Python Debugging

D. Leeuw

10 juni 2025

1 Over dit Document

1.1 Leerdoelen

Na het bestuderen van dit document heeft de lezer basis kennis van:

- syntax errors en wat dat zijn
- runtime errors en wat dat zijn
- het lezen de errors die door Python gegeven kunnen worden
- hoe je met try/except runtime errors kan afvangen

1.2 Voorkennis

Van de lezer wordt verwacht dat deze weet hoe:

- variabelen gebruikt worden in Python
- for/while loops werken in Python
- functies gemaakt worden in Python

2 Bugs, errors en exceptions

In software kunnen er allerlei zaken fout gaan. Er kan een fout in de software zitten, er kan een conditie ontstaan die niet voorzien is of een gebruiker kan verkeerde data invoeren. Een programmeerfout heet een bug en deze kan hersteld worden door de code aan te passen. Het probleem is vaak dat we de fout niet zien en dat deze pas naar voren komt als veel verschillende gebruikers de software getest hebben in allerlei verschillende omstandigheden. Gebruikers hebben dan de neiging om te roepen “Hij doet het niet” inplaats van dat ze kunnen aangeven waar in de code het fout gegaan is. Binnen de software zou het dus fijn zijn als we kunnen detecteren dat er iets fout gegaan is en dat de software aangeeft wat er fout gegaan is.

Binnen het vakgebied van software development zijn er verschillende termen voor fouten die kunnen optreden. De meest algemene term is de `bug`. Een bug is een fout in de logica of structuur van de software. De bug zorgt ervoor dat het programma niet doet wat je zou willen of het kan een `error` veroorzaken. Een `error` is een algemene benaming voor iets dat fout gaat in de software. Dit kan een syntax error of een runtime error zijn. Een

error zorgt er meestal voor dat je programma crashed. Python zal een error-melding geven, die je hopelijk helpt om het probleem op te lossen. De syntax error is een door de programmeur gemaakte fout. Tot slot is er nog de `exception`. De exception is een specifiek soort error die optreedt tijdens het uitvoeren van een programma, namelijk de runtime error.

We hebben dus een syntax error die onstaat doordat de programmeur zich niet aan de syntax van Python houdt:

```
if x = 5:
```

Voor een vergelijking gebruiken we in Python == en niet =. Dit is een syntax-error en die moeten wij als programmeur oplossen.

Runtime errors kunnen ontstaan tijdens het draaien van het programma. Ze zijn dus niet syntactisch fout, maar zijn het gevolg van de werking van het programma. Je zou bijvoorbeeld een bestand kunnen opvragen dat er niet is, of een getal delen door 0. Dat zijn allemaal zaken die optreden tijdens de werking van het programma, die je niet van te voren kon weten. Je had er natuurlijk wel op kunnen controleren. Je had kunnen kijken of het bestand bestond voordat je het wilde opvragen en je had kunnen kijken of de deler 0 is, voordat je de deling uitvoerde. Deze fouten worden exceptions genoemd en exceptions kunnen we afvangen.

2.1 Errors leren lezen

Errors in Python kunnen in het begin overweldigend zijn omdat er vrij veel informatie op het scherm wordt afgebeeld. Met enige training is er al snel duidelijkheid in de brei te ontdekken. Als voorbeeld nemen we deze code:

```
for a in list
   print("Hallo {a}")
```

In deze code zitten verschillende fouten. Python stopt (crashed) bij de eerste fout die er waargenomen wordt.

Als we de bovenstaande code runnen dan is de eerste error die we krijgen:

```
  File "/home/Boeken/Programmeren/Python/code/error.py", line 1
    for a in list
                 ^
SyntaxError: expected ':'
```

We lezen de error-melding van onder naar boven. De melding is een SyntaxError, dus wij hebben iets niet goed gedaan. Met de caret (^) geeft Python aan waar er een syntactische fout is en als het mogelijk is geeft Python ook aan wat hij verwachtte (expected). In ons voorbeeld verwachtte Python een : na `list`.

