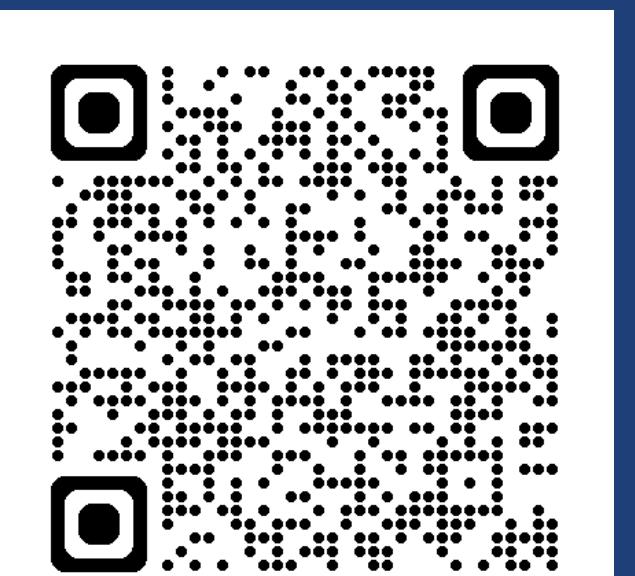




# Übungslektion 2

Informatik II, Nils Serck-Hanssen

18.02.2026 CHN D 29 16:15-18:00



Materialien: <https://github.com/Nilsser/Inf2>

# Willkommen

## Kurz zu mir:

- im 4. Semester MAVT
- erstes Mal TA
- Meine coding erfahrung
  - Mobile App Development
  - Maturitätsarbeit zur Erkennung von Hautkrebs mit AI ([viele Skills in inf2 sind dafür relevant!](#))
  - Bei aCe engagiert gewesen für ihr derzeitiges Fokusprojekt AISCO

# ...Letzte Woche

## Python Setup & jupyterLab

- Hoffe alle haben eine Lösung für sich gefunden
- **Falls nicht:**
  - gerne **auf mich zukommen**
  - oder: **Quick fix:** [colab.google.com](https://colab.google.com) , erfahrungsgemäss völlig ausreichend für inf2 ;)

# Feedback zu Hello World!

## nur eine Kleinigkeit, nicht prüfungsrelevant

- **Docstrings**, der String unter der Funktionsdefinition, dient einer kleinen Beschreibung der Funktion und ist in grossen Projekten sehr hilfreich.
- Der Code wird **unter** dem Docstring geschrieben, sonst würde Der Docstring nicht korrekt angezeigt werden.
- Wer sich dafür interessiert schönen, verständlichen Code zu schreiben (wichtig bei grossen Projekten): [pep8.org](http://pep8.org) (absolut optional)

# Letzte Vorlesung & Stoff für CodeExpert

## Variablen&Typen, Containers

**c++**

```
int      i = 1;  
double   d = 1.0;  
char     c = 'a';  
bool    b = true;
```

**Schema:**

DatenTyp variablen\_name = wert

**python**

```
i = 1;  
d = 1.0;  
c = 'a';  
b = true;
```

**Schema:**

variablen\_name = wert

Wir müssen in Python keine Datentypen angeben!

-  **Weshalb ist es nicht nötig?**
  - weil es gar nicht möglich ist: In Python ist alles\* ein Pointer!  
\* alle „Variablennamen“.
  - **Hilfe Pointer?!?**
    - **relax!** ...in Python ist es ganz natürlich damit zu arbeiten.
  - **merken für die Serie:** variable1=variable2 ist **kein Kopieren!** Die Pointer zeigen jetzt auf dasselbe Objekt!
    - **um eine Liste zu kopieren:** kopie= l.copy()
  - Nächste Woche mehr dazu...

