Exception handling in Python

Was sind Exceptions in Python?: In Python sind Exceptions spezielle Ereignisse, die während der Programmausführung auftreten, wenn ein unerwarteter Fehler auftritt.

Im Gegensatz zu Syntaxfehlern, die verhindern, dass ein Programm überhaupt ausgeführt wird, treten Exceptions auf, wenn ein Programm bereits läuft, aber auf eine unerwartete Situation stößt.

Zum Beispiel könnte eine Division durch Null oder der Versuch, auf eine nicht existierende Datei zuzugreifen, eine Exception auslösen.

Das Ziel der Exception-Behandlung in Python besteht darin, unerwartete Fehler während der Programmausführung **rechtzeitig zu erkennen** und darauf so zu **reagieren**, dass das Programm nicht unerwartet beendet wird.

> Stattdessen soll der Fehler abgefangen und eine **geeignete Maßnahme** ergriffen werden, wie zum Beispiel das Anzeigen einer Fehlermeldung, das Bereinigen von Ressourcen, oder das Fortsetzen des Programms mit einer alternativen Vorgehensweise.
>
> Dadurch wird die Robustheit des Programms erhöht, indem es in der Lage ist, **auch bei unvorhergesehenen Problemen kontrolliert und sicher weiterzulaufen.**

Exception handling in Python

Struktur der Fehlerbehandlung: try – except – else – finally

Die Handhabung von Exceptions in Python erfolgt durch das Verwenden von vier Schlüsselwörtern: **try, except, else** und **finally**.

- Der try-Block enthält den Code, der potenziell eine Exception auslösen könnte. Dies ist der "riskante" Teil des Codes.
- Der except-Block wird ausgeführt, wenn im try-Block eine Exception auftritt. Hier wird festgelegt, wie das Programm auf den Fehler reagieren soll.
- Der else-Block wird nur ausgeführt, wenn im try-Block keine Exception auftritt. Dies ist nützlich, wenn Sie eine Aktion nur im Erfolgsfall durchführen möchten.
- Der finally-Block wird immer ausgeführt, unabhängig davon, ob eine Exception aufgetreten ist oder nicht. Dies ist nützlich für Aufräumarbeiten, wie das Schließen von Dateien oder das Freigeben von Ressourcen.

Exception handling in Python

Beispiel: Datei öffnen mit vollständiger Fehlerbehandlung

```
try:
    datei = open('meine_datei.txt', 'r')
    inhalt = datei.read()
except FileNotFoundError:
    print("Die Datei wurde nicht gefunden.")
else:
    print("Dateiinhalt:", inhalt)
finally:
    print("In Finally Block - Datei Schließen")
```

- In diesem Beispiel wird versucht, eine Datei zu öffnen und ihren Inhalt zu lesen.
- Wenn die Datei nicht existiert, wird eine FileNotFoundError-Exception ausgelöst und im **except**-Block behandelt.
- Tritt jedoch kein Fehler auf, wird der Code im **else**-Block ausgeführt, der den Inhalt der Datei ausgibt.
- Unabhängig vom Erfolg oder Misserfolg wird die Datei im **finally**-Block ausgeführt.

Exception handling in Python

Häufige eingebaute Exceptions in Python

Python bietet eine Vielzahl von eingebauten Exceptions, die häufig in der Praxis auftreten. Zu den häufigsten gehören:

- ZeroDivisionError: Diese Exception tritt auf, wenn versucht wird, eine Zahl durch Null zu teilen. Dies ist mathematisch nicht definiert und führt daher zu einem Fehler.
- FileNotFoundError: Diese Exception wird ausgelöst, wenn versucht wird, auf eine Datei zuzugreifen, die nicht existiert. Dies kann passieren, wenn der Dateipfad falsch ist oder die Datei gelöscht wurde.
- ValueError: Diese Exception tritt auf, wenn eine Operation auf einen ungültigen Wert angewendet wird. Zum Beispiel, wenn versucht wird, eine Zeichenkette in eine Zahl zu konvertieren und die Zeichenkette keine gültige Zahl darstellt.
- TypeError: Operation mit inkompatiblen Typen, z. B. "abc" + 5
- NameError: Tritt auf, wenn versucht wird, eine Variable oder ein Name zu verwenden, der nicht definiert oder im aktuellen Gültigkeitsbereich (Scope) nicht verfügbar ist – z. B. print(wert), obwohl wert nie zuvor definiert wurde.
- IndexError: Tritt auf, wenn auf einen Index zugegriffen wird, der außerhalb der Grenzen einer Liste liegt – z. B. liste[99], wenn die Liste kürzer ist.
- KeyError: Zugriff auf nicht vorhandenen Schlüssel im Dictionary: daten["xyz"]

