Exceptions in Python

Bas Terwijn

February 8, 2026

1 Introductie

Dit document behandelt het gebruik exceptions voor foutafhandeling.

1.1 Doelen

Bekend raken met:

- exceptions

2 Exceptions

Documentatie over exceptions in Python is te vinden op de Errors and Exceptions pagina van de officiële Python tutorial.

2.1 Grades

In directory `assignments` vindt u programma `grades1.py` dat cijfers in kan lezen uit "comma separated values" (CSV) file `grades_good.csv` :

assignments/grades_good.csv

```
name    , g1   , g2   , g3
Alice   , 7.5  , 8.3  , 6.2
Bob     , 4.5  , 6.2  , 5.1
Carol   , 6.2  , 3.2  , 4.6
Dave    , 9.0  , 5.1  , 4.1
Eve     , 4.6  , 7.4  , 5.3
```

Het berekent en print welke studenten slagen met een gemiddeld cijfer van 5.5 of hoger:

```
$ python grades1.py grades_good.csv
grades: {'Alice': [7.5, 8.3, 6.2], 'Bob': [4.5, 6.2, 5.1], 'Carol': [6.2, 3.2, 4.6],
         'Dave': [9.0, 5.1, 4.1], 'Eve': [4.6, 7.4, 5.3]}
averages: {'Alice': 7.333333333333333, 'Bob': 5.266666666666667, 'Carol': 4.666666666666667,
           'Dave': 6.066666666666666, 'Eve': 5.766666666666667}
passing students: ['Alice', 'Dave', 'Eve']
```

Dit programma werkt goed, maar heeft nog geen fout foutafhandeling voor als de input file fouten bevat. Als voorbeeld van een file met fouten gebruiken we nu `grades_bad.csv` met 'XXXX' waarden waar we float waarden verwachten:

assignments/grades_bad.csv

```
name    , g1   , g2   , g3
Alice   , 7.5  , 8.3  , 6.2
Bob     , XXXX , 6.2  , 5.1
Carol   , 6.2  , 3.2
Dave    , 9.0  , 5.1  , XXXX
Eve     , XXXX , XXXX
```

Als we deze file nu proberen in te lezen stop het programma met een ValueError:

```
$ python grades1.py grades_bad.csv
...
File "grades1.py", line 20, in get_grades
    grade_float = float(grade)
ValueError: could not convert string to float: 'XXXX'
```

We gaan nu op verschillende manieren foutafhandeling toevoegen om deze errors op te lossen met als startpunt drie keer hetzelfde programma:

- grades1.py
- grades2.py
- grades3.py

2.2 Opdracht `grades1.py`

Gebruik een try-except block om ValueErrors af te vangen en gebruik in dat geval als cijfer dan de waarde `0.0` zodat dit de output wordt:

```
$ python grades1.py grades_good.csv
grades: {'Alice': [7.5, 8.3, 6.2], 'Bob': [0.0, 6.2, 5.1], 'Carol': [6.2, 3.2],
         'Dave': [9.0, 5.1, 0.0], 'Eve': [0.0, 0.0]}
averages: {'Alice': 7.333333333333333, 'Bob': 3.766666666666667, 'Carol': 4.7,
           'Dave': 4.7, 'Eve': 0.0}
passing students: ['Alice']
```

2.3 Opdracht `grades2.py`

Gebruik een try-except block om ValueErrors af te vangen en geef de student in dat geval nu geen cijfer. Dit resulteert dan wel in een exception bij het berekenen van het gemiddelde cijfer van Eve. Gebruik daar dan ook een try-except block om geen gemiddelde voor Eve te berekenen. Dit zou dan de output moeten worden:

```
$ python grades2.py grades_good.csv
grades: {'Alice': [7.5, 8.3, 6.2], 'Bob': [6.2, 5.1], 'Carol': [6.2, 3.2],
         'Dave': [9.0, 5.1], 'Eve': []}
averages: {'Alice': 7.333333333333333, 'Bob': 5.65, 'Carol': 4.7,
           'Dave': 7.05}
passing students: ['Alice', 'Bob', 'Dave']
```

2.4 Opdracht `grades3.py`

Soms is het nuttig om een eigen exception class te definiëren zodat je specifieke fouten beter kunt afhandelen. Voeg deze eigen GradeError exception class toe:

```
class GradeError(Exception):
    def __init__(self, message):
        super().__init__(message)  # init Exception class with message
```

Gebruik vervolgens dit try-except block voor alle code in de `print_passing_students()` functie om daar alle ValueErrors af te vangen, maar niet de FileNotFoundErrors. Raise in het geval van een ValueError (en in de toekomst mogelijke andere fouten die met cijfers te maken hebben) dan een GradeError exception met deze code:

```
try:
    # all code in print_passing_students() function
except ValueError as e:
    raise GradeError("Error reading grade.")
```

Andere code die `grade3.py` importeert en de `print_passing_students()` functie aanroept, kan nu onze eigen GradeError exception afvangen wat nu meer specifiek over grades gaat. Dit kan de leesbaarheid van foutafhandeling vergroten. Een voobeeld hiervan zien we hier waar we de verschillende fouten ieder op een eigen manier zouden kunnen afhandelen:

assignments/import_grades3.py

```
import grades3  # imports all code in 'grade3.py'

def main(filename: str):
    try:
        grades3.print_passing_students(filename)
    except grades3.GradeError as e:
        print("Caught GradeError:", e)  # handle GradeError
    except FileNotFoundError as e:
        print("Caught FileNotFoundError:", e)  # handle FileNotFoundError

main('grades_good.csv')
main('grades_bad.csv')
main('non_existent_file.csv')
```