## 3. Eigene Exception auslösen

Wie im Kapitel 2 erwähnt, sind es oft Benutzereingaben, die zu einem Fehlverhalten bei einer Software führen können - sofern man diese Eingaben eben nicht im Voraus prüft. Dazu folgendes Beispiel.

---

*Beispiel 3.2: Personendaten erfassen*
In eine Eingabefeld kann die Körpergrösse eines Menschen eingegeben werden.

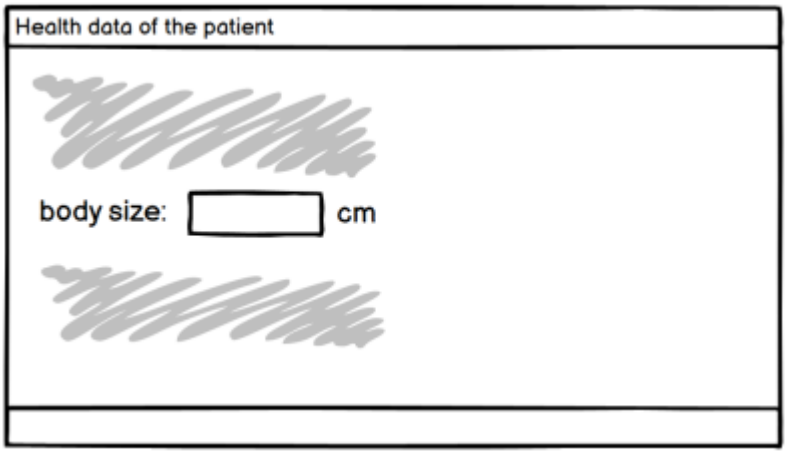


*Abb 3.3:* Eingabe mit begrenztem Wertebereich

Es ist klar ersichtlich, dass für die Eingabe der Körpergrösse negative Werte aber auch Werte über ca. 250cm keinen Sinn machen. Diese Situation kann übewacht und mit einem eigenen Fehlertyp (z.B. BodySizeError) signalisiert werden.

---

**Warum wirft man eigene Exception?**
Ein Fehler stellt immer eine Ausnahme im Ablauf des Programms dar. Natürlich kann man z.B. den Wert -1 (bei Zahlen) als Fehlercode zurückgaben. Aber was, wenn eben auch negative Werte zulässig sind? Dieses Beispiel haben wir in der Aufgabe 4 der LU02 bei der Klasse `BankAccount` gesehen. Dort darf der Saldo (`balance`) bis zu einem gewissen Wert negativ sein. Aber welcher negative Wert soll dann einen Fehler signalisieren?
Es ist also um vieles besser, wenn im Fehlerfall ein entsprechendes Fehlerobjekt erzeugt und geworfen wird. Auf dieses Fehlerobjekt kann dann an anderer Stelle im Code mittels `try-except` reagiert werden.

---

*Beispiel 3.3: Körpergrösse überwachen*

```
class BodySizeError(Exception):
    """
    Dieser Fehlerfall wird ausgelöst, wenn die vom
    Benutzer eingegebene Körpergrösse nicht im Bereich
    20...250 cm liegt.
    """
    def __init__(self, size):
```

```
        super().__init__(f'Der Wert {size} liegt nicht im Bereich 20...250
cm')


class Person():
    def __init__(self, name):
        self._name = name
        self._body_size = 0

    @property
    def body_size(self):
        return self._body_size

    @body_size.setter
    def body_size(self, size):
        if 50 <= size <= 250:
            self._body_size = size
        else:
            raise BodySizeError(size)

if __name__ == '__main__':
    jonathan = Person('Jonathan')
    try:
        size = int(input('Körpergrösse: '))
        jonathan.body_size = size
    except BodySizeError as body_size_error:
        print(body_size_error)
    # OK hier gehts weiter
    print('...')
```

Kopieren Sie den Code in Ihre Entwicklungsumgebung und spielen Sie mit ein paar Eingabewerten. Schauen Sie, was der jeweilge Effekt ist.

---

Im Folgenden betrachten wir die Codesequenzen, die wichtig sind und bei denen neue Techniken angewendet werden.