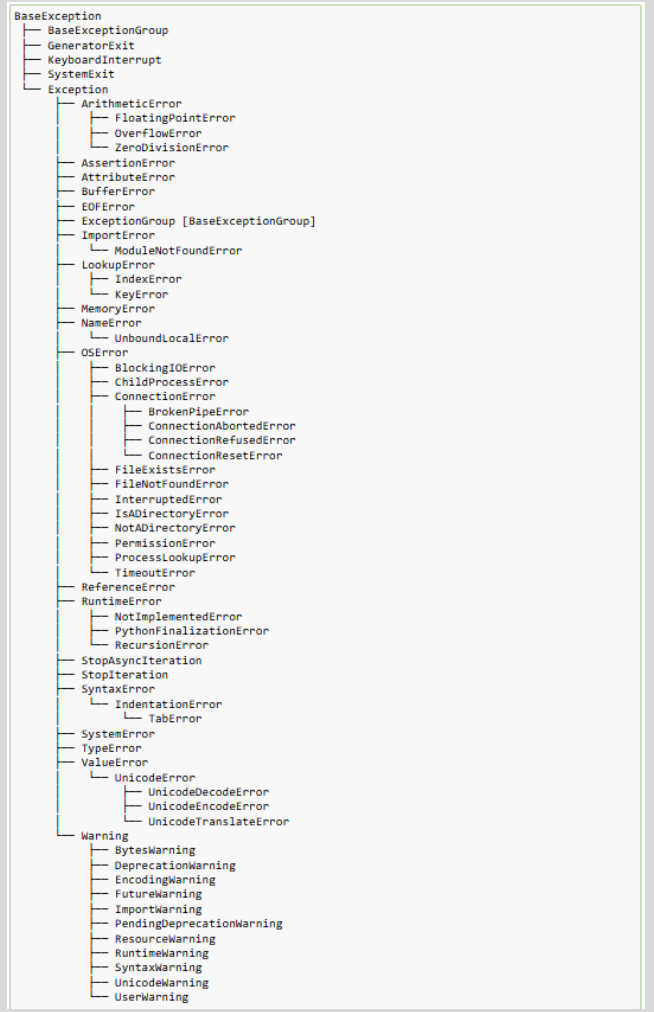
Exception handling in Python

Fehlerhierarchie in Python – Überblick

Dies ist die Hierarchie und Liste der Exceptions:
https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Bekannte Fehler wie ValueError, ZeroDivisionError oder IndexError gehören zu dieser Hierarchie.

Das Verständnis dieser Struktur erleichtert das gezielte Abfangen und Gruppieren von Fehlern im Code.



Exception handling in Python

Mehrere except-Blöcke und gruppierte Fehlerbehandlung

```
try:
    zahl = int(input("Bitte eine Zahl eingeben: "))
    ergebnis = 10 / zahl
except ValueError:
    print("Ungültige Eingabe – bitte eine Zahl eingeben.")
except ZeroDivisionError:
    print("Division durch Null ist nicht erlaubt.")
except (TypeError, KeyError):
    print("Ein anderer Fehler ist aufgetreten (Typ- oder Schlüssel-Fehler).")
else:
    print("Ergebnis:", ergebnis)
finally:
    print("Berechnung abgeschlossen.")
```

Hinweis – sofern Sie bereits mit Klassen & Vererbung vertraut sind:

- Wenn eine Elternklasse (z. B. LookupError) oder ein allgemeiner Fehler-Tupel zuerst im Code steht (z. B. except (TypeError, KeyError)), wird dieser Block bevorzugt.
- Dann wird der except-Block für eine genauere Kindklasse oder eine spezifische Fehlerklasse (wie KeyError) nicht mehr ausgeführt – selbst wenn er weiter unten steht.

- Mit mehreren except-Blöcken können verschiedene Fehlertypen gezielt behandelt werden.
- Python führt immer nur den ersten passenden except-Block aus.
- Die **Reihenfolge** der Blöcke ist wichtig – sie werden von oben nach unten geprüft.
- Mit einem Tupel wie (TypeError, KeyError) kannst du mehrere Fehlertypen zusammenfassen, wenn du sie gleich behandeln willst.
- Der else-Block wird nur ausgeführt, wenn kein Fehler im try-Block auftritt.
- Der finally-Block wird immer ausgeführt – unabhängig davon, ob ein Fehler aufgetreten ist oder nicht