Nadat we deze fout hersteld hebben en we de code opnieuw runnen komt Python met de melding:

```
Traceback (most recent call last):
  File "/home/Boeken/Programmeren/Python/code/error.py", line 1, in
    <module>
    for a in list:
TypeError: 'type' object is not iterable
```

Hier zien we een TypeError. Dat wil zeggen dat het type niet klopt. Python geeft aan dat het object niet iterable (doorloopbaar) is. Het moet dus gaan om de `list`, want die willen we doorlopen. Als we naar de code kijken dan klopt dat ook. `list` is nergens gedefinieerd. Zo zie je dat het soms niet meteen duidelijk is wat er fout gegaan is, maar dat je even door moet denken om de fout te vinden.

We voegen aan het begin van het script de regel toe:

```
list = [Achmed, Tom, Quinten, Mohamed]
```

We krijgen nu een nieuwe error melding:

```
Traceback (most recent call last):
  File "/home/Boeken/Programmeren/Python/code/error.py", line 1, in
    <module>
    list = [Ahmed, Tom, Quinten, Mohamed]
            ^^^^^
NameError: name 'Ahmed' is not defined
```

Python geeft aan dat Ahmed "not defined" is. Hij ziet Achemd dus als variabele en niet als string. Dat komt omdat wij de quotes vergeten zijn. De regel had er natuurlijk zo uit moeten zien:

```
list = ["Ahmed", "Tom", "Quinten", "Mohamed"]
```

Nadat we dit verbeterd hebben draait de code zonder fouten en toch is de output niet correct. We zijn bij de `print` functie de `f` optie vergeten. Je hebt kennelijk foutloze code zonder dat het gewenste resultaat uit de code komt.

Pas nadat we de `print` veranderd hebben in:

```
   print(f"Hallo {a}")
```

doet het script wat het moet doen.

3 Syntax Errors

Syntax errors zijn waarschijnlijk de meest voorkomende errors die je tegen komt als je een programmeertaal aan het leren bent. Een syntax error be-

tekent dat je code geschreven hebt die zich niet aan de regels van Python houdt. Vaak zijn het kleine simpele dingen als het vergeten van een : of niet juist inspringen.

Een syntax error wordt in Python meestal duidelijk aangegeven als je de code probeert te runnen:

```
  File "/home/Boeken/Programmeren/Python/code/syntaxerror.py", line 1
    while True print('Hello world')
               ^^^^^
SyntaxError: invalid syntax
```

Met de carets (circumflex) (^) geeft Python aan waar vermoedelijk de fout zit. In dit geval moet er natuurlijk na de `True` een : komen en dus geeft Python aan dat we hier te maken hebben met een syntax error.

Omdat syntax errors fouten zijn die wij zelf gemaakt hebben moeten wij ze ook oplossen door de code aan te passen zodat deze wel voldoet aan de regels van Python.

Alle syntax errors kunnen we uit onze code halen door deze te testen (runnen). Deze fouten zouden er dus allemaal uit kunnen zijn voordat we het script daadwerkelijk in "productie" gaan gebruiken.

4 Runtime errors

Exceptions zijn runtime errors ontstaan doordat er iets in de software gebeurd dat we niet hebben voorzien. Deze fouten worden pas gevonden door de software te gebruiken. Runtime errors hoeven geen fouten te zijn waarop de software crashed. We kunnen ze afvangen met een zogenaamd `try` / `except` block. Dat ziet er zo uit:

```
try:
    resultaat = 10 / 0
except ZeroDivisionError:
    print("Delen door nul is niet toegestaan.")
```

Onder het kopje `try` wordt een actie uitgevoerd en bij het kopje `except` wordt de fout afgehandeld. De fout afhandeling kan een simpele print zijn zoals in het voorgaande voorbeeld waarbij je de gebruiker vertelt wat er fout gegaan is.

Natuurlijk zou je zelf nooit door 0 delen, maar het kan zijn dat er een variabele is die in sommige situaties 0 wordt en dat daarna deze fout ontstaat. Zeker als een programma veel gebruik maakt van gebruikers invoer of data van anderen die via een bestand wordt ingelezen, dan kunnen er soms waarden in variabelen terecht komen die je niet zou verwachten. Zo kan er dan een fout situatie ontstaan die je vooraf niet bedacht had.

