Zahlen, Listen und Schleifen

Richard Müller, Tom Felber

28. Oktober 2021

Python-Kurs

Gliederung

1. Wiederholung
2. Zahlen
3. Listen
4. for in Schleife
5. while Schleife

Wiederholung

Wiederholung

Beim letzten Mal:

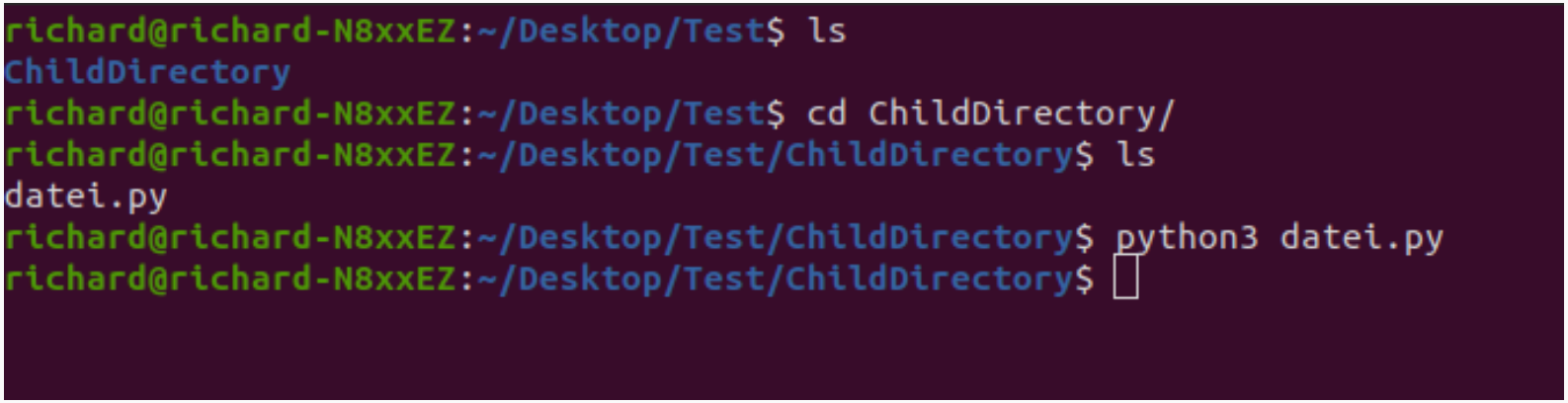
- der Python Interpreter
- "Hallo Welt" mit der `print()` Funktion
- Interaktion mit dem Terminal mit `input()`
- string Formatierung mit f-Strings

```
1 f'my string {4} {"vier"}'
```

- `if / else / elif` statements
- string methoden z.B. `isnumeric(), isupper()`

Wiederholung

- Navigation mir dem Terminal in das richtige directory



- starten eines Python-Scripts

```
1 python3 <textdatei.py>
```

- IndentationError: tritt z.B. auf, wenn Tabs und Leerzeichen nicht konsistent benutzt werden, oder bei flaschen Einrückungen

```
1 def bad_indentation(num):
2     if num < 10:
3         a = 10
4       b = 5
```

Zahlen

Zahlen

Die zwei Typen

Python kennt zwei verschiedene Zahlentypen:

- `int` (Integer/Ganzzahl)

```
number = 42
number = -42
```

- `float` (floating point number/Gleitkommazahl)

```
number = 42.1337
number = -42.1337
```

Zahlen

Mathematische Operationen

Auf allen Zahlen sind verschiedene mathematische Operationen möglich:

```
1 number = 5 + 3 # Addition
2 number = 5 - 3 # Subtraktion
3 number = 5 * 3 # Multiplikation
4 number = 5 / 3 # Division
5 number = 5 % 3 # Modulo
6 number = 5 ** 3 # Potenz
```

Achtung: Das Ergebnis wird bei der Division automatisch zu einem `float` konvertiert, selbst dann, wenn es keinen Rest gibt.

