

Informatik I: Einführung in die Programmierung

5. Bedingungen, bedingte Ausführung

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Peter Thiemann

30. Oktober 2018

1 Bedingungen und der Typ `bool`



- Typ `bool`
- Vergleichsoperationen
- Logische Operatoren

Bedingungen

Typ `bool`

Vergleichsoperationen

Logische Operatoren

Bedingte Anweisungen

Anwendung

Zusammenfassung



- Neben *arithmetischen Ausdrücken* gibt es noch **Boolesche Ausdrücke** mit `True` oder `False` als Werten.
- Die einfachsten Booleschen Ausdrücke sind Vergleiche mit dem Gleichheitsoperator `==`.
- Die Werte `True` und `False` gehören zum Typ **`bool`**.
- Arithmetische Operationen konvertieren Boolesche Werte nach `int` (`False` \mapsto 0, `True` \mapsto 1):

Python-Interpreter

```
>>> 42 == 42
True
>>> 'egg' == 'spam'
False
>>> type('egg' == 'spam')
<class 'bool'>
>>> True + True
2
```

Bedingungen

Typ **`bool`**

Vergleichsoperatio-
nen

Logische
Operatoren

Bedingte An-
weisungen

Anwendung

Zusammen-
fassung

Es gibt die folgenden Vergleichsoperatoren:

Syntax	Bedeutung
<code>x == y</code>	Ist x gleich y?
<code>x != y</code>	Ist x ungleich y?
<code>x > y</code>	Ist x echt größer als y?
<code>x < y</code>	Ist x echt kleiner als y?
<code>x >= y</code>	Ist x größer oder gleich y?
<code>x <= y</code>	Ist x kleiner oder gleich y?

Python-Interpreter

```
>>> 2.1 - 2.0 > 0.1
```

```
True
```

```
>>> 2 - 1 < 1
```

```
False
```

```
>>> False < True
```

```
True
```

Bedingungen

Typ **bool**
Vergleichsoperato-
ren

Logische
Operatoren

Bedingte An-
weisungen

Anwendung

Zusammen-
fassung

Strings werden anhand der **lexikographischen Ordnung** verglichen, wobei für Einzelzeichen der Unicode-Wert (Ergebnis der `ord`-Funktion) benutzt wird.

Python-Interpreter

```
>>> 'anton' < 'antonia'  
True  
>>> 'anton' < 'berta'  
True  
>>> 'anton' < 'ulf'  
True  
>>> 'antonia' < 'antonella'  
False
```

Bedingungen

Typ **bool**
Vergleichsoperatio-
nen

Logische
Operatoren

Bedingte An-
weisungen

Anwendung

Zusammen-
fassung

Definition: Lexikographische Ordnung

Gegeben

Zwei Strings der Längen $m, n \geq 0$:

$$\vec{a} = "a_1 a_2 \dots a_m"$$

$$\vec{b} = "b_1 b_2 \dots b_n"$$

$\vec{a} \leq \vec{b}$ in der lexikographischen Ordnung, falls

Es gibt $0 \leq k \leq \min(m, n)$, so dass

- $a_1 = b_1, \dots, a_k = b_k$ und

$$\vec{a} = "a_1 a_2 \dots a_k a_{k+1} \dots a_m" \quad \vec{b} = "a_1 a_2 \dots a_k b_{k+1} \dots b_n"$$

- $k = m$

$$\vec{a} = "a_1 a_2 \dots a_m" \quad \vec{b} = "a_1 a_2 \dots a_m b_{m+1} \dots b_n"$$

- oder $k < m$ und $a_{k+1} \leq b_{k+1}$.

Bedingungen

Typ `bool`

Vergleichsoperatoren

Logische Operatoren

Bedingte Anweisungen

Anwendung

Zusammenfassung

- Werte unvergleichbarer Typen sind ungleich.
- Bei den Anordnungsrelationen gibt es einen Fehler, wenn die Typen nicht zusammenpassen!