De melding "Je kunt niet delen door nul" is ook geen zinnige meldig. We zouden hier meer informatie naar de gebruiker terug kunnen geven. Bijvoorbeeld om welke variabele het gaat, of wat de functie van de deling is. Het goed schrijven van een error-melding is vaak nog een hele kunst, omdat je ook niet altijd weet de technische kennis is van de gebruiker.

Tijdens de werking van een programma kunnen er verschillende fouten ontstaan die afgevangen kunnen worden. Voor een overzicht van de mogelijke fouten en hun naam (except) is er een lijst van Built in exceptions op W3Schools: https://www.w3schools.com/python/python_ref_exceptions.asp. Lees deze lijst door zodat je weet welke fouten je met try/except standaard kunt afvangen. Niet alle exceptions zullen meteen duidelijk zijn, maar een aantal zou je direct kunnen gaan gebruiken.

Met try/except kunnen we bijvoorbeeld de input van een gebruiker controleren:

```
while True:
    try:
        x = int(input("Geef een nummer op: "))
        break
    except ValueError:
        print("Dat was geen nummer, probeer het nog een keer.")

print(f"De ingevoerde waarde is: {x}")
```

Dit werkt zo:

1. De `while`-loop loopt door tot de `break` uitgevoerd kan worden. Daarna krijgt de gebruiker de melding welke waarde er opgegeven is.

2. De `try` vraagt aan de gebruiker om een getal in te voeren en dit getal moet een integer (int) zijn. Als dit niet zo is dan is er een error, een `ValueError`. Een error breekt de `try` meteen af zodat deze nooit bij de `break` komt.

3. De `except` vangt de error af en geeft een melding naar de gebruiker met wat deze fout gedaan heeft. De `while` zorgt ervoor dat we meteen weer bij de `try` terecht komen.

We kunnen de twee voorbeelden ook combineren, waarbij we één `try` hebben met meerdere exceptions, namelijk: `ValueError` en `ZeroDivisionError`:

```
try:
    x = int(input("Geef een nummer op: "))
    resultaat = 10 / x
    print("Resultaat is: ", resultaat)
except ValueError:
```

```
    print("Dat was geen nummer, probeer het nog een keer.")
except ZeroDivisionError:
    print("Delen door nul is niet toegestaan.")
```

Tot slot is er nog de algemene exception genaamd `Exception`. Die kunnen we gebruiken als een laatste red middel:

```
try:
    x = int(input("Geef een nummer op: "))
    resultaat = 10 / x
    print("Resultaat is: ", resultaat)
except ValueError:
    print("Dat was geen nummer, probeer het nog een keer.")
except ZeroDivisionError:
    print("Delen door nul is niet toegestaan.")
except Exception as error:
    print("Er is iets fout gegaan. Het systeem meldde: ", error)
```

Hier geven de door Python gemeldde exception in de `except` door als waarde in de variabele `error`. En deze `error` variabele kunnen we dan weer op het scherm afbeelden in de hoop dat de gebruiker of de programmeur er iets mee kan.

4.1 Module exceptions

Modules kunnen hun eigen exceptions hebben. Een exception uit een module heeft de module naam als voorvoegsel:

```
#----------------#
# Import modules #
#----------------#
try:
    import requests
except ModuleNotFoundError:
    exit("De 'requests' module is niet aanwezig op dit systeem")

#------------------------------#
# Variabelen voor ons programma #
#------------------------------#
# Tegen welke host en port gaan we testen
url = "http://127.0.0.1"
# Header met domain name
headers = {
"Host: localhost"
}
# Timeout bij geen reactie
timeout = 5
```

```
#----------------------------------#
# Test de webserver op domain name #
#----------------------------------#
try:
    response = requests.get(url, headers=headers, timeout=timeout)
    if response.status_code == 200:
        print(f"De server op {ip} reageert op domein '{domain}' (HTTP
    {response.status_code}).")
    else:
        print(f"De server op {ip} reageert, maar gaf status
    {response.status_code} voor domein '{domain}'.")
except requests.ConnectionError:
    print(f"Geen verbinding mogelijk met {ip}.")
except requests.Timeout:
    print(f"Timeout bij het verbinden met {ip}.")
except requests.RequestException as e:
    print(f"Er trad een fout op: {e}")
```

Hier zien we dat de requests module zeker drie exceptions kent: `ConnectionError`, `Timeout` en de generieke error `RequestException`.

Index