Python les-materialen

# Fouten (Errors) en afhandeling van uitzonderingen (exceptions)

In deze lezing zullen we leren over het afhandelen van fouten en uitzonderingen in Python. Je bent op dit punt in de cursus zeker al fouten tegengekomen. Bijvoorbeeld:

print('Hello)

File "<ipython-input-1-db8c9988558c>", line 1  
 print('Hello)  
 ^  
SyntaxError: EOL while scanning string literal

Merk op hoe we een SyntaxError krijgen, met de verdere beschrijving dat het een EOL (End of Line Error) was tijdens het letterlijk scannen van de string. Dit is specifiek genoeg voor ons om te zien dat we een enkel citaat aan het einde van de regel zijn vergeten. Als u deze verschillende fouttypen begrijpt, kunt u veel sneller fouten in uw code opsporen.

Dit type fout en beschrijving staat bekend als een uitzondering. Zelfs als een instructie of expressie syntactisch correct is, kan deze een fout veroorzaken wanneer wordt geprobeerd deze uit te voeren. Fouten die tijdens de uitvoering worden gedetecteerd, worden uitzonderingen genoemd en zijn niet onvoorwaardelijk fataal.

U kunt de volledige lijst met ingebouwde uitzonderingen [hier](https://docs.python.org/3/library/exceptions.html) bekijken. Laten we nu leren hoe we met fouten en uitzonderingen in onze eigen code kunnen omgaan.

## try and except

De basisterminologie en syntaxis die worden gebruikt om fouten in Python af te handelen, zijn de instructies try en except. De code die een uitzondering kan veroorzaken, wordt in het blok try geplaatst en de afhandeling van de uitzondering wordt vervolgens geïmplementeerd in het codeblok behalve. De syntaxis volgt:

try:  
 You do your operations here...  
 ...  
except ExceptionI:  
 If there is ExceptionI, then execute this block.  
except ExceptionII:  
 If there is ExceptionII, then execute this block.  
 ...  
else:  
 If there is no exception then execute this block.

We kunnen ook gewoon op elke uitzondering controleren door gewoon behalve: te gebruiken. Laten we, om dit alles beter te begrijpen, een voorbeeld bekijken: We zullen wat code bekijken die een bestand opent en schrijft:

try:  
 f = open('testfile','w')  
 f.write('Test write this')  
except IOError:  
 # This will only check for an IOError exception and then execute this print statement  
 print("Error: Could not find file or read data")  
else:  
 print("Content written successfully")  
 f.close()

Content written successfully

Laten we nu eens kijken wat er zou gebeuren als we geen schrijfrechten hadden (alleen openen met ‘r’):

try:  
 f = open('testfile','r')  
 f.write('Test write this')  
except IOError:  
 # This will only check for an IOError exception and then execute this print statement  
 print("Error: Could not find file or read data")  
else:  
 print("Content written successfully")  
 f.close()

Error: Could not find file or read data

Geweldig! Merk op hoe we alleen een verklaring hebben afgedrukt! De code liep nog steeds en we konden doorgaan met het uitvoeren van acties en het uitvoeren van codeblokken. Dit is uitermate handig wanneer u rekening moet houden met mogelijke invoerfouten in uw code. U kunt voorbereid zijn op de fout en code blijven uitvoeren, in plaats van dat uw code gewoon breekt, zoals we hierboven hebben gezien.

We hadden ook gewoon behalve: kunnen zeggen als we niet zeker wisten welke uitzondering zou optreden. Bijvoorbeeld:

try:  
 f = open('testfile','r')  
 f.write('Test write this')  
except:  
 # This will check for any exception and then execute this print statement  
 print("Error: Could not find file or read data")  
else:  
 print("Content written successfully")  
 f.close()

Error: Could not find file or read data

Geweldig! Nu hoeven we die lijst met uitzonderingstypen niet echt te onthouden! Wat als we code blijven willen uitvoeren nadat de uitzondering is opgetreden? Dit is waar finally binnenkomt.

## finally

Het finally -codeblok wordt altijd uitgevoerd, ongeacht of er een uitzondering was in het codeblok try. De syntaxis is:

try:  
 Code block here  
 ...  
 Due to any exception, this code may be skipped!  
finally:  
 This code block would always be executed.