Python-Interpreter

```
>>> 42 == 'zweiundvierzig'
```

```
False
```

```
>>> 41 < '42'
```

```
Traceback (most recent call last): ...
```

```
TypeError: unorderable types: int() < str()
```

Bedingungen

Typ **bool**
Vergleichsoperatio-
nen

Logische
Operatoren

Bedingte An-
weisungen

Anwendung

Zusammen-
fassung



- Es gibt die folgenden **logischen Operatoren**: `or`, `and`, `not` – mit aufsteigender Operatorpräzedenz.
- Wie die Bitoperationen mit (`False` \mapsto 0, `True` \mapsto 1), d.h.
 - `x < 10 or y > 100` hat den Wert `True`, wenn `x` kleiner als 10 ist oder wenn `y` größer als 100 ist.
 - `1 <= x and x <= 10` hat den Wert `True`, wenn `x` zwischen 1 und 10 (inklusive) liegt.
 - Alternative Schreibweise dafür: `1 <= x <= 10`.
 - `not(x < y)` ist `True` wenn `x` nicht kleiner als `y` ist.
- **Nullwerte** sind `None`, `0`, `0.0`, `(0 + 0j)` und `''`. Sie werden wie `False` behandelt, alle anderen Werte wie `True`!
- Die **Auswertung wird beendet**, wenn das Ergebnis klar ist.

Bedingungen

Typ `bool`

Vergleichsoperationen

Logische Operatoren

Bedingte Anweisungen

Anwendung

Zusammenfassung

Python-Interpreter

```
>>> 1 < 5 < 10
```

```
True
```

```
>>> 5 < 1 or 'spam' < 'egg'
```

```
False
```

```
>>> 'spam' or True
```

```
'spam'
```

```
>>> '' or 'default'
```

```
'default'
```

```
>>> 'good night' and 'ding ding ding'
```

```
'ding ding ding'
```

```
>>> 0 and 10 < 100
```

```
0
```

```
>>> not 'spam' and (None or 0.0 or 10 < 100)
```

```
False
```

Bedingungen

Typ **bool**

Vergleichsoperatio-
nen

Logische
Operatoren

Bedingte An-
weisungen

Anwendung

Zusammen-
fassung

2 Bedingte Anweisungen

- `if`-Anweisung
- `if-else`-Anweisung
- `elif`-Anweisung

Bedingungen

Bedingte Anweisungen

`if`-Anweisung

`if-else`
Anweisung

`elif`
Anweisung

Anwendung

Zusammenfassung

- Bisher wurde jede eingegebene Anweisung ausgeführt.
- Die **bedingte Anweisung** (Konditional, `if`-Anweisung) ermöglicht es, Anweisungen nur unter bestimmten Bedingungen auszuführen.

Python-Interpreter

```
>>> x = 3
>>> if x > 0:
...     print('x ist strikt positiv')
...
x ist strikt positiv
>>> x = 0
>>> if x > 0:
...     print('x ist strikt positiv')
...
>>>
```

Bedingungen

Bedingte Anweisungen

if-Anweisung

if-els

Anweisung

elif

Anweisung

Anwendung

Zusammenfassung

- Die **if-else-Anweisung** ermöglicht es, durch eine Bedingung zwischen zwei Anweisungen auszuwählen.
- Der **else-Zweig** wird ausgewertet, wenn die Bedingung **nicht** erfüllt ist.

Python-Interpreter

```
>>> x = 3
>>> if x%2 == 0:
...     print('x ist gerade')
... else:
...     print('x ist ungerade')
...
x ist ungerade
```

Bedingungen

Bedingte Anweisungen

if-Anweisung

if-else
Anweisung

elif
Anweisung

Anwendung

Zusammenfassung



- Eine **verkettete bedingte Anweisung** kann mehr als zwei Fälle behandeln.

Python-Interpreter

```
>>> x = 3
>>> y = 0
>>> if x < y:
...     print('x ist kleiner als y')
... elif x > y:
...     print('x ist größer als y')
... else:
...     print('x und y sind gleich')
...
x ist größer als y
```

- Die Bedingungen werden der Reihe nach ausgewertet. Der erste Block, dessen Bedingung erfüllt ist, wird ausgeführt.

