# Exceptions

## 1. Einführung in Ausnahmen

In der Programmierung sind Ausnahmen spezielle Ereignisse, die während der Ausführung eines Programms auftreten und die normale Ausführung unterbrechen. In Python gibt es verschiedene Gründe, warum eine Ausnahme auftreten kann, darunter:

- **Fehler in der Logik**: Zum Beispiel, wenn du versuchst, durch Null zu dividieren.
- **Unerwartete Eingaben**: Zum Beispiel, wenn ein Benutzer einen nicht numerischen Wert eingibt, wenn eine Zahl erwartet wird.
- **Ressourcenprobleme**: Zum Beispiel, wenn eine Datei nicht gefunden wird, die das Programm öffnen möchte.

Die Ausnahmebehandlung ermöglicht es dir, mit solchen Situationen umzugehen, ohne dass das Programm abstürzt. Anstatt das Programm abrupt zu beenden, kannst du die Ausnahme abfangen und eine geeignete Reaktion darauf definieren.

## 2. Grundlegende Struktur zur Behandlung von Ausnahmen

Die grundlegende Struktur zur Behandlung von Ausnahmen in Python verwendet die Schlüsselwörter `try` und `except`. Hier ist ein einfaches Beispiel:

```
try:
    # Code, der eine Ausnahme auslösen könnte
    result = 10 / 0
except:
    # Dieser Block wird ausgeführt, wenn eine ZeroDivisionError-Ausnahme au
    print("Fehler: Division durch Null ist nicht erlaubt.")
```

In diesem Beispiel wird der Versuch, durch Null zu dividieren, eine `ZeroDivisionError`-Ausnahme auslösen. Der Code im `except`-Block wird ausgeführt, um auf diese Ausnahme zu reagieren.

## 3. Reihenfolge der `except`-Blöcke

Die Reihenfolge, in der die `except`-Blöcke definiert sind, ist entscheidend, da Python die `except`-Blöcke der Reihe nach überprüft, um zu entscheiden, welcher Block

ausgeführt werden soll. Der erste `except` -Block, der mit der aufgetretenen Ausnahme übereinstimmt, wird ausgeführt, und alle nachfolgenden Blöcke werden ignoriert.

Hier ist ein Beispiel, das die Bedeutung der Reihenfolge verdeutlicht:

```
try:
    value = int(input("Geben Sie eine Zahl ein: "))
    result = 10 / value
except ValueError:
    print("Fehler: Ungültige Eingabe. Bitte geben Sie eine Zahl ein.")
except ZeroDivisionError:
    print("Fehler: Division durch Null ist nicht erlaubt.")
except Exception as e:
    print(type(e), e)
```

In diesem Beispiel wird zuerst versucht, eine Benutzereingabe in eine Zahl zu konvertieren. Wenn der Benutzer jedoch einen nicht numerischen Wert eingibt, wird eine `ValueError` -Ausnahme ausgelöst. Der entsprechende `except` -Block wird ausgeführt, und die `ZeroDivisionError` wird nicht erreicht. Wenn die Eingabe eine Null ist, wird stattdessen der `ZeroDivisionError` -Block ausgeführt.

**Wichtiger Hinweis**: Wenn du einen allgemeineren `except` -Block (z. B. `except Exception` ) vor spezifischeren Blöcken platzierst, wird der spezifischere Block niemals erreicht. Daher ist es wichtig, spezifische Ausnahmen zuerst und dann allgemeine Ausnahmen zuletzt zu behandeln.

## 4. Mehrere Ausnahmen abfangen

Du kannst mehrere Ausnahmen in einem einzigen `try` -Block abfangen. Dies erfolgt durch das Hinzufügen mehrerer `except` -Blöcke:

```
try:
    # Code, der mehrere Ausnahmen auslösen könnte
    value = int(input("Geben Sie eine Zahl ein: "))
    result = 10 / value
except (ZeroDivisionError, ValueError) as e:
    print(f"Fehler: {e}")
```

Hier werden sowohl `ZeroDivisionError` als auch `ValueError` behandelt. Python wird die Ausnahme der Reihe nach prüfen, bis sie eine Übereinstimmung findet.

