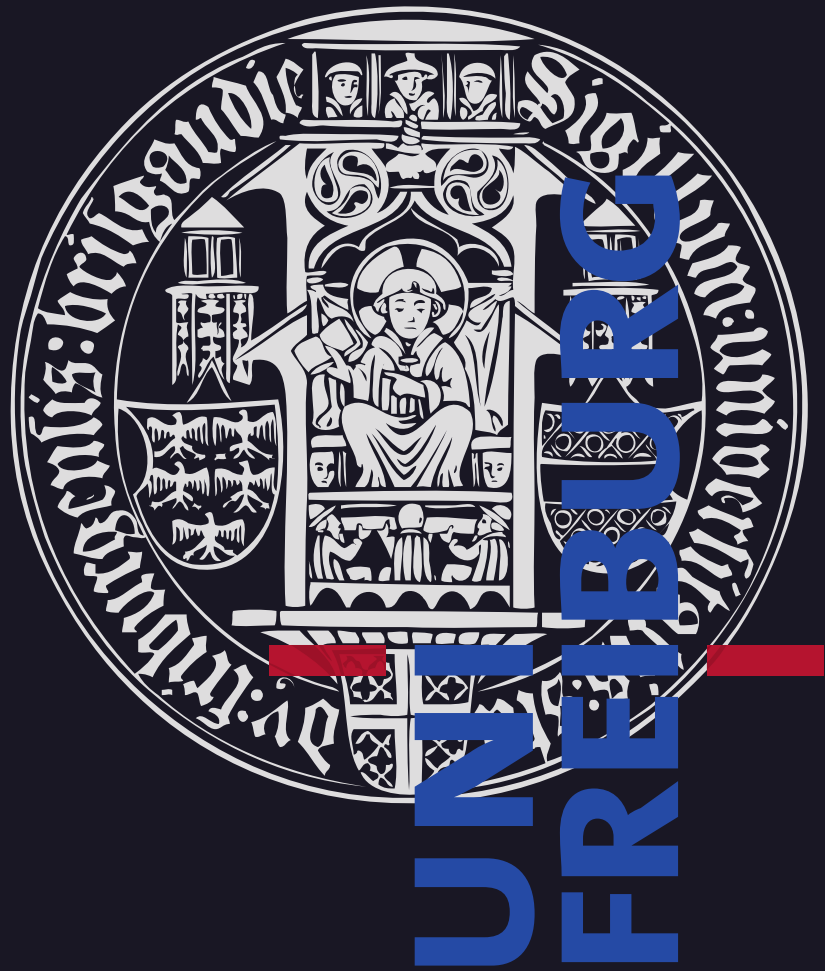
# Tutorium 04 - 2024-11-07

**Blatt 03, Iterieren von Sequenzen, Debugger, Blatt 04**



# Blatt 03

# Recap Vorlesung

**Was sind diese Listen?**

# List

- Listen benutzen `[,]` als Syntax
- Listen können aus beliebigen Elementen bestehen
- Es können Elemente entfernt und hinzugefügt werden

```
my_list_of_things: list[Any] = ["that's", "a", "list", 42]
my_numbers: list[int] = [42, 1337, 666]
```

# Tupel

- Tupel benutzen `(,)` als Syntax
- Tupel sind konstante Listen, einmal erstellt bleiben diese gleich

```
coords: tuple[float, float]
    = (48.013043566567289, 7.8340248371324135)
tutorium_location: tuple[str, str, tuple[float, float]]
    = ("Georges-Köhler-Allee 51", "00-034", coords)
```

# Strings?

- Strings sind auch nur Listen von `Chars`

```
s: list[str] = ['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
print("".join(s))
>>> hello world
for char in "hello world":
    print(char, end='')
>>> hello world
```

# Was iteriert man da eigentlich?

- `char` ist in Python von Typ `str`
- **Aber**, in vielen Sprachen wird der `String` als Liste von `Chars` abstrahiert

**Zum Beispiel in Rust:**

```
for ch: char in "hello world".chars() {
    println("{}", ch)
}
```

# Loops

**Wie iteriert man?**

# Loops

- Es gibt zwei verschiedene Loops
  - `while`
  - `for`
- mehr braucht man eigentlich auch nicht
- andere Sprachen haben noch
  - `do while`
  - `loop`

# for

- wir iterieren meisten durch eine Menge
- *Für jedes **x** in der Menge **X***

```
def is_prime(n: int) -> bool:
    if n < 2:
        return False
    for i in range(2, n // 2 + 1):
        if n % i == 0:
            return False
    return True
```

# `while`

- benötigt eine Bedingung
- meistens wird irgendetwas berechnet bis es eintritt

```
def next_prime(n: int) -> int:
    num = n + 1
    while not is_prime(num)
        num += 1
    return num
```

# Teaser - Rekursion

- Was passiert wenn wir einfach `next_prime` nochmal in `next_prime` aufrufen?
- Häufiger Fehler am Anfang im Programmieren

```python
def next_prime(n: int) -> int:
  num = n + 1
  if is_prime(num):
      return num
  else:
      return next_prime(num) # ??????
```

# Rekursion - was passiert?

- Ein Rekursionsbaum baut sich auf
- Wenn `num` keine Primzahl ist, dann rufen wir erneut `next_prime` auf
- Die erste `next_prime` und jede weitere, bei der `num` keine Primzahl ist, *warten* auf die jeweils nächste `next_prime`

```
next_prime(50)
next_prime(next_prime(51))
next_prime(next_prime(next_prime(52)))
next_prime(next_prime(next_prime(next_prime(53)))) # 53 ist eine Primzahl
>>> 53
```

# Was ist ein Debugger?

# Was ist ein Debugger?

- Debugger lassen euch euren Quellcode während dieser läuft (at runtime) anschauen
- Ihr könnt den kompletten Zustand anschauen
- So könnt ihr schnell Bugs finden

# Blatt 04