# Ausnahmebehandlung

Eine Ausnahme (exception) ist eine Abweichung von der Normalität. Bei der Programmierung ist eine Ausnahme ein Fehler, der während der Programmausführung auftritt. Bei der Programmausführung kann es folglich zu Fehlern kommen, die ein Programmierer nicht eingeplant hat (bspw. wird ein Buchstabe statt einer Zahl eingegeben). Dabei führen Fehler bei der Programmausführung zum Abbruch des Programms. Mit der Ausnahmebehandlung (eng. exception handling) können Fehler „aufgefangen“ und „repariert“ werden. Sequenzen von Code bei der die Ausnahmeprüfung „aktiv“ sein soll, werden in Python in einem *try*-Block ausgeführt. Die Ausnahmen werden dann im *except*-Block „behandelt“.

|  |  |
| --- | --- |
| 1  2  3  4 | try:  #something  except:  print("Ein Fehler ist aufgetreten") |

## Ausnahmebehandlung - *ValueError*

Der ValueError kann beispielsweise vorkommen, wenn ein Nutzer keine Zahl eingibt, die Eingabe aber in einer Zahl umgewandelt wird (durch *float()* oder *int()* ).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | class Konto:  def \_\_init\_\_(self, p\_inhaber, p\_kontostand):  self.inhaber = str(p\_inhaber)  self.kontostand = float(p\_kontostand)    def abheben(self):  while True:  try:  betrag=float(input("Bitte geben Sie den gewünschten Betrag ein:"))  self.kontostand = self.kontostand - betrag  print("Neuer Kontostand", self.kontostand, "EUR")  break  except ValueError:  print("Fehler! Geben Sie eine Zahl ein!")  # Hauptprogramm  k1 = Konto("Simon Amsel", 500.0)  k1.abheben() |

Die Methode *abheben()* beinhaltet eine Endlosschleife (*while True*) die so lange ausgeführt wird, bis das break die Schleife beendet, d. h. bis der Programmcode einmal korrekt ausgeführt wurde. Gibt der Nutzer in Zeile 9 keine Zahl ein, entsteht ein ValueError, weil die Eingabe nicht in eine Zahl umgewandelt werden kann. Das Programm springt automatisch in den except-Block und führt den darin enthaltenen Code aus. Da die Schleife in dem except-Block nicht mit break beendet wird, wird der try-Block so lange ausgeführt, bis in Zeile 9 eine Zahl eingegeben wurde und somit keine Ausnahme auftritt. Dann wird der except-Block auch nicht ausgeführt, sondern übersprungen.

**Ausgabe des Programms:**

|  |
| --- |
| Bitte geben Sie den gewünschten Betrag ein: **a**  Fehler! Geben Sie eine Zahl ein!  Bitte geben Sie den gewünschten Betrag ein: **10**  Neuer Kontostand 490.0 EUR |

## Ausnahmebehandlung assert - *AssertionError*

Mit dem Keyword assert kann getestet werden, ob eine Bedingung wahr (True) ist. Trifft dies nicht zu, d. h. ist die Bedingung False, wird ein AssertionError erzeugt. Dieser kann später im Code verarbeitet werden (Zeile 12 – 13).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | class Konto:  […]  def abheben(self):  while True:  try:  betrag=float(input("Bitte geben Sie den gewünschten Betrag ein: "))  assert betrag > 0  self.kontostand = self.kontostand - betrag  print("Neuer Kontostand", self.kontostand, "EUR")  break  except ValueError:  print("Fehler! Geben Sie eine Zahl ein!")  except AssertionError:  print("Geben Sie eine positive Zahl ein!")  # Hauptprogramm  k1 = Konto("Simon Amsel", 500.0)  k1.abheben() |

Das Programm prüft nun neben der korrekten Eingabe einer Zahl auch den Fall, ob der Nutzer eine positive Zahl eingegeben hat. Hat er dies nicht, wird der except-Block mit dem AssertionError ausgeführt. Die *while*-Schleife wird nicht durch ein break beendet und der Benutzer wird erneut aufgefordert eine positive Zahl einzugeben. Erst wenn der Nutzer dieser Aufforderung nachkommt, wird die *while*-Scheife durch ein break beendet.

**Ausgabe des Programms:**

|  |
| --- |
| Bitte geben Sie den gewünschten Betrag ein: **a**  Fehler! Geben Sie eine Zahl ein!  Bitte geben Sie den gewünschten Betrag ein: **-100**  Geben Sie eine positive Zahl ein!  Bitte geben Sie den gewünschten Betrag ein: **100**  Neuer Kontostand 400.0 EUR |

## Ausnahmebehandlung *finally*

Neben dem *try*-Block und dem *except*-Block kann auch ein *finally*-Block verwendet werden. Der *finally*-Block wird immer ausgeführt. Egal ob ein Fehler auftritt oder nicht.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class Konto:  […]  def abheben(self):  while True:  try:  betrag=float(input("Bitte geben Sie den gewünschten Betrag ein: "))  assert betrag > 0  self.kontostand = self.kontostand - betrag  print("Neuer Kontostand", self.kontostand, "EUR")  break  except ValueError:  print("Fehler! Geben Sie eine Zahl ein!")    except AssertionError:  print("Geben Sie eine positive Zahl ein!")    finally:  print("Kennen Sie schon unser neues Prämiensparen?")  # Hauptprogramm  k1 = Konto("Simon Amsel", 500.0)  k1.abheben() |

Im abgebildeten Beispiel wird der finally-Block in jedem Schleifendurchlauf ausgeführt, also auch wenn ein Buchstabe oder keine positive Zahl eingeben wird.

**Ausgabe des Programms:**

|  |
| --- |
| Bitte geben Sie den gewünschten Betrag ein: **-1**  Geben Sie eine positive Zahl ein!  Kennen Sie schon unser neues Prämiensparen?  Bitte geben Sie den gewünschten Betrag ein: **a**  Fehler! Geben Sie eine Zahl ein!  Kennen Sie schon unser neues Prämiensparen?  Bitte geben Sie den gewünschten Betrag ein: **100**  Neuer Kontostand 400.0 EUR  Kennen Sie schon unser neues Prämiensparen? |

Weiteres Beispiel für den *finally*-Block:

In Python können Dateien (bspw. Textdateien) geöffnet und eingelesen werden. Betriebssysteme können allerdings nur eine bestimmte Anzahl von Dateien gleichzeitig öffnen. Deswegen müssen geöffnete Dateien nach der Bearbeitung auch wieder geschlossen werden. Tritt nun ein Fehler auf, während eine Datei geöffnet ist, wird diese nie geschlossen, weil der *try*-Block nach dem Fehler nicht mehr ausgeführt wird. Der *finally*-Block kann jetzt verwendet werden, um die Datei zu schließen. So ist gewährleistet, dass die Datei immer geschlossen wird, egal ob ein Fehler auftritt oder nicht.