Exception handling in Python

Eigene Exceptions mit Attributen und detaillierten Meldungen

In Python können Sie eigene Exceptions erstellen, indem Sie eine neue Klasse definieren, die von der Basisklasse **Exception** (oder einer ihrer **Unterklassen**) erbt. Das ist besonders nützlich, wenn Sie spezifische Fehlertypen definieren möchten, die in Ihrem Programm auftreten können.

```
class ZuWenigGuthaben(Exception):
    """ Fehler, wenn nicht genug Guthaben vorhanden ist."""
    def __init__(self, betrag, guthaben):
        self.betrag = betrag
        self.guthaben = guthaben
        super().__init__(f"{betrag}€ angefragt, aber nur {guthaben}€ verfügbar." )

        # super().__init__() übergibt eine aussagekräftige Fehlermeldung an die Exception-Klasse, sodass beim print() später ein sinnvoller Text erscheint.

def bezahlen(betrag, guthaben):
    if betrag > guthaben:
        raise ZuWenigGuthaben(betrag, guthaben)
    return guthaben - betrag

try:
    rest = bezahlen(50, 30)
except ZuWenigGuthaben as fehler:
    print("Fehler:", fehler)
# Fehler: 50€ angefragt, aber nur 30€ verfügbar.
```

In der Funktion bezahlen wird der Fehler gezielt mit **raise** ausgelöst, wenn eine bestimmte Bedingung verletzt wird – so kann man Fehler kontrolliert behandeln.

Die Klasse ZuWenigGuthaben speichert die Ursache des Fehlers (betrag und guthaben) als **Attribute**.

Über super().__init__() wird eine individuelle Fehlermeldung erzeugt.

Der except-Block behandelt die Ausnahme gezielt und gibt eine aussagekräftige Meldung aus.

Exception handling in Python

Fehler manuell auslösen mit raise

Mit dem raise-Befehl können Sie eine Exception manuell an einer bestimmten Stelle im Code auslösen. Dies ist besonders nützlich, wenn Sie eine bestimmte Bedingung validieren und daraufhin eine Exception auslösen möchten, um den Programmfluss zu steuern.

```
def check_alter(alter):
    if alter < 0:
        raise ValueError("Das Alter darf nicht negativ sein.")
    elif alter < 18:
        print("Du bist minderjährig.")
    else:
        print("Du bist volljährig.")

try:
    check_alter(-1)
except ValueError as e:
    print(f"Fehler: {e}")
```

In diesem Beispiel wird eine ValueError-Exception ausgelöst, wenn das Alter negativ ist. Dies verhindert, dass ungültige Daten weiter im Programm verwendet werden.

Best Practices bei der Fehlerbehandlung

Eine effektive Exception-Behandlung erfordert Best Practices:

- **Spezifische Exceptions abfangen**: Fangen Sie nur die Exceptions ab, die Sie wirklich erwarten und behandeln möchten. Das Abfangen von allgemeinen Exceptions wie Exception kann es schwieriger machen, tatsächliche Fehler zu erkennen und zu beheben.

- **Benutzerdefinierte Fehlermeldungen verwenden**: Stellen Sie sicher, dass die Fehlermeldungen klar und informativ sind. Dies erleichtert die Fehlersuche und das Debuggen.

- **Den Programmfluss nicht durch zu viele try-except-Blöcke unterbrechen**: Versuchen Sie, Ihren Code so zu strukturieren, dass try-except-Blöcke sinnvoll eingesetzt werden und den Programmfluss nicht unnötig komplizieren.

- **Logik in finally-Blöcken gut überdenken**: Verwenden Sie den finally-Block für kritische Bereinigungsaktionen, aber seien Sie vorsichtig, dass keine neuen Exceptions im finally-Block selbst ausgelöst werden.

Exception handling in Python

Zusammenfassung

Thema	Inhalt kurz erklärt
try, except, else, finally	Programmfehler gezielt behandeln und Ablauf absichern.
Spezifische Fehlerbehandlung	Bestimmte Fehlertypen wie ValueError oder ZeroDivisionError gezielt abfangen.
Mehrere except-Blöcke	Unterschiedliche Fehler je nach Ursache unterschiedlich behandeln.
Gruppierte Fehlertypen	Mehrere Fehlertypen gemeinsam mit (Fehler1, Fehler2) abfangen.
Eigene Fehlerklassen	Eigene Exceptions mit Namen und Attributen definieren.
raise-Befehl	Eigene Fehler aktiv auslösen – z. B. bei ungültiger Eingabe.
Fehlerhierarchie	Struktur und Aufbau von Python-Fehlertypen verstehen (Vererbung von Exception).

Exception handling in Python

Mehr erkunden :

https://www.datacamp.com/tutorial/exception-handling-python

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

https://www.geeksforgeeks.org/python-exception-handling/

https://www.programiz.com/python-programming/exceptions

https://www.w3schools.com/python/python_try_except.asp