# Trotzdem...

bezüglich Types kann man sich nicht alles erlauben

bsp: `print( 30 + "Sekunden" )`

```
print( 30 + "Sekunden" )
```

~~~~~^~~~~~

TypeError: unsupported operand type(s) for +: 'int' and 'str'

python  weiss nicht was die + Operation für string und int bedeuten

Lösung: `print( str(30) + "Sekunden" )`

sogenannte casting funktion **wandelt den int in einen String um!**

weitere: `float()`, `int()`

- Das Thema Typisierung wird zu einem späteren Zeitpunkt im Kurs wieder aufgegriffen.

# Containers

Eine Familie für alles was mehrere Elemente enthalten kann

Sequences (geordnet)

- tuple mein\_tupel = ("Auto", "Bus", "Bahn")
- list meine\_liste = ["Apfel", "Banane", "Kirsche"]
- range eine\_range= range(start, stop, step)
- string string = "string"

Collections (ungeordnet)

- set meine\_menge = {"Hund", "Katze", "Maus"}
- dictionary mein\_dict = {"Name": "Max", "Alter": 25}

**Sequences (ordered):** wir können über einen index auf die elemente zugreifen, es ist gewährleistet dass die Ordung von python aus erhalten bleibt.

# Operationen auf Containers

## Aufgabe: von c++ zu python

wie viele elemente hat unser container c ?

```
c.size();
```

beinhaltet c x?

```
std::find(c.begin(), c.end(), x);
```

jedes element in c ausgeben

```
for(int i=0;i<c.size();i++)
    std::cout << c[i] << "\n";
```

# Quiz Aufbauend dazu

## Findet die jeweiligen outputs

Für alle Fragen auf dieser Folie kann folgendes angenommen werden:

c = 

|   |      |   |     |      |
|---|------|---|-----|------|
| 1 | 3.14 | 7 | 'a' | True |
| 0 | 1    | 2 | 3   | 4    |

- len(c)
- 2 in c

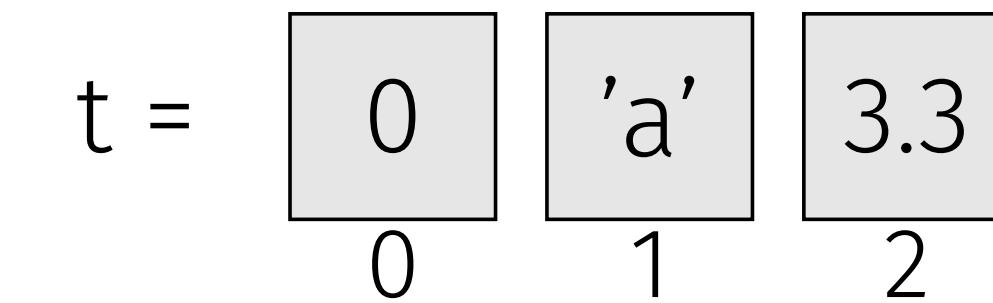
```
for x in c:  
    print(x)
```

-

# Sequences

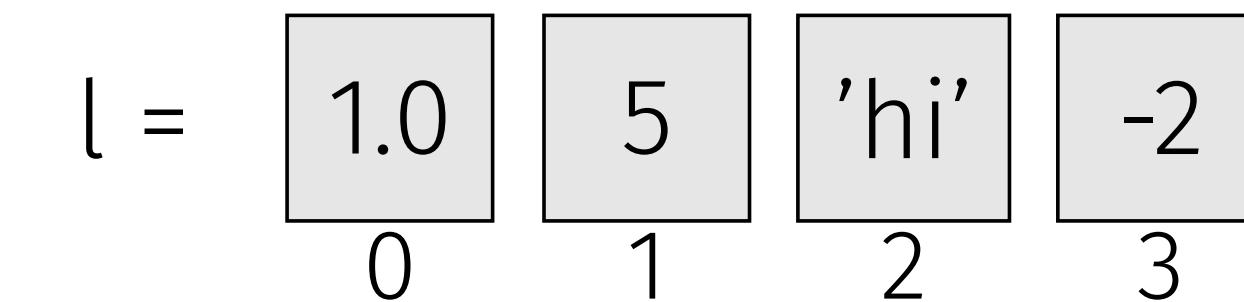
## ■ **tuple** (alle Objekttypen, unveränderlich)