For example:

try:  
 f = open("testfile", "w")  
 f.write("Test write statement")  
 f.close()  
finally:  
 print("Always execute finally code blocks")

Always execute finally code blocks

We kunnen dit gebruiken in combinatie met except. Laten we een nieuw voorbeeld bekijken waarin rekening wordt gehouden met een gebruiker die de verkeerde invoer geeft:

def askint():  
 try:  
 val = int(input("Please enter an integer: "))  
 except:  
 print("Looks like you did not enter an integer!")  
  
 finally:  
 print("Finally, I executed!")  
 print(val)

askint()

Please enter an integer: 5  
Finally, I executed!  
5

askint()

Please enter an integer: five  
Looks like you did not enter an integer!  
Finally, I executed!  
  
  
  
---------------------------------------------------------------------------  
  
UnboundLocalError Traceback (most recent call last)  
  
<ipython-input-8-cc291aa76c10> in <module>()  
----> 1 askint()  
  
  
<ipython-input-6-c97dd1c75d24> in askint()  
 7 finally:  
 8 print("Finally, I executed!")  
----> 9 print(val)  
  
  
UnboundLocalError: local variable 'val' referenced before assignment

Merk op hoe we een fout kregen bij het afdrukken van val (omdat het nooit correct was toegewezen). Laten we dit oplossen door de gebruiker te vragen en te controleren of het invoertype een geheel getal is:

def askint():  
 try:  
 val = int(input("Please enter an integer: "))  
 except:  
 print("Looks like you did not enter an integer!")  
 val = int(input("Try again-Please enter an integer: "))  
 finally:  
 print("Finally, I executed!")  
 print(val)

askint()

Please enter an integer: five  
Looks like you did not enter an integer!  
Try again-Please enter an integer: four  
Finally, I executed!  
  
  
  
---------------------------------------------------------------------------  
  
ValueError Traceback (most recent call last)  
  
<ipython-input-9-92b5f751eb01> in askint()  
 2 try:  
----> 3 val = int(input("Please enter an integer: "))  
 4 except:  
  
  
ValueError: invalid literal for int() with base 10: 'five'  
  
  
During handling of the above exception, another exception occurred:  
  
  
ValueError Traceback (most recent call last)  
  
<ipython-input-10-cc291aa76c10> in <module>()  
----> 1 askint()  
  
  
<ipython-input-9-92b5f751eb01> in askint()  
 4 except:  
 5 print("Looks like you did not enter an integer!")  
----> 6 val = int(input("Try again-Please enter an integer: "))  
 7 finally:  
 8 print("Finally, I executed!")  
  
  
ValueError: invalid literal for int() with base 10: 'four'

Hmmm… dat deed maar één controle. Hoe kunnen we continu blijven checken? We kunnen wel een while-lus gebruiken!

def askint():  
 while True:  
 try:  
 val = int(input("Please enter an integer: "))  
 except:  
 print("Looks like you did not enter an integer!")  
 continue  
 else:  
 print("Yep that's an integer!")  
 break  
 finally:  
 print("Finally, I executed!")  
 print(val)

askint()

Please enter an integer: five  
Looks like you did not enter an integer!  
Finally, I executed!  
Please enter an integer: four  
Looks like you did not enter an integer!  
Finally, I executed!  
Please enter an integer: 3  
Yep that's an integer!  
Finally, I executed!

Dus waarom printte onze functie “Finally, I executed!” na elke proef (trial), maar het heeft nooit ‘val’ zelf afgedrukt? Dit komt omdat met een try/except/finally-clausule alle continue- of break-statements worden gereserveerd totdat *na* de try-clausule is voltooid.

Dit betekent dat hoewel een succesvolle invoer van **3** ons naar het else:-blok bracht en er een break-statement werd gegenereerd, de try-clausule doorging tot < code>finally: voordat de while-lus wordt doorbroken. En aangezien print(val) zich buiten de try-clausule bevond, verhinderde (prevent) de break-instructie dat het werd uitgevoerd.

Laten we nog een laatste aanpassing maken:

def askint():  
 while True:  
 try:  
 val = int(input("Please enter an integer: "))  
 except:  
 print("Looks like you did not enter an integer!")  
 continue  
 else:  
 print("Yep that's an integer!")  
 print(val)  
 break  
 finally:  
 print("Finally, I executed!")

askint()

Please enter an integer: six  
Looks like you did not enter an integer!  
Finally, I executed!  
Please enter an integer: 6  
Yep that's an integer!  
6  
Finally, I executed!

**Geweldig! Nu weet je hoe je omgaat met fouten en uitzonderingen in Python met de notatie try, except, else en finally notaties!**