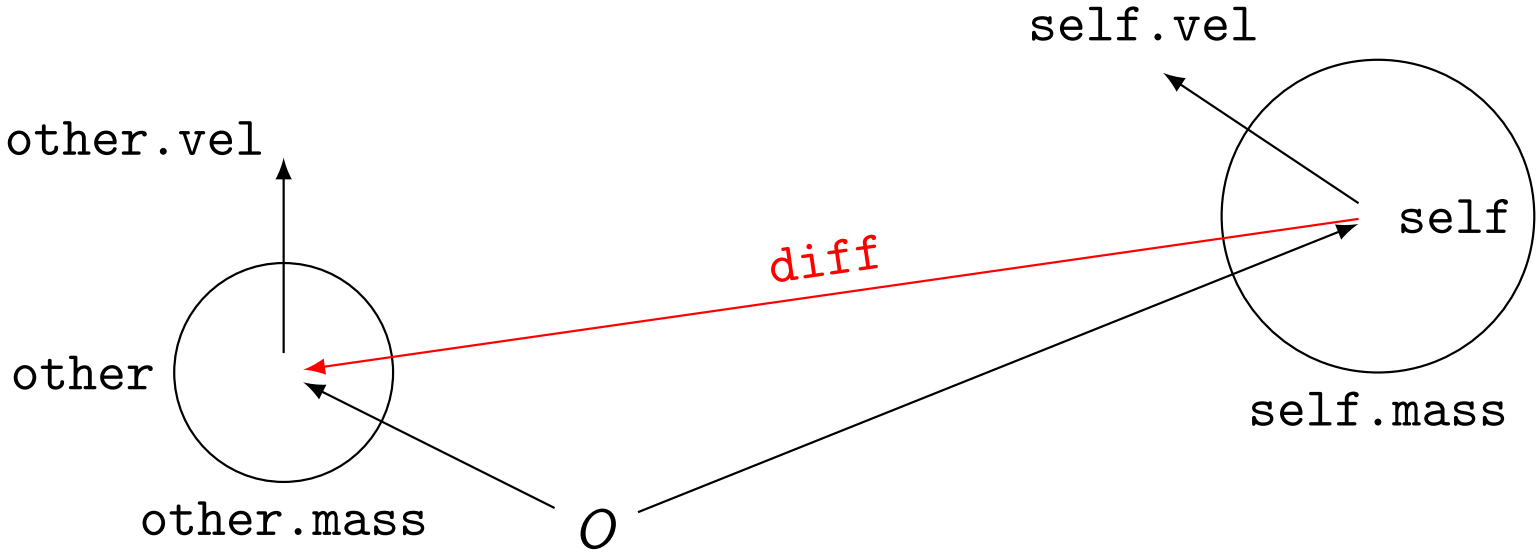


$\dot{s} = v$	`self += vel * dt`	Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`	
$F = m \cdot a$	`acc = force / mass`	
$d = s_2 - s_1$		