```
t = (0, 'a', 3.3)
```



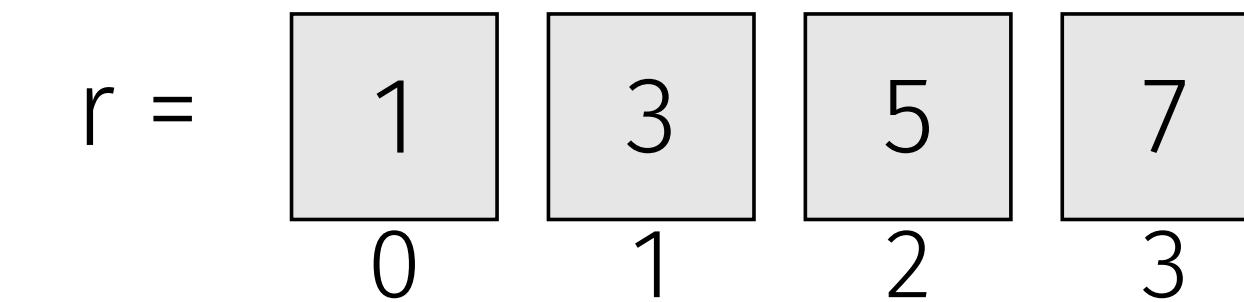
## ■ **list** (alle Objekttypen, veränderlich)

```
l = [1.0, 5, 'hi', -2]
```



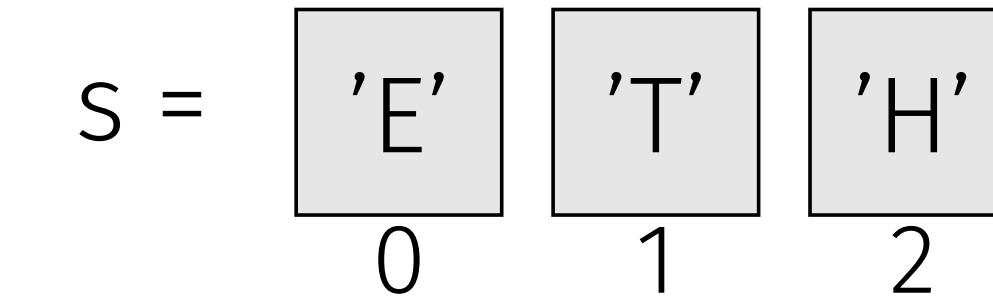
## ■ **range** (Integers, unveränderlich)

```
r = range(1,8,2)
```



## ■ **string** (Zeichen, unveränderlich)

```
s = "ETH"
```



# Operationen auf Sequences

- Subscript-Operator

```
s[i]
```

- Enumeration

- Verbinde jedes Element mit seiner Position.

```
for (i,x) in enumerate(s):  
    print(i,x)
```

sehr nützlich!

- Zip

- Verbinde zwei Sequenzen.

```
z = zip(s,t)  
l = list(z)
```

# Enumeration

$s = \begin{array}{c|c|c|c|c} & 2 & 3 & 5 & 8 & 13 \\ \hline 0 & & & & & \\ 1 & & & & & \\ 2 & & & \boxed{5} & & \\ 3 & & & & & \\ 4 & & & & & \end{array}$

```
for (i,x) in enumerate(s):  
    print(i,x)
```

0 2  
1 3  
2 5  
3 8  
4 13

Direkt Index und Element (objekt) zugänglich!  
ihr könnt direkt auf beide separat zugreifen

# Zip

$s = \begin{array}{c|c|c|c|c} 2 & 3 & 5 & 8 & 13 \\ \hline 0 & 1 & 2 & 3 & 4 \end{array}$

$t = \begin{array}{c|c|c|c|c} 3 & 6 & 9 & 12 & 15 \\ \hline 0 & 1 & 2 & 3 & 4 \end{array}$