1. Die Deklaration der eigenen Fehlerklasse als Ableitung der Klasse `Exception`.
   Das Thema Vererbung werden Sie erst in der LU06 kennenlernen. Dennoch brauchen wir diese Technik hier, um eine eigene Exception-Klasse zu erstellen. Der Code dazu sieht wie folgt aus:

   ```
   class BodySizeError(Exception): # Vererbung der Eigenschaften der
   Klasse Exception an die Klasse BodySizeError
   ```

2. Aufruf des Konstruktors in der Oberklasse (hier Exception)
   In der Regel verwendet man den Konstruktor der Oberklasse, um das Objekt vollständig zu initialisieren. Das sieht dann wie Folgt aus:

   ```
   super().__init__(f'Der Wert {size} liegt nicht im Bereich 20...250 cm')
   ```

```
# Aufruf des super-Konstruktors mit den entsprechenden Parametern.
```

Offenbar kann die Klasse `Excpetion` eine Fehlermeldung ausgeben. Es ist also wichtig, an Hand einer Dokumentation die Funktionalität der Oberklasse in Erfahrung zu bringen.

3. Auslösen der Exception im eigenen Code.

```
if 50 <= size <= 250:
   self._body_size = size
else:
   raise BodySizeError(size) # hier wird das Fehlerobjekt erzeugt und
"geworfen"
```

Mittels einer Bedingung wird die Gültigekit eines Wertes überprüft. Im Fehlerfall wird durch `raise` die Exception zuerst erzeugt und dann geworfen. Im Code ist ersichtlich, dass hier nicht eine Objektreferenz sondern die Klasse (`BodySizeError`) angeschrieben wird. Das heisst, dass in diesem Moment der Programmausführung ein entsprechendes Objekt erzeugt wird. (vrgl dazu die Skizze unten)

4. Fehlerobjekt benennen und Aktion für den Fehlerfall ausführen.

```
except BodySizeError as body_size_error:  # das geworfene Objekt vom
Typ BodySizeError benennen
  print(body_size_error)
```

Wir geben bei `except` an, welchen Fehlertyp wir erwarten und wie das Fehlerobjekt heissen soll.

### Was läuft im Code ab?

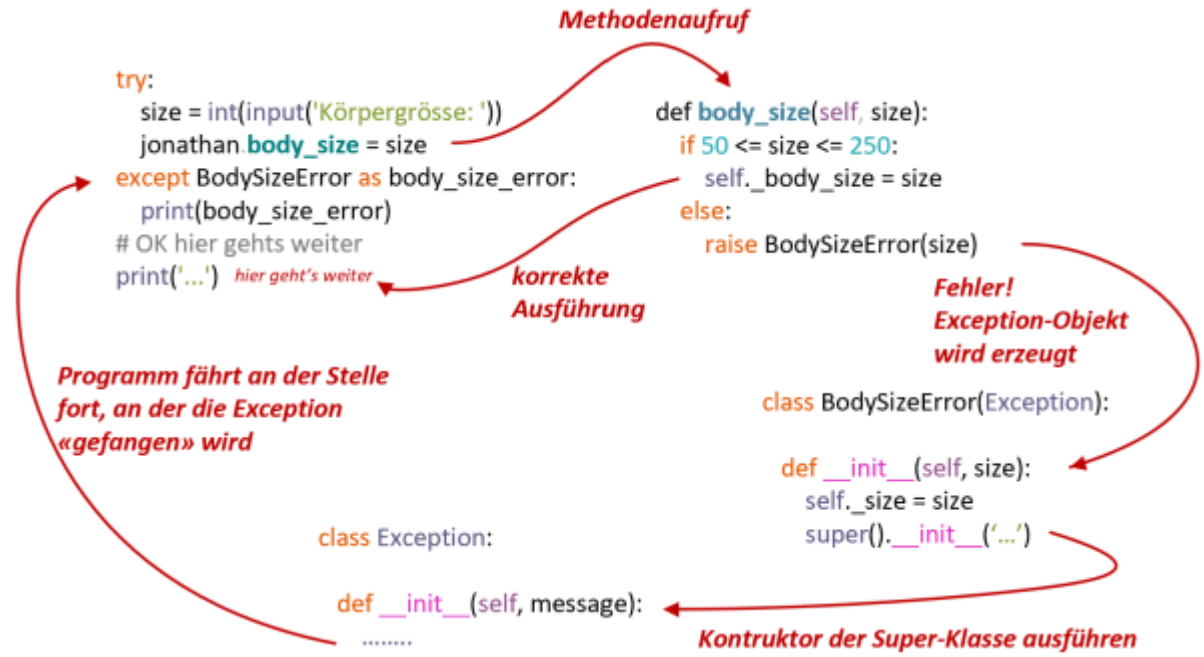


*Abb 3.4:* Ablauf bei einer Exception