$|F_G| =$

$F_G =$

Planeten-Eigenschaften

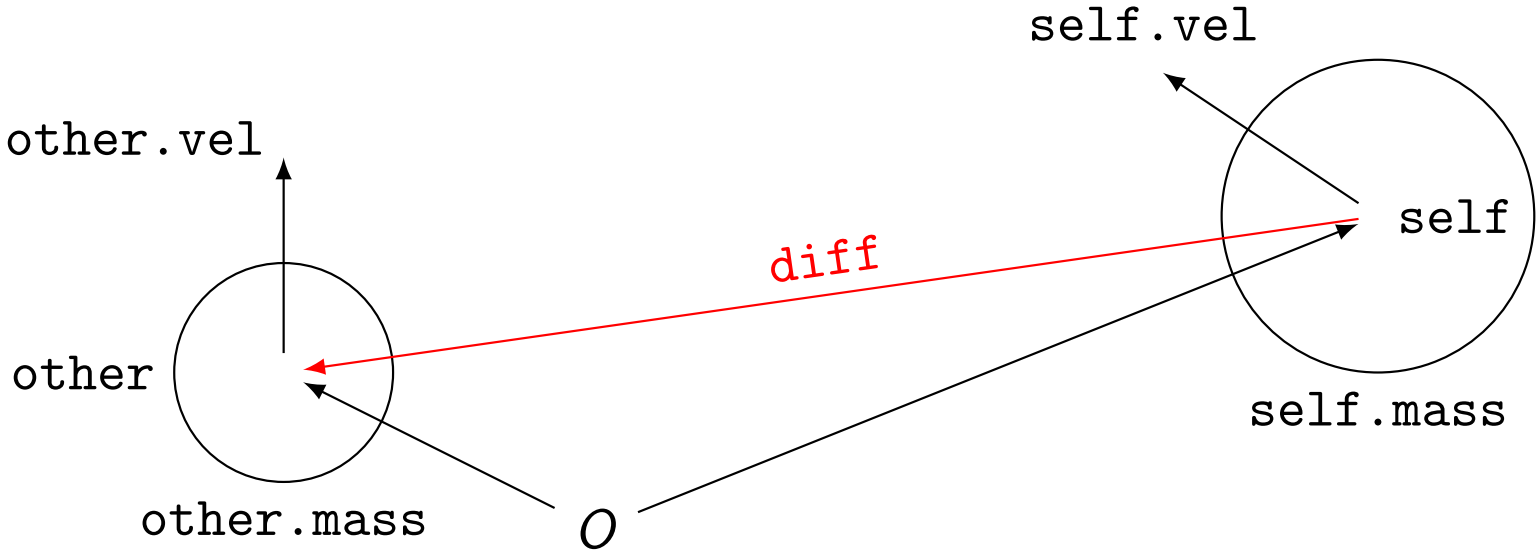


$\dot{s} = v$	`self += vel * dt` Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`
$F = m \cdot a$	`acc = force / mass`
$d = s_2 - s_1$	`diff = other - self`

$|F_G| =$

$F_G =$

Planeten-Eigenschaften

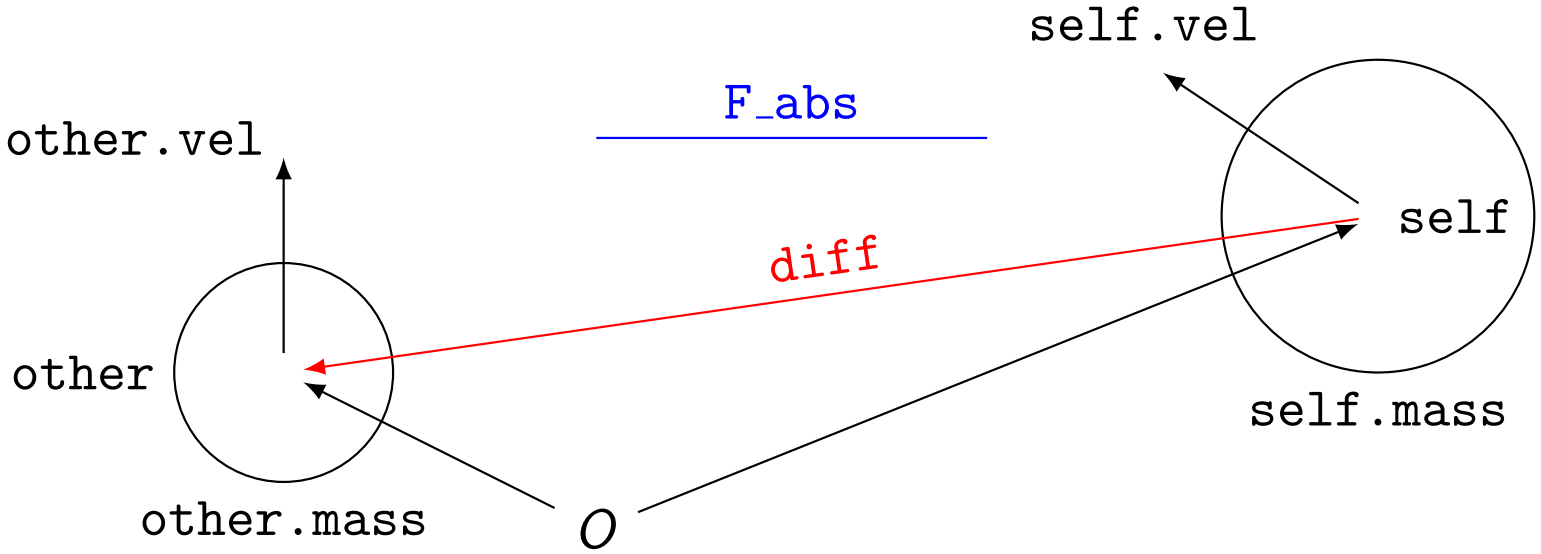


$\dot{s} = v$	`self += vel * dt` Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`
$F = m \cdot a$	`acc = force / mass`
$d = s_2 - s_1$	`diff = other - self`