Bedingungen

Bedingte Anweisungen

if-Anweisung

if-els
Anweisung

elif
Anweisung

Anwendung

Zusammenfassung

- Bedingte Anweisungen können geschachtelt werden.

Python-Interpreter

```
>>> x = 5
>>> if x > 0:
...     if x > 10:
...         print('successful encyclopedia salesman')
...     else:
...         print('unsuccessful encyclopedia salesman')
...
>>>
```

- Durch die Einrückung ist immer klar, wozu die bedingte Anweisung gehört!

Bedingungen

Bedingte Anweisungen

`if`-Anweisung

`if-els`
Anweisung

`elif`
Anweisung

Anwendung

Zusammenfassung

3 Anwendung

- Auswerten eines Tests
- Freizeitpark

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Anwendung — Auswerten eines Tests

Bestanden oder nicht?

In einem Test kann eine maximale Punktzahl erreicht werden. Ein gewisser Prozentsatz an Punkten ist notwendig um den Test zu bestehen.

Aufgabe

Entwickle eine Funktion, die die Eingaben

- maximale Punktzahl,
- Prozentsatz zum Bestehen und
- tatsächlich erreichte Punktzahl

nimmt und als Ergebnis entweder 'pass' oder 'fail' liefert.

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Aufgabe

Entwickle eine Funktion `test_result`, die die Eingaben

- `max_points: int` maximale Punktzahl,
- `percentage: int` Prozentsatz zum Bestehen und
- `points: int` tatsächlich erreichte Punktzahl

nimmt und als Ergebnis entweder 'pass' oder 'fail' (vom Typ `str`) liefert.

- Bezeichner für Funktion und Parameter festlegen
- Typen der Parameter angeben
- Typ des Rückgabewertes angeben

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



```
def test_result(  
    max_points: int,  
    percentage: int,  
    points: int) -> str:  
    # fill in  
    return
```

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

- Funktionsgerüst aufschreiben.
- Wenn klar ist, dass eine Zeile fortgesetzt werden muss (hier: innerhalb einer Parameterliste), wird das durch zusätzliche Einrückung gekennzeichnet.
- Typen werden durch **Typannotationen** “: int” für Parameter bzw. “-> str” für das Ergebnis angegeben (ab Python 3.6).



```
def test_result(  
    max_points, # int  
    percentage, # int  
    points      # int  
):             # -> str  
  
    # fill in  
    return
```

- Alternativ werden Typen durch **Typkommentare** `# int` bzw. `# -> str` angegeben (vor Python 3.6).

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



```
test_result(100, 50, 50) == 'pass'  
test_result(100, 50, 30) == 'fail'  
test_result(100, 50, 70) == 'pass'
```

- Sinnvolle Beispiele erarbeiten
 - Eingaben so wählen, dass alle mögliche Ergebnisse erreicht werden.
 - Randfälle bedenken (z.B. `points == max_points`, `points == 0`, `percentage == 0`, `percentage == 100`, ...)
- Ergebnisse der Beispiele von Hand ausrechnen!
- Die Beispiele dienen später als **Tests**, dass der Code zumindest für die Beispiele funktioniert.

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Schritt 4: Funktionsrumpf ausfüllen



```
def test_result(
    max_points: int,
    percentage: int,
    points: int) -> str:
    passed = (points
              >= max_points * percentage / 100)
    if passed:
        return 'pass'
    else:
        return 'fail'
```

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

- Die Zuweisung an `passed` erstreckt sich über **zwei Zeilen**.
- Dafür **muss** der Ausdruck rechts geklammert sein.
- Zeilenumbruch **vor** dem Operator `>=`.

- Fertig?
- Was ist, wenn
 - `max_points < 0?`
 - `percentage < 0?`
 - `percentage > 100?`
 - `points < 0?`
 - `points > max_points?`
- Wollen wir diese Fälle zulassen?

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Defensives Programmieren

Fange alle unerwünschten Fälle im Code ab und erzeuge eine Fehlermeldung.