Zahlen

Man kann auch bestehende Variablen einfach mit einem Ergebnis überschreiben, ohne eine neue Variable anlegen zu müssen:

```
number = 5
number = number + 3
# das '+' mit der gewünschten Operation ersetzen
```

Dafür bietet Python eine Abkürzung:

```
# dieser Code tut das gleiche wie oben
number = 5
number += 3
# das '+' mit der gewünschten Operation ersetzen
```

Zahlen

Vergleichsoperationen

Es sind mehrere Vergleichsoperationen möglich. Jede dieser Operationen resultiert in einem `bool`:

```
5 == 5 # gleich
5 != 3 # ungleich
5 < 6 # kleiner
5 <= 6 # kleiner gleich
5 > 3 # größer
5 >= 3 # größer gleich
```

Alle Ausdrücke in diesem Beispiel nehmen den Wert `True` an.

Listen

Listen - Initialisierung

Listen sind eine Datenstruktur vom Typ `list`, eine geordnete Sammlung von Elementen.

```
# eine Liste initialisieren
prime_list = [2, 3, 5, 7, 11, 13]

# Listen können jegliche Objekte,
# auch gemischt und mehrfach enthalten
stuff = [5, 1.56363, 5, False, prime_list, None, False]
```

Indexierung startet in Python mit 0.

```
# einzelne Elemente referenzieren
first_prime = prime_list[0] #2
third_prime = prime_list[2] #5
last_prime = prime_list[-1] #13
# Elemente zu schreiben ist auch möglich
prime_list[0] = 42
```

Listen - append/remove

Listen bieten Funktionen zur einfachen Veränderung

```
prime_list = [2, 3, 5, 7, 11, 13]
```

```
# neue Elemente anfügen
prime_list.append(15)
print(prime_list) #[2, 3, 5, 7, 11, 13, 15]
```

```
# Elemente entfernen
prime_list.remove(11)
print(prime_list) #[2, 3, 5, 7, 13, 15]
```

```
# Elemente per Index entfernen
last_prime = prime_list.pop() #13
print(prime_list) #[2, 3, 5, 7, 11]
```

for in Schleife

for in Schleife

Die for-Schleife wird genutzt, um über eine Sequenz zu iterieren, z.B. Listen oder Strings. Dabei definiert sie eine Variable, die nach jeder Iteration den Wert des nächsten Elementes annimmt:

```
1 # Iteration über eine Liste
2 cities = ['Tokio', 'Rio', 'Denver', 'Berlin', 'Nairobi', '
      Moskau', 'Helsinki', 'Oslo']
3 for city in cities:
4     print(city)
```

```
1 # Iteration über String
2 for letter in 'abcdefg':
3     print(letter)
```

Die Schleife bricht automatisch ab, wenn es kein nächstes Element mehr gibt.

for in Schleife

Die `range()`-Funktion gibt eine Sequenz aus Zahlen zurück. Grundsätzlich gibt sie, bei der Eingabe *n*, die Zahlen 0 bis *n*-1 zurück. In diesem Beispiel also würde die Variable `i` nacheinander die Werte 0 bis 999 annehmen:

```
1 # Iteration über eine range
2 for i in range(1000):
3     print(i)
```

Achtung: Die durch die `for`-Schleife definierte Variable bleibt auch nach Beendigung der Schleife noch erhalten. Sie besitzt dann den Wert, den sie nach der letzten Iteration hatte (hier also 999).

while Schleife

while Schleife

Die `while`-Schleife prüft vor jeder Iteration, ob eine gegebene Bedingung den Wert `True` besitzt:

```
1 while bedingung:
2     print("Die Bedingung ist wahr")
```

Die Schleife bricht erst ab, wenn die Bedingung den Wert `False` annimmt:

```
1 number = 0
2
3 # die Bedingung wird erst False, wenn number größer oder
      gleich 5 ist
4 while number < 5:
5     print(number)
6     number += 1
```

while Schleife

Die Bedingung muss während der Schleifenausführung verändert werden, sonst ist die Schleife endlos:

```
1 while True:
2     print("Endlosschleife")
```

Man kann das `break`-Statement verwenden, um die Schleife auch unabhängig der Bedingung zu unterbrechen (geht auch bei der `for`-Schleife):

```
1 while True:
2     # es wird nur einmal "Endlosschleife" ausgegeben
3     print("Endlosschleife")
4     break
```