$$|F_G| = G \cdot \frac{m_1 \cdot m_2}{|d|^2}$$

$$F_G =$$

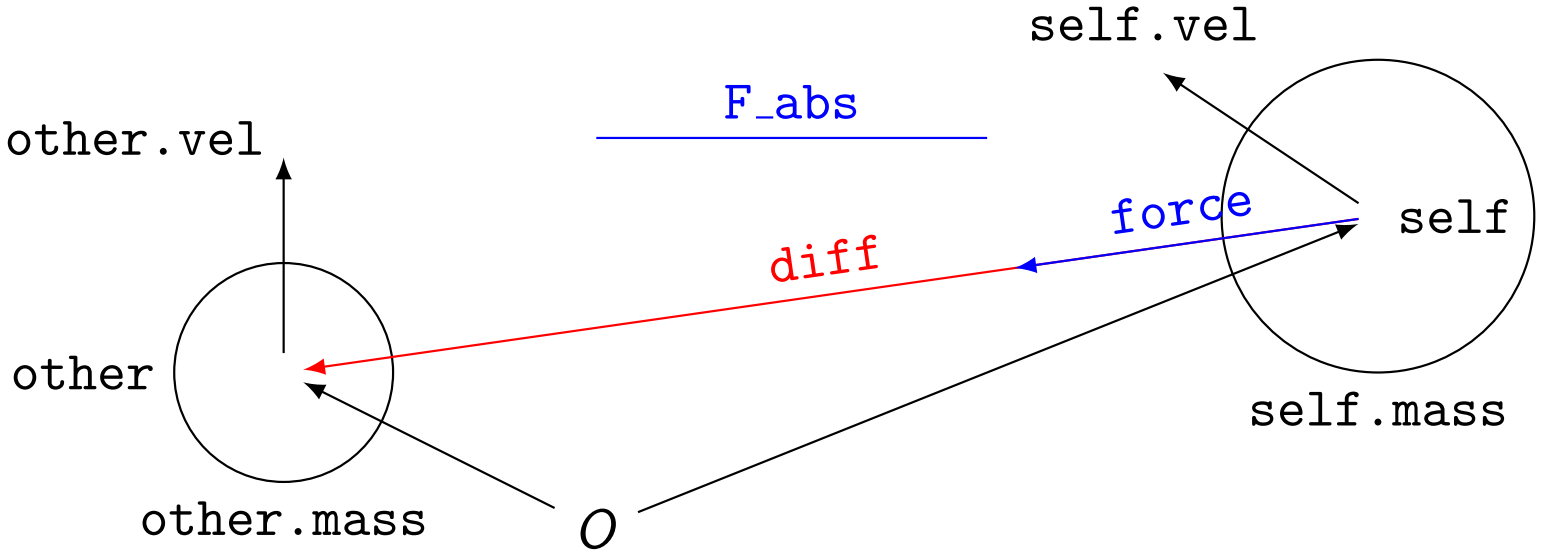
Planeten-Eigenschaften



$\dot{s} = v$	`self += vel * dt` Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`
$F = m \cdot a$	`acc = force / mass`
$d = s_2 - s_1$	`diff = other - self`
$\lvert F_G \rvert = G \cdot \frac{m_1 \cdot m_2}{\lvert d \rvert^2}$	`F_abs = G * s.mass * o.mass / dist ** 2`