```
z = zip(s,t)
l = list(z)
```

$l = \begin{array}{c|c|c|c|c} (2,3) & (3,6) & (5,9) & (8,12) & (13,15) \\ \hline 0 & 1 & 2 & 3 & 4 \end{array}$

# Slicing

wichtig!

Auswahl einer Subsequenz gemäss der folgenden Regel:

- Starte bei **start**, Stoppe **bevor stop**, Schrittgrösse **step**

```
s[start:stop:step]
s[start:stop] #step = 1
s[:stop:step] #start = 0
s[start::step] #stop = len(s)
```

standartwerte, wenn nichts angegeben

# Slicing

Auswahl einer Subsequenz gemäss der folgenden Regel:

- Starte bei **start**, stoppe **bevor stop**, Schrittgrösse **step**

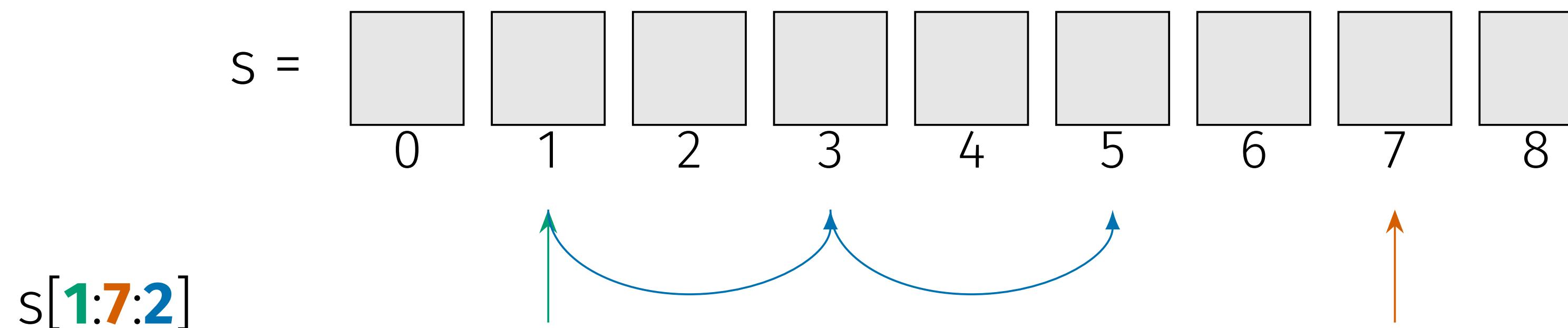
```
s[start:stop:step]
```

```
s[start:stop] #step = 1
```

```
s[:stop:step] #start = 0
```

```
s[start::step] #stop = len(s)
```

standardwerte, wenn nichts angegeben



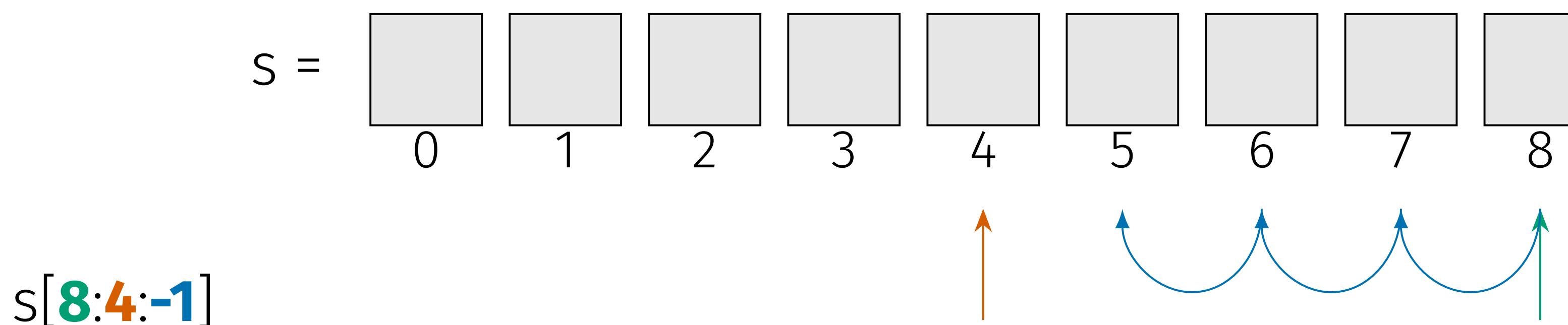
# Slicing

Auswahl einer Subsequenz gemäss der folgenden Regel:

- Starte bei **start**, stoppe **bevor stop**, Schrittgrösse **step**

