

# Einführung in Python: Kontrollstrukturen

Ludwig Ettner

11. Juli 2024

# übersicht

- 1 Boolesche Ausdrücke
- 2 Bedingungen (if, else, elif)
- 3 Schleifen (while, for)
- 4 Erweiterte Funktionen der for-Schleife
- 5 Fazit

# Boolesche Ausdrücke

- Boolesche Ausdrücke sind Ausdrücke, die entweder wahr (True) oder falsch (False) sind.
- Sie werden häufig verwendet, um Bedingungen auszudrücken oder Entscheidungen zu treffen.

# Logische Operatoren

- **AND (und)** - **and**: Wahr, wenn beide Operanden wahr sind. Beispiel: `True and True` ergibt `True`.
- **OR (oder)** - **or**: Wahr, wenn mindestens ein Operand wahr ist. Beispiel: `True or False` ergibt `True`.
- **NOT (nicht)** - **not**: Kehrt den Wert des Operanden um. Beispiel: `not True` ergibt `False`.

# Vergleichsoperatoren

- Größer als ( $>$ )
- Kleiner als ( $<$ )
- Größer oder gleich ( $>=$ )
- Kleiner oder gleich ( $<=$ )
- Gleich ( $==$ )
- Ungleich ( $!=$ )

# Bedingungen

- Bedingungen ermöglichen es uns, Codeblöcke nur dann auszuführen, wenn bestimmte Bedingungen erfüllt sind.
- **if-Bedingung:**

```
1 if Bedingung:  
2     # Code wird ausgef\ "uhrt, wenn die Bedingung  
3     # wahr ist
```

- **else-Bedingung:**

```
1 else:  
2     # Code wird ausgef\ "uhrt, wenn die Bedingung  
3     # falsch ist
```

# Erweiterte Bedingungen mit elif

- elif-Bedingung:

```
1 if Bedingung1:
2     # Code wird ausgefuehrt, wenn Bedingung1
3     # wahr ist
4 elif Bedingung2:
5     # Code wird ausgefuehrt, wenn Bedingung1
6     # falsch, aber Bedingung2 wahr ist
7 else:
8     # Code wird ausgefuehrt, wenn keine der
9     # Bedingungen wahr ist
```

# while-Schleife

- Eine while-Schleife wird verwendet, um einen Codeblock solange zu wiederholen, wie eine bestimmte Bedingung wahr ist.
- **Syntax:**

```
1 while Bedingung:  
2     # Code wird wiederholt ausgeführt,  
3     # solange die Bedingung wahr ist
```



# while-Schleife Beispiel

```
1 counter = 0
2 while counter < 5:
3     print("Counter:", counter)
4     counter += 1
```

- Ausgabe:

```
1 Counter: 0
2 Counter: 1
3 Counter: 2
4 Counter: 3
5 Counter: 4
```

# for-Schleife

- Eine for-Schleife wird verwendet, um einen Codeblock für jedes Element in einer Sequenz auszuführen.
- **Syntax:**

```
1 for Element in Sequenz:  
2     # Code wird fuer jedes Element in  
3     # der Sequenz ausgefuehrt
```

# for-Schleife Beispiel

```
1 fruits = ["Apple", "Banana", "Orange"]  
2 for fruit in fruits:  
3     print(fruit)
```

- Ausgabe:

```
1 Apple  
2 Banana  
3 Orange
```

# Erweiterte Funktionen der for-Schleife

- For-Schleifen können mit Listen und Dictionaries verwendet werden, um komplexe Datenstrukturen zu durchlaufen.

# For-Schleife mit Listen

- List Comprehensions: Eine kompakte Möglichkeit, Listen zu erstellen.

```
1 squares = [x**2 for x in range(10)]  
2 print(squares)
```

- Ausgabe:

```
1 [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# For-Schleife mit Dictionaries

- Dictionaries durchlaufen:

```
1 student_grades = {"Alice": 85, "Bob": 72, "  
    Charlie": 90}  
2 for student, grade in student_grades.items():  
3     print(f"{student}: {grade}")
```

- Ausgabe:

```
1 Alice: 85  
2 Bob: 72  
3 Charlie: 90
```

# Nützliche Funktionen

- `enumerate()`: Fügt den Elementen in der Schleife einen Zähler hinzu.

```
1 fruits = ["Apple", "Banana", "Orange"]
2 for index, fruit in enumerate(fruits):
3     print(index, fruit)
```

- `zip()`: Ermöglicht das Durchlaufen mehrerer Listen gleichzeitig.

```
1 names = ["Alice", "Bob", "Charlie"]
2 grades = [85, 72, 90]
3 for name, grade in zip(names, grades):
4     print(f"{name}: {grade}")
```

- Kontrollstrukturen sind essenziell, um den Programmfluss zu steuern.
- Boolesche Ausdrücke, Bedingungen und Schleifen bieten vielseitige Möglichkeiten.
- Erweiterte Funktionen der for-Schleife ermöglichen effizientes Arbeiten mit Listen und Dictionaries.