$F_G =$

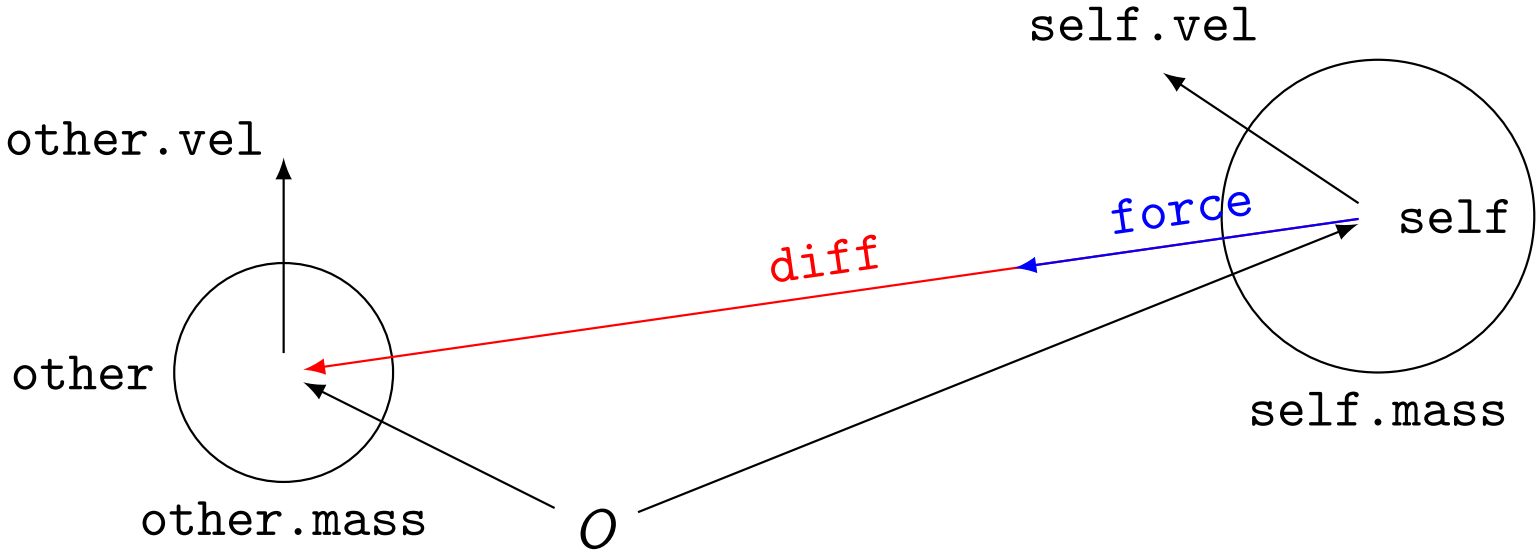
Planeten-Eigenschaften



$\dot{s} = v$	`self += vel * dt`	Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`	
$F = m \cdot a$	`acc = force / mass`	
$d = s_2 - s_1$	`diff = other - self`	
$\lvert F_G \rvert = G \cdot \frac{m_1 \cdot m_2}{\lvert d \rvert^2}$	`F_abs = G * s.mass * o.mass / dist ** 2`	

