

Errors und Exceptions

Ein Deep Dive zu Fehlern

Fehler und Ausnahmen

Während Sie ein Programm entwickeln und testen, treten häufig Fehler auf. Das ist normal und auch wichtig für den Lernprozess. Es gibt drei Arten von Fehlern, **Syntaxfehler**, **Laufzeitfehler** und **logische Fehler**:

Fehler und Ausnahmen

- Syntaxfehler bemerken Sie spätestens beim Start eines Programms
- Laufzeitfehler, also Fehler zur Laufzeit des Programms, die einen Programmabsturz zur Folge haben, können Sie mit einem **try-except-Block** behandeln


Fehler und Ausnahmen

- Logische Fehler treten auf, wenn das Programm vollständig abläuft, aber **nicht die erwarteten Ergebnisse liefert**
 - Hier hat der Entwickler den Ablauf nicht richtig durchdacht
 - Diese Fehler sind erfahrungsgemäß am schwersten zu finden,
 - Dabei bietet das Debugging eine gute Hilfestellung.

Syntaxfehler

Spätestens beim Start eines Programms macht Python auf Syntaxfehler aufmerksam. Die Programmiererin erhält eine Meldung und einen Hinweis auf die Fehlerstelle. Das Programm wird nicht weiter ausgeführt.

Beispiel:



```
1 x = 12
2 if x > 10
3 print(x
```

Laufzeitfehler

Laufzeitfehler treten auf, wenn das Programm versucht, eine unzulässige Operation durchzuführen, zum Beispiel eine Division durch 0 oder das Öffnen einer nicht vorhandenen Datei

Laufzeitfehler

Logische Fehler treten auf, wenn eine Anwendung zwar ohne Syntaxfehler übersetzt und ohne Laufzeitfehler ausgeführt wird, aber nicht das geplante Ergebnis liefert. Ursache hierfür ist ein Fehler in der Programmlogik.

```
1 # 15-single-step.py
2
3 def sumfunc(a, b):
4     c = a + b
5     return c
6
7
8 for i in range(5):
9     erg = sumfunc(10, i)
10    print(erg)
```

Fehlertypen

Im folgenden Programm werden unterschiedliche Arten von Fehlern, die zu Ausnahmen führen, spezifisch abgefangen.

Damit erfährt die Benutzerin mehr über den Fehler, und es wird eine komfortablere Programmbedienung ermöglicht.

```
1 # 15-error-types.py
2
3 while True:
4     try:
5         num = float(input("A positive number: "))
6         if num == 0:
7             raise RuntimeError("Number equals 0")
8         if num < 0:
9             raise RuntimeError("Number to small")
10        kw = 1.0 / num
11        break
12    except ValueError:
13        print("Error: No nu,ber")
14    except ZeroDivisionError:
15        print("Error: Number 0 found")
16    except RuntimeError as e:
17        print("Error:", e)
```


Fehlertypen

Im folgenden Programm werden unterschiedliche Arten von Fehlern, die zu Ausnahmen führen, spezifisch abgefangen.

Damit erfährt die Benutzerin mehr über den Fehler, und es wird eine komfortablere Programmbedienung ermöglicht.

```
1 # 15-error-types.py
2
3 while True:
4     try:
5         num = float(input("A positive number: "))
6         if num == 0:
7             raise RuntimeError("Number equals 0")
8         if num < 0:
9             raise RuntimeError("Number to small")
10        kw = 1.0 / num
11        break
12    except ValueError:
13        print("Error: No nu,ber")
14    except ZeroDivisionError:
15        print("Error: Number 0 found")
16    except RuntimeError as e:
17        print("Error:", e)
```

Fehlertypen

Im folgenden Programm werden unterschiedliche Arten von Fehlern, die zu Ausnahmen führen, spezifisch abgefangen.

Damit erfährt die Benutzerin mehr über den Fehler, und es wird eine komfortablere Programmbedienung ermöglicht.

```
1 # 15-error-types.py
2
3 while True:
4     try:
5         num = float(input("A positive number: "))
6         if num == 0:
7             raise RuntimeError("Number equals 0")
8         if num < 0:
9             raise RuntimeError("Number to small")
10        kw = 1.0 / num
11        break
12    except ValueError:
13        print("Error: No nu,ber")
14    except ZeroDivisionError:
15        print("Error: Number 0 found")
16    except RuntimeError as e:
17        print("Error:", e)
```

Fehlertypen

Im folgenden Programm werden unterschiedliche Arten von Fehlern, die zu Ausnahmen führen, spezifisch abgefangen.

Damit erfährt die Benutzerin mehr über den Fehler, und es wird eine komfortablere Programmbedienung ermöglicht.

```
1 # 15-error-types.py
2
3 while True:
4     try:
5         num = float(input("A positive number: "))
6         if num == 0:
7             raise RuntimeError("Number equals 0")
8         if num < 0:
9             raise RuntimeError("Number to small")
10        kw = 1.0 / num
11        break
12    except ValueError:
13        print("Error: No nu,ber")
14    except ZeroDivisionError:
15        print("Error: Number 0 found")
16    except RuntimeError as e:
17        print("Error:", e)
```

Übung: Errors

Schauen Sie sich kurz folgenden Link an:

<https://www.tutorialsteacher.com/python/error-types-in-python>



```
1 dc = {"Peter": 31, "Julia": 28, "Werner": 35}
2 print("Dictionary:", dc)
3 try:
4     print(dc[23])
5 except ???
```

Versuchen Sie danach den Code mit dem treffenden Error zu ergänzen und schreiben Sie eine kurze Nachricht, die passt.

Lösung: Errors



```
1 dc = {"Peter": 31, "Julia": 28, "Werner": 35}
2 print("Dictionary:", dc)
3 try:
4     print(dc[23])
5 except KeyError:
6     print("Key was not found")
```

Lösung: Eigenes Modul



```
1 # reader.py
2
3 def read_int():
4     """Reads an integer from the input and returns the entered number as int"""
5     return int(input("Please enter a number (integer-only): "))
6
7
8 def read_float():
9     """Reads a number from the input and returns the entered number as float"""
10    return float(input("Please enter a number: "))
```

Übung: Eigenes Modul

Erstellen Sie einen neuen Skript **calculator.py** mit gleichnamiger Funktion **def calculator**, das keine Parameter aufnimmt, sondern den Benutzer auffordert, zwei ganze Zahlen einzugeben, und die Summe beider Zahlen zurückgibt.

Verwenden Sie eine geeignete Funktion aus dem `read_input`-Modul, um die Integer-Zahlen einzulesen.

Lösung: Eigenes Modul

```
1 # calculator.py
2
3 from reader import read_float
4
5
6 def calculator():
7     number_1 = read_float()
8     number_2 = read_float()
9
10     return number_1 + number_2
11
12
13 calculator()
```


Übung: Eigenes Modul

Versuchen Sie nun den `calculator.py` zu erweitern und weitere Hilfsfunktionen aus `basicfunc.py` zu importieren.

Aussage	Bedeutung
<code>import circle</code>	Alle Funktionen können mit <code>circle.function()</code> aufgerufen werden, z. B. <code>circle.area(radius=2)</code>
<code>from circle import area</code>	Nur die Funktion <code>area</code> wird aus dem Modul <code>circle</code> importiert. Die importierte Funktion kann dann direkt verwendet werden, z. B. <code>area(2)</code>

Ende

Das war alles für dieses Kapitel