Design by Contract

- **Spezifiziere** die Funktion und programmiere unter der Annahme, dass nur die zulässigen Fälle auftreten (wie im Codebeispiel).
- `max_points >= 0`
- `0 <= percentage <= 100`
- `0 <= points <= max_points`

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



Anwendung — Freizeitpark

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Mitfahren oder nicht?

In einem Freizeitpark gibt es verschiedene Attraktionen, die mit Alters- und Größenbeschränkungen belegt sind.

Beispiel

Attraktion	Beschränkung	Begleitung
Silver-Star	11 Jahre und 1,40m	—
Euro-Mir	8 Jahre und 1,30m	unter 10 Jahre
blue fire	7 Jahre und 1,30m	—
Eurosat	6 Jahre und 1,20m	unter 10 Jahre
Matterhorn-Blitz	6 Jahre und 1,20m	unter 8 Jahre
Tiroler Wildwasserbahn	4 Jahre und 1,00m	unter 9 Jahre

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Aufgabe

Entwickle eine Funktion zur Einlasskontrolle bei Euro-Mir, die als Eingaben

- das Alter,
- die Größe und
- ob ein erwachsener Begleiter dabei ist

nimmt und als Ergebnis entweder 'Du darfst mitfahren!' oder 'Du musst leider draussenbleiben.' liefert.

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



Aufgabe

Entwickle eine Funktion `enter_euro_mir` zur Einlasskontrolle bei Euro-Mir, die als Eingaben

- `age: int` das Alter (in Jahren),
- `height: int` die Größe (in cm) und
- `accompanied: bool` ob ein erwachsener Begleiter dabei ist

nimmt und als Ergebnis entweder `'Du darfst mitfahren!'` oder `'Du musst draussenbleiben.'` -> `str` liefert.

- Festlegen von **Einheiten** für die Eingaben!

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



```
def enter_euro_mir(  
    age: int,  
    height: int,  
    accompanied: bool  
    ) -> str:  
    # fill in  
    return
```

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung



```
enter_euro_mir(4, 101, 'Mama') == False
enter_euro_mir(8, 125, 'Papa') == False
enter_euro_mir(7, 130, 'Oma') == False
enter_euro_mir(9, 135, 'Opa') == True
enter_euro_mir(10, 135, '') == True
```

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

Schritt 4: Funktionsrumpf ausfüllen



```
def enter_euro_mir(
    age: int,
    height: int,
    accompanied: bool
) -> str:
    age_ok = age >= 8
    height_ok = heigher >= 130
    admitted = (age_ok
        and height_ok
        and (age >= 10 or accompanied))
    if admitted:
        return 'Du darfst mitfahren!'
    else:
        return 'Du musst draussenbleiben.'
```

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

- Entwickle eine `enter` Funktion, die die Bedingungen aus den globalen Variablen `min_age`, `min_height` und `min_age_alone` berechnet.
- Ändere die Funktion, so dass sie bei einer Zurückweisung den Grund angibt. Zum Beispiel `'Du_bist_zu_klein.'`, `'Du_bist_zu_jung.'` usw.

Bedingungen

Bedingte Anweisungen

Anwendung

Auswerten eines Tests

Freizeitpark

Zusammenfassung

4 Zusammenfassung



**UNI
FREIBURG**

Bedingungen

Bedingte An-
weisungen

Anwendung

**Zusammen-
fassung**



- `bool` ist ein weiterer Typ, dessen einzige Werte `True` und `False` sind.
- Vergleiche, wie z.B. `==` oder `<`, liefern **Boolesche Werte**.
- Boolesche Werte werden bei Bedarf nach `int` konvertiert, wobei `True` \mapsto 1 und `False` \mapsto 0 gilt.
- **Nullwerte** werden als `False` interpretiert, alle anderen Werte als `True`.
- **Bedingte Anweisungen** (`if-(elif)-else`) erlauben die Auswahl zwischen alternativen Anweisungen.
- **Checkliste** zum Entwurf von Funktionen: Bezeichner und Datentypen, Funktionsgerüst, Beispiele, Funktionsrumpf

Bedingungen

Bedingte Anweisungen

Anwendung

Zusammenfassung