

# Würgeschlange 3 - Lesson 9

---

Tobias Maschek, Viktor Reusch

<https://github.com/jemx/wise1920-python>

mit Materialien von Felix Döring, Felix Wittwer <https://github.com/fsr/python-lessons>

Lizenz: CC BY 4.0 <https://creativecommons.org/licenses/by/4.0/>

7. Januar 2020

Python-Kurs

1. Special Method Names

2. `*args` und `**kwargs`

## Special Method Names

---

# Hey, i'm different!

```
1 my_list = ["a", "b", "c"]
2 print(len(my_list)) #3
3
4 #####
5
6 class AudioFile:
7
8     def __init__(self, interpret, name, length):
9         self.interpret = interpret
10        self.name = name
11        self.length = length
12
13
14 song = AudioFile("Darude", "Sandstorm", 232)
15
16 print(len(song)) # TypeError: object of type 'AudioFile' has no len()
```

# Hey, i'm different!

```
1 class AudioFile:
2
3     def __init__(self, interpret, name, length):
4         self.interpret = interpret
5         self.name = name
6         self.length = length
7
8     def __len__(self):
9         return self.length
10
11    def __str__(self):
12        return f"{self.name} by {self.interpret}"
13
14
15 song = AudioFile("Darude", "Sandstorm", 232)
16
17 print(len(song)) #232
18 print(song) #Sandstorm by Darude
```

# Übersicht über Special Method Names

Special Method Name	Benutzung	Erklärung
<code>__str__</code>	<code>str()</code>	informelle Darstellung
<code>__repr__</code>	<code>repr()</code>	erzeugender Python Ausdruck
<code>__bingo__</code>	<code>BINGO()</code>	erzeugt Gummibärchen
<code>__lt__</code> , <code>__ge__</code> , <code>__eq__</code> , etc.	<code>&lt;</code> , <code>&gt;=</code> , <code>==</code>	Vergleichsoperatoren
<code>__bool__</code>	<code>bool()</code> , <code>if</code> , <code>while</code>	boolsche Darstellung
<code>__len__</code>	<code>len()</code>	Größe des Container-Objekts
<code>__getitem__</code> , <code>__delitem__</code>	<code>myobj[key]</code> , <code>del myobj[key]</code>	ändern von Container-Elementen
<code>__contains__</code>	<code>in</code>	Element in Container enthalten
<code>__add__</code> , <code>__truediv__</code> , <code>__imul__</code>	<code>+</code> , <code>/</code> , <code>*=</code>	Zahlen nachbilden

[Dokumentation](#)

## Aufgabe 9-1

Programmiert eine Vektor-Klasse. Wir gehen grundsätzlich von dreidimensionalen Vektoren aus. Wird nur eine Koordinate übergeben, sind die anderen Koordinaten 0. Z. B.: `vect = Vector(7)`; entspricht dem Vektor  $(7 \ 0 \ 0)^T$

- `len(vect)` soll den Betrag des Vektors zurückgeben
- `bool(vect)` soll `False` zurückgeben, falls der Vektor der Nullvektor ist
- ein Vektor ist größer als ein anderer, falls er einen größeren Betrag hat
- Vektoren sollen mithilfe von `+` addiert werden können
- Sie sollen auch mithilfe von `*` (Skalarprodukt,  $\bullet$ ) multipliziert werden können

Nutzt die [Dokumentation!](#)

**\*args und \*\*kwargs**

---



# Problem

```
1 def greet(name0, name1 = None, name2 = None):  
2     print(f"Hello {name0}")  
3     if name1 is not None:  
4         print(f"Hello {name1}")  
5     if name2 is not None:  
6         print(f"Hello {name2}")  
7  
8 greet("Alice", "Bob")  
9 # Hello Alice  
10 # Hello Bob
```

Der \*- und der \*\*-Operator erlauben das Umwandeln von Argumenten von und zu Listen/Dicts.

- **\*args** im Funktionskopf packt alle übergebenen *benötigten Argumente* in eine *Liste*
- **\*args** im Funktionsaufruf entpackt die *Liste args* zu *Funktionsargumenten*
- **\*\*kwargs** im Funktionskopf packt alle übergebenen *optionalen Argumente* in ein *Dict*
- **\*\*kwargs** im Funktionsaufruf entpackt das *Dict kwargs* zu *optionalen Funktionsargumenten*

# Solution

```
1 def greet_at_least_one(name, *names):
2     print(f"Hello {name}")
3     for n in names:
4         print(f"Hello {n}")
5
6 def greet_all(*names):
7     for n in names:
8         print(f"Hello {n}")
9
10
11 greet_at_least_one("Alice", "Bob")
12 # Hello Alice
13 # Hello Bob
14 greet_all("Alice", "Bob")
15 # Hello Alice
16 # Hello Bob
```

# Beispiel

```
1 def print_squares(*numbers, **kwargs):  
2     squares = [number ** 2 for number in numbers]  
3     print(*squares, **kwargs)  
4  
5  
6 print_squares(1, 3, 4, sep=", ") # 1, 9, 16
```

## Aufgabe 9-2

Es soll eine Funktion zur Vergabe der Zeugnisnoten erstellt werden. Dabei kann ein Schüler beliebig viele Namen haben. Die Funktion soll

- die Namen als *\*args* entgegennehmen
- die Noten als *\*\*kwargs* entgegennehmen
- den Durchschnitt auf eine Kommastelle gerundet ausgeben.

und diese entsprechend des Beispiels (siehe unten) ausgeben.

```
1 print_marks("Rainer", "Maria", "Rilke", german=1, english=3)
```

```
1 Notenubersicht von Rainer Maria Rilke:  
2 german: 1  
3 english: 3  
4 Durchschnitt: 2.0
```