```
s[start:stop:step]  
s[start:stop] #step = 1  
s[:stop:step] #start = 0  
s[start::-step] #stop = len(s)
```

Negativer **step**: Gehe zurück.



# Slicing negative Werte

---

s=[1,2,3,4,5,6,7,8,9,10]

- 
- **finden Sie s[-9:-2:3]**      **Lösung:** [2,5,8]
- **vorgehen:**
  - **element @ index= -1** ist das letzte (ganz rechts, hier: wert 10)
  - **runterzählen nach links** wir finden -9 ist beim Wert 2
  - von dort normal verfahren

# Index Error

## ein sehr häufiger Fehler

- Listen sind **keine Ringe!**
- => Indexe müssen im zulässigen Bereich liegen.
- **Trick**, falls ihr die Liste wie ein Ring behandeln wollt:
  - bsp. falls ihr in in der Mitte einer Liste starten und enden wollt (iteration)
  - `l[index % len(l)]`
  - wobei % der Modulo-Operator ist!
  - meiner Erinnerung nach ein sehr nützlicher Trick in Inf2 :)

# Slicing: Quiz

Für diese Folie kann Folgendes angenommen werden:

```
s = [1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Was ist der Output des folgenden Befehls?

```
s[3::5]
```

Wie würdest du die Sequenz s aufteilen, um den folgenden Output zu produzieren?

```
[34, 8, 2]
```

# Slicing: Quiz

Es sei folgende Sequenz s gegeben:

```
s = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

Wie würdest du diese Sequenz aufteilen, um die folgenden Listen zu erzeugen?

```
['E', 'F', 'G', 'H', 'I']
['L', 'K', 'J', 'I']
['C', 'H']
['O', 'L', 'I', 'F']
```

# Range

Eine Sequenz, die bei **start** beginnt, **bevor stop** endet mit der Schrittgrösse **step**.

```
range(start, stop, step)
range(start, stop) #step = 1
range(stop) #start = 0, step = 1
```

Range wird häufig in for-loops verwendet:

**Python**

```
for i in range(a, b, c):
    do_something
```

**C++**

```
for(int i=a; i<b; i+=c)
    do_something;
```

# Tupel

Eine generell unveränderliche Sequenz.

```
t = () #empty tuple  
t = (1,) #tuple with a single element  
t = (1,2) #tuple with two elements  
t = tuple(range(6)) #tuple from a range
```

# Quiz

Was ist der Output des folgenden Befehls?

```
tuple(range(3,15,4))
```

```
tuple(range(19,2,-2)[2:7:3])
```

Wie würdest du den folgenden Output mit einem range-Befehl generieren? Fällt dir ein anderer range-Befehl ein, der das gleiche Ergebnis erzielt? Wie viele solcher Möglichkeiten gibt es?