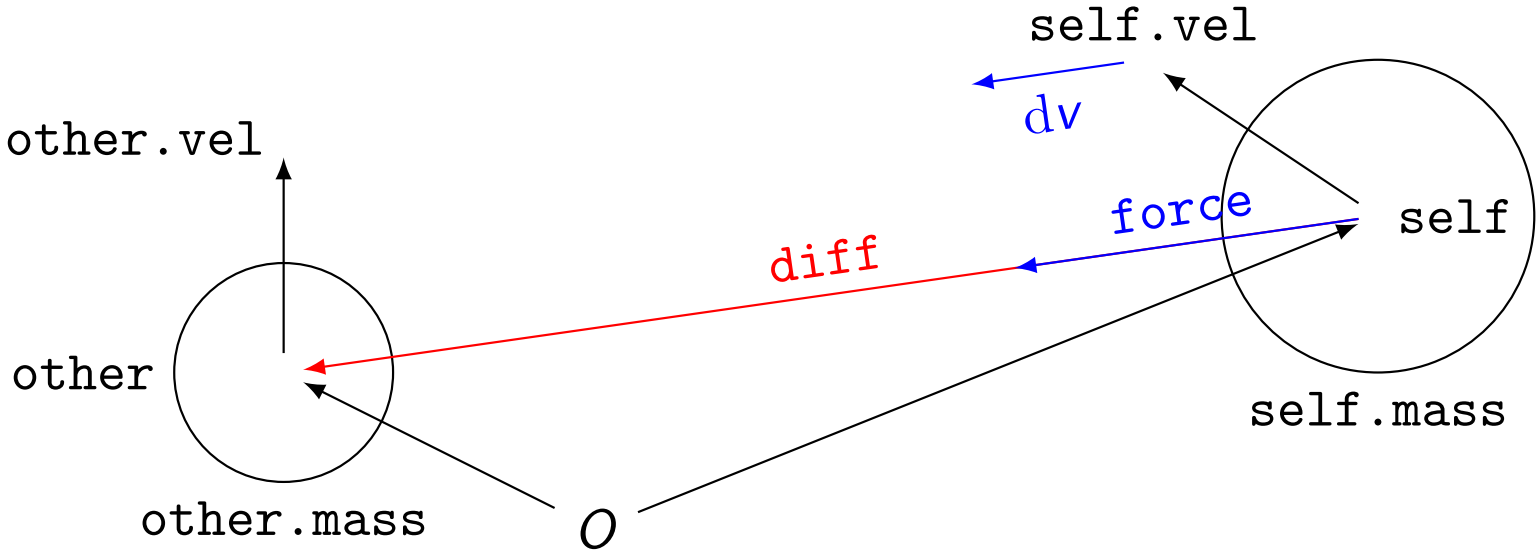
$$F_G = \frac{d}{|d|} \cdot |F_G|$$

Planeten-Eigenschaften



$\dot{s} = v$	`self += vel * dt` Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`
$F = m \cdot a$	`acc = force / mass`
$d = s_2 - s_1$	`diff = other - self`
$\lvert F_G \rvert = G \cdot \frac{m_1 \cdot m_2}{\lvert d \rvert^2}$	`F_abs = G * s.mass * o.mass / dist ** 2`
$F_G = \frac{d}{\lvert d \rvert} \cdot \lvert F_G \rvert$	`force = diff * (F_abs / diff.abs())`

Planeten-Eigenschaften



$\dot{s} = v$	`self += vel * dt` Planet(Vector)
$\dot{v} = a$	`vel += acc * dt`
$F = m \cdot a$	`acc = force / mass`
$d = s_2 - s_1$	`diff = other - self`
$\lvert F_G \rvert = G \cdot \frac{m_1 \cdot m_2}{\lvert d \rvert^2}$	`F_abs = G * s.mass * o.mass / dist ** 2`
$F_G = \frac{d}{\lvert d \rvert} \cdot \lvert F_G \rvert$	`force = diff * (F_abs / diff.abs())`

list - und Schleifen

```
1 students = ["Franz", "Anastasia", ...]
2 for student in students:
3     print(student)
```

- enthält variable Anzahl von Objekten mit fester Reihenfolge
- kann verschiedene Datentypen enthalten
- Auch Listen können in Listen gespeichert werden!
- Speicherverständnis!

list - und Schleifen

```
students = ["Franz", "Anastasia", ...]
for student in students:
    print(student)
```

- enthält variable Anzahl von Objekten mit fester Reihenfolge
- kann verschiedene Datentypen enthalten
- Auch Listen können in Listen gespeichert werden!
- Speicherverständnis!

```
def print_list_by_pop(data_list):
    while len(data_list) > 0:
        print(data_list.pop())

students = ["Franz", "Anastasia", ...]
print_list_by_pop(my_list)
print(my_list)
```

list - Unpacking-Beispiel

```
1 students = [("Anastasia", True), ("Janneke", False)]
2
3 for student, present in students:
4     if present:
5         print(student + " is here")
6     else:
7         print(student + " is absent")
```

- `tuple` enthalten mehrere Elemente,
- sind aber unveränderbar $\rightarrow$ Vor- und Nachteile

list - Unpacking-Beispiel

```
students = [("Anastasia", True), ("Janneke", False)]

for student, present in students:
    if present:
        print(student + " is here")
    else:
        print(student + " is absent")
```