### 5. Allgemeine Ausnahmen abfangen

Du kannst auch eine allgemeine Ausnahme abfangen, die jede Ausnahme behandelt, die nicht bereits durch spezifische `except`-Blöcke behandelt wird:

```
try:
    value = int(input("Geben Sie eine Zahl ein: "))
    result = 10 / value
except ZeroDivisionError:
    print("Fehler: Division durch Null ist nicht erlaubt.")
except ValueError:
    print("Fehler: Ungültige Eingabe. Bitte geben Sie eine Zahl ein.")
except Exception as e:
    print(f"Ein unerwarteter Fehler ist aufgetreten: {e}")
```

In diesem Beispiel wird der `Exception`-Block zuletzt behandelt, sodass er nur dann ausgeführt wird, wenn keine der spezifischen Ausnahmen zutrifft.

### 6. `else` und `finally` Blöcke

Zusätzlich zu `try` und `except` kannst du auch `else` und `finally` verwenden, um die Ausführung zu steuern:

- **else**: Der `else`-Block wird ausgeführt, wenn im `try`-Block keine Ausnahme aufgetreten ist.
- **finally**: Der `finally`-Block wird immer ausgeführt, unabhängig davon, ob eine Ausnahme aufgetreten ist oder nicht. Dies ist nützlich, um Ressourcen freizugeben, z. B. um eine Datei zu schließen.

Hier ein Beispiel:

```
while True:
    try:
        value = int(input("Geben Sie eine Zahl ein: "))
        result = 10 / value
    except ZeroDivisionError:
        print("Fehler: Division durch Null ist nicht erlaubt.")
    except ValueError:
        print("Fehler: Ungültige Eingabe. Bitte geben Sie eine Zahl ein.")
    else:
        print(f"Das Ergebnis ist: {result}")
        break
```

```
    finally:
        print("Der Block wurde ausgeführt.")
```

## 7. Benutzerdefinierte Ausnahmen

Du kannst auch deine eigenen Ausnahmen erstellen, indem du von der `Exception` -Klasse erbst. Dies ist nützlich, wenn du spezifische Fehlermeldungen für deine Anwendungen definieren möchtest.

```
class MeinFehler(Exception):
    pass

def my_function(value):
    if value < 0:
        raise MeinFehler("Wert darf nicht negativ sein.")

try:
    my_function(-1)
except MeinFehler as e:
    print(f"{type(e)} Ein benutzerdefinierter Fehler ist aufgetreten: {e}")
```

In diesem Beispiel wird ein benutzerdefinierter Fehler `MeinFehler` erstellt, der ausgelöst wird, wenn ein negativer Wert übergeben wird.

## 8. Hilfe zu Exceptions

Alle Unterklassen von `Exception` anzeigen:

```
Exception.__subclasses__()
```

### Beispiele für wichtige Exceptions

1. `ZeroDivisionError`

Diese Ausnahme tritt auf, wenn du versuchst, durch Null zu dividieren.

2. `ValueError`

Diese Ausnahme tritt auf, wenn eine Funktion einen Parameter erhält, der den richtigen Typ hat, aber einen unangemessenen Wert hat. Zum Beispiel, wenn du versuchst, einen

String in eine Zahl umzuwandeln, der nicht konvertierbar ist.

3. `IndexError`

Diese Ausnahme tritt auf, wenn du versuchst, auf einen Index in einer Liste zuzugreifen, der nicht existiert.

4. `KeyError`

Diese Ausnahme tritt auf, wenn du versuchst, auf einen Schlüssel in einem Dictionary zuzugreifen, der nicht existiert.

5. `TypeError`

Diese Ausnahme tritt auf, wenn eine Operation oder Funktion auf einen falschen Typ angewendet wird. Zum Beispiel, wenn du versuchst, einen String mit einer Zahl zu addieren.

6. `FileNotFoundError`

Diese Ausnahme tritt auf, wenn du versuchst, eine Datei zu öffnen, die nicht existiert.

7. `AttributeError`

Diese Ausnahme tritt auf, wenn du versuchst, auf ein Attribut eines Objekts zuzugreifen, das nicht existiert.