```
(2019, 2023, 2027)
```

range(2019,2028,4), der Endwert (stop) kann auch 2029, 2030 oder 2031 sein.

# Quiz

Wie würdest du den folgenden Output mit einem range-Befehl generieren?

`[-12, -6, 0, 6, 12]`

`range(-12,13,6), ..., range(-12,18,6)`

`[8, 4, 0, -4]`

`range(8,-5,-4), ..., range(8,-8,-4)`

Wie würdest du `range(15,-15,-3)` aufteilen, um den folgenden Output zu generieren?

`[-9, -3, 3, 9]`

`range(15,-15,-3)[8:1:-2], range(15,-15,-3)[-2:1:-2]`

# Methoden/Operartionen auf Listen

- Ändern eines Elements: `l[index] = desired_value`
- Anhängen eines Elements `l.append(new_element)`
- Entfernen eines Elements `l.remove(value) oder del l[index]`
- Häufigkeit eines Elements `anzahl = l.count(value)`
- Index eines Elements `position = l.index(value)`
- Sortieren der Liste `l.sort()`
- Umkehren der Liste `l.reverse()`

**aufgepasst:** Methoden wie bsp. `.reverse()` haben **keinen Rückgabewert**

# Quiz

Wie sieht die Liste l aus nach jedem einzelnen Schritt?

```
l = [0] * 4  
l[1] = 3  
l.append(5)  
l.reverse()  
del l[3]
```

l = [0,0,0,0]

# Collections

## ■ **set**

```
s = {1, 6, 2, 7}
```

**C++-Äquivalente:** std::set, std::unordered\_set

## ■ **dictionary (dict) key:value**

```
d = {1:3, 6:2, 2:6, 7:5}
```

**C++-Äquivalente** std::map, std::unordered\_map

# Dictionary crashkurs

eigentlich erst Thema nächste Woche, ihr benötigt es aber für eine Expert aufgabe :(

```
# beispiel dictionary:  
person = {'name': 'Alice', 'age': 30, 'city': 'New York'} #es gilt key:value  
  
#iterieren über dictionary:  
for key in person:  
    print(key, ':', person[key])  
  
#oder:  
for key, value in person.items():  
    print(key, ':', value,)  
  
#erstellen eines neuen dictionarys (ohne comprehension):  
squared_numbers = {}  
for key in range(1, 6): #key ist hier die Zahl von 1 bis 5, welche wir quadrieren wollen.  
    squared_numbers[key] = key ** 2  
  
print(squared_numbers)
```

ist im github repo ;)

# **Unterscheidung in der Anwendung von Container**

## **Welcher Container eignet sich am Besten um...**

- Alle E-Mail-Adressen für einen Newsletter zu speichern?
- Einen 6-Stelligen Pin aus Ziffern zu speichern?
- Alle Teilnehmer-Namen für ein Event zu speichern?
- Alle geraden Zahlen von 0 bis  $10^4$  zugänglich zu haben?
- Eine Lat&Long-Koordinate eines Notrufs zu speichern?
- Die Postleitzahlen verschiedener Gemeinden zu speichern?

**... und weshalb?**

# Exceptions

nur noch 4 Theorie Slides

# bsp: Unerwünschte Inputs abfangen

Du hast den User gebeten nur Ziffern einzugeben

```
try:  
    eingabe = input("Bitte gib dein Alter ein: ")  
    alter = int(eingabe) # Hier passiert der Fehler, wenn es keine Zahl ist  
    print(f"Cool, du bist {alter} Jahre alt.")  
  
except ValueError:  
    # Dieser Block wird nur ausgeführt, wenn int() scheitert  
    print("Fehler: Das war keine Zahl! Bitte benutze nur Ziffern (0-9).")
```

# Beispiel

- Exceptions werden ausgelöst, wenn das Programm syntaktisch korrekt ist, der Code jedoch zu einem Fehler geführt hat.
- Einige der am häufigsten vorkommenden Exceptions umfassen **IndexError**, **ImportError**, **IOError**, **ZeroDivisionError**, **TypeError**, and **FileNotFoundException**
- Das folgende Beispiel löst eine **ZeroDivisionError** Exception aus, da wir versuchen, eine Zahl durch 0 zu teilen.