- tuple enthalten mehrere Elemente,
- sind aber unveränderbar → Vor- und Nachteile

```
class DataManager:
    def get_name_and_position(self):
        return name, pos

if __name__ == '__main__':
    dm = DataManager(...)
    name, position = dm.get_name_and_position()
```

list - Anhängen und map

```
1 weird_list = [2, 6.1, "ein Text...", str]
2
3 squares = []
4 for i in [0, 1, 2, 3, 4]:
5     squares += [i**2]
6 print("Quadrate 0..4:", squares)
```

list - Anhängen und map

```
weird_list = [2, 6.1, "ein Text...", str]

squares = []
for i in [0, 1, 2, 3, 4]:
    squares += [i**2]
print("Quadrate 0..4:", squares)
```

```
print(
    "Wurzeln 0..4:",
    " >> ".join(map(
        lambda i: str(i ** .5),
        range(0, 5)
    ))
)
```

list - filter

Primzahlen mit `filter(func: data -> bool, data)` finden

list - filter

Primzahlen mit `filter(func: data -> bool, data)` finden

```
1  max_n = 1000
2
3  numbers = list(range(2, max_n))
4  for p in range(2, int(max_n ** .5)):
5      if p in numbers:
6          numbers = list(filter(
7              #welche n behalten?
8              lambda n:
9                  n == p  or
10                 n % p != 0,
11                 # % == mod
12             numbers
13         ))
14
15 print(numbers)
```

dict

Zuordnung: *key* $\rightarrow$ *value*

```
def main(args):
    settings = { "wait": 0.1 }
    def set_wait(arg_str):
        settings["wait"] = float(arg_str)

    arg_reader = {
        #argument key: (read, do-function)
        "--wait":    (1, set_wait),
        "--no-wait": (0, lambda: set_wait(0)),
    }
    # read, do = arg_reader["--wait"]
```

Es gibt Mengen in Python

- kann nur hashbare Einträge enthalten
- `students = { "Elia", "Muzaina", ... }`
- Fun fact: Python kann Unicode, aber mein LaTeX nicht

Iteration

- `for` *a* `in` *A*`:` *do*
- *a* wird hier *A* einmal durchlaufen,
- für jedes *a* wird der block *do* ausgeführt
- für Iterationen über Integer: `range([start], stop, step=1)`
- `__iter__` Methode wird intern aufgerufen

Iteration

- `for` *a* `in` *A*`:` *do*
- *a* wird hier *A* einmal durchlaufen,
- für jedes *a* wird der block *do* ausgeführt
- für Iterationen über Integer: `range([start], stop, step=1)`
- `__iter__` Methode wird intern aufgerufen
- Gegensatz: `while` mit Bedingung

File Handling

- Dateien können mit `open(filename, mode="r")` geöffnet werden
- *File Handler* sind Iteratoren über die Zeilen einer Datei
- **Wichtig:** File Handler müssen auch wieder geschlossen werden
- `r` steht für Lesezugriff, `w` für Schreibzugriff

Beachte: Wird eine Datei mit Schreibzugriff geöffnet, wird sie geleert! Also wichtige Inhalte vorher auslesen.

File Handling - Beispiel

```
with open(myfile, mode='r') as f:
    for line in f:
        # code


with open(myfile, mode='w+') as f:
    for line in document:
        f.write(line)
        # oder
        print(line, file=f)

f = open(myfile)

# code

f.close()
```

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`
- `from math import *` → `sin, cos, pi, exp, ...`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`
- `from math import *` → `sin, cos, pi, exp, ...`
- `from matplotlib import pyplot as plt`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`
- `from math import *` → `sin, cos, pi, exp, ...`
- `from matplotlib import pyplot as plt`
  → `plt.plot(X, Y); plt.show()`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`
- `from math import *` → `sin, cos, pi, exp, ...`
- `from matplotlib import pyplot as plt`
  → `plt.plot(X, Y); plt.show()`
- `from Planet import Planet`

Import

- Funktionen, Klassen, ... aus Bibliotheken importieren
- `import turtle` → `turtle.goto(100, 100)`
- `from math import sin` → `sin(angle)`
- `from math import *` → `sin, cos, pi, exp, ...`
- `from matplotlib import pyplot as plt`
  → `plt.plot(X, Y); plt.show()`
- `from Planet import Planet`
  → import `Planet` from file `Planet.py`