```
a = 1000
b = a / 0
print(b)
```

```
ZeroDivisionError
Cell In[1], line 2
  1 a = 1000
----> 2 b = a / 0
      3 print(b)

ZeroDivisionError: division by zero
```

# Behandlung von Exceptions

- Wir können **Try**- und **Except**-Klauseln verwenden, um Exceptions zu behandeln.
- Eine try-Anweisung kann mehr als eine except-Klausel haben, um Handler für verschiedene Exceptions anzugeben.
- Es wird jedoch höchstens ein Handler ausgeführt.

```
a = [1, 2, 3]
try:
    print("Second element = %d" %(a[1]))
    # Throws error since there are only 3 elements in array
    print("Fourth element = %d" %(a[3]))
except IndexError:
    print("An error occurred")
```

## Output:

```
Second element = 2
An error occurred
```

# Finally

- **finally** definiert Code, der immer nach einem Try-and-Except-Block ausgeführt wird, unabhängig davon, ob eine Exception ausgelöst wird oder nicht.

```
try:  
    k = 5//0 # raises divide by zero exception.  
    print(k)  
  
# handles zerodivision exception  
except ZeroDivisionError:  
    print("Can't divide by zero")  
  
finally:  
    # this block is always executed  
    # regardless of exception generation.  
    print("This is always executed")
```

## Output:

```
Can't divide by zero  
This is always executed
```

# **Gruppenübung**

# Lesen von User-Input

```
word = input("Enter a word : ")
```

- Dieser Befehl schreibt den Text "Enter a word :" in die Konsole und wartet auf User-Input.
- Nachdem der User Text eingegeben hat, wird dieser in der Variable word als String gespeichert.

# Lesen von User-Input in einer Schleife

```
word = input("Enter a word : ")
again = True
while again:
    #Do something with word...
    word = input("Enter a word (or just <ENTER> to stop): ")
    again = len(word) > 0
```

- Dieser Code liest sequentiell String des Benutzers und verarbeitet sie.
- Wenn der Nutzer einen leeren String eingibt, wird das Programm beendet.

# Gruppenübung: Palindrome

Ein **Palindrom** ist ein Wort, das vorwärts- wie rückwärtsgesehen den gleichen Sinn ergibt.

Schreibe ein Python-Programm, das:

- sequentiell Wörter (potentiell mit Leerzeichen) vom User Input einliest.
- für jedes Wort per print ausgibt, ob es sich um ein Palindrom handelt.
- Wenn der Nutzer einen leeren String eingibt, automatisch terminiert.

Wechsle zu CodeExpert - Code Examples - Exercises 2 - In-class

**Hinweis:** Ein String ist eine Sequence. Alle Sequence-Operationen können auf Strings angewandt werden.

# Gruppenübung: Werte über dem Durchschnitt zählen

Schreibe ein Python-Programm mit dem folgenden In- und Output:<sup>2</sup>

**Input:** Eine Liste  $s$ , die Zahlen enthält.

**Output:** Die Anzahl der Zahlen in der Liste  $s$ , die streng grösser sind als der Durchschnittswert aller Elemente.

**Example:**  $s = [1, 1, 2, 3, 4, 1]$

Der Durchschnittswert in der Liste  $s$  ist gleich 2. Es gibt zwei Zahlen in  $s$ , die grösser sind als 2: 3 und 4. Daher sollte der Output der Funktion 2 sein.

---

<sup>2</sup>Optionale Aufgabe bei ausreichend Zeit.

# Gruppenübung: Werte über dem Durchschnitt zählen

Schreibe ein Python-Programm mit dem folgenden In- und Output:<sup>2</sup>

**Input:** Eine Liste  $s$ , die Zahlen enthält.

**Output:** Die Anzahl der Zahlen in der Liste  $s$ , die streng grösser sind als der Durchschnittswert aller Elemente.

**Example:**  $s = [1, 1, 2, 3, 4, 1]$

Der Durchschnittswert in der Liste  $s$  ist gleich 2. Es gibt zwei Zahlen in  $s$ , die grösser sind als 2: 3 und 4. Daher sollte der Output der Funktion 2 sein.

---

<sup>2</sup>Optionale Aufgabe bei ausreichend Zeit.

# Exercise 1: Python I, fällig 4.März

## Tipps & Hinweise

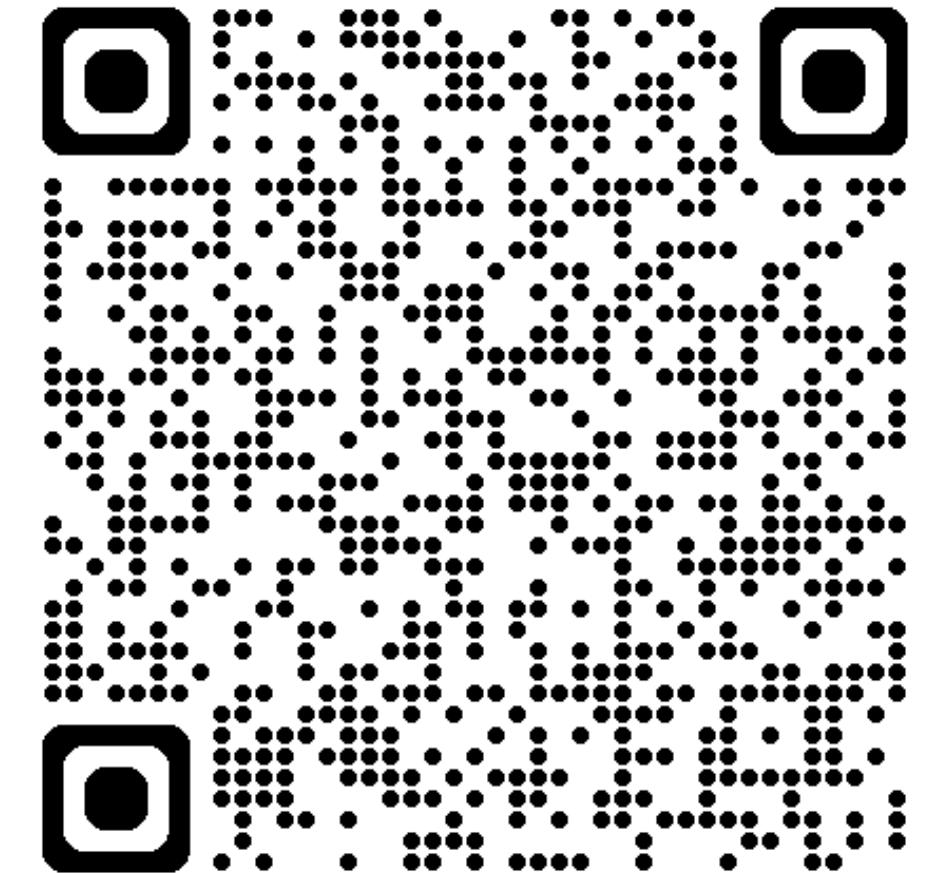
- **Sum and Maximum:** Sollte gut möglich sein, **keine built-in Funktionen verwenden!**
- **Bergprofil:** nicht verwirren lassen, ihr bekommt bereits die Liste mountain, sprich, der Input wird für euch bereits in eine Liste konvertiert. Das Ziel ist es nun, diese Liste Mountain zu analysieren.  
**Tipp für eine elegantere Lösung:** Benutzt die Methoden der Liste, so lassen sich viele Schlaufen vermeiden.  
**Ausserdem:** Falls ihr eine Liste kopieren wollt: denkt daran, sie tatsächlich zu kopieren.
- **Crops & Dictionaries:** die gezeigten Dictionary basics anschauen (Slides oder .ipynb) auf github.

es steht ihr solltet Comprehensions nutzen, habt ihr aber erst in der Vorlesung morgen...  
ich gebe euch auch ohne elegante comprehensions 10/10 ;)

# Bis nächste Woche :)

...nächste Woche:

- comprehensions
- aliasing/pointers
- dictionaries
- 



Feedback sehr willkommen, einfach reinschreiben