

Les 9 - Fouten

[« Vorige les](#)[» Volgende les](#)

Huiswerk

Programmeren voor Beginners - Les 9

Je hebt je huiswerk nog niet ingediend.

[Huiswerk inzenden](#)

Je hebt 1 vraagmoment. 1 moment over.

[Ga naar stel een vraag](#)[!\[\]\(d3102649f02e825ddb76dc3de0190154_img.jpg\) Markeer als deelgenomen aan deze les](#)

Inhoudsopgave

9.1	Inleiding
9.2	Wat is een softwarefout?
9.3	Verschillende soorten fouten
9.4	Debug-strategieën
9.5	Overige hulpmiddelen
9.6	Project 'IJssalon'
9.7	Samenvatting
9.8	Vraagmoment
9.9	Oefenopgaven
9.10	Huiswerkopgaven

9.1 Inleiding

We beginnen deze les met enkele tegeltjeswijscheden:



Zo zouden we nog veel meer spreken kunnen noemen. De betekenis komt in veel gevallen op hetzelfde neer: fouten maken is onvermijdelijk. Dat geldt ook voor programmeren: als u gaat programmeren, zult u fouten maken, of u nu een beginner bent of al de nodige ervaring heeft.

Maakt u een fout, dan krijgt u in de meeste gevallen (maar helaas niet altijd) een foutmelding die u:

- bewust maakt van het feit dat uw code een fout bevat;
- wijst in de richting van de juiste oplossing.

In deze les zult u zien dat u allerlei fouten kunt maken. We tonen u hoe u ze kunt herkennen, maar ook hoe u ze kunt oplossen (of nog beter: voorkomen).

Leerdoelen

Aan het einde van deze les weet u:

- wat een softwarefout is;
- welke soorten softwarefouten er zoal zijn;
- welk type fout u gemaakt heeft;
- hoe u de fout zelf kunt oplossen;
- welke debug-strategie u kunt toepassen;
- waar u iemand anders om hulp kunt vragen;
- hoe u de try...except-constructie toepast.



Enkele afbeeldingen in deze les zijn moeilijk leesbaar in de printversie van deze les. Wilt u een afbeelding uitvergroten? Bekijk dan de digitale versie van de les op Plaza. Klik op de afbeeldingen om ze te vergroten.

9.2 Wat is een softwarefout?

Voor dat we het gaan hebben over verschillende soorten fouten, is het tijd om te kijken naar fouten in het algemeen. Want wanneer is er eigenlijk sprake van een softwarefout?

Een softwarefout is een fout in de programmeercode. Zo'n fout zorgt ervoor dat de gebruiker onverwachte of onjuiste resultaten te zien krijgt. Of erger nog: dat het programma gaat **crashen**, waardoor het voortijdig stopt met functioneren (al dan niet met een foutmelding voor de gebruiker).



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete



For more information about this issue and possible fixes, visit <https://www.windows.com/stopcode>

If you call a support person, give them this info:
Stop code: CRITICAL_PROCESS_DIED

Na een ernstige systeemcrash krijgt een Windows-gebruiker een 'Blue Screen of Death' te zien.

Onder programmeurs wordt de term 'softwarefout' overigens zelden gebruikt. Gebruikelijker is de Engelse term *bug*. Over de herkomst van deze term bestaan verschillende verhalen. Bijvoorbeeld:

- De ontdekking van de eerste *bug* wordt (ten onrechte) toegeschreven aan Grace Hopper. Zij werkte in 1947 op de Harvard University. Bij het zoeken naar de oorzaak van een storing in de *Mark II Aiken Relay Calculator* vond een van de operators een nachtvlinder op een paneel. Dit insect werd in het logboek geplakt, met als bijschrift *First actual case of bug being found* ('eerste echte vondst van ongedierte').
- Volgens Grace Hopper zelf werd de term al tijdens de Tweede Wereldoorlog gebruikt, met als betekenis 'storing in de radar'. Die storing kon bijvoorbeeld worden veroorzaakt door ongedierte, zoals kevers. Het Engelse woord voor dergelijk ongedierte is *bug*.
- Thomas Edison schreef vele jaren eerder, in 1878, het volgende in een brief:

It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise - this thing gives out and [it is] then that 'Bugs' - as such little faults and difficulties are called - show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.

- We kunnen nog verder terug: Webster's 10th Collegiate Dictionary beschrijft het woord *bug* als '*an unexpected defect, fault, flaw, or imperfection*' ('een onverwachte fout of onvolkomenheid'). Daarbij vermeldt men bronnen die teruggaan tot 1622.

0.800	autan started	
1.000	stopped - autan ✓	
13'00 (032)	MP-MC	$\left\{ \begin{array}{l} 1.2700 \cdot 9.037847025 \\ 2.130476415 \end{array} \right.$
(033)	PRO 2	$9.037846995 \text{ const}$
	const	2.130476415
		2.130476415
	Relays 6-2 in 033	Relay 2145
	in Relay	Relay 3370
17'00	Started Cosine Tape (Sine check)	
15'25	Started Multi Adder Test.	
15'45		Relay #70 Panel F (moth) in relay.
16'00	autan not started.	
17'00	closed door.	

Het logboek van Grace Hopper, met daarin een 'bug'.

Uit het gebruik van de term *bugs* ontstond ook de term *debuggen*. Hiermee doelde men in eerste instantie op het controleren of er geen ongedierte in de computer verscholen zat. Nog later kreeg het de betekenis van het verwijderen van fouten uit programmeercode.



Hoewel bijna alle programma's *bugs* bevatten, is *debuggen* niet altijd een absolute noodzaak. Niet alle *bugs* worden als storend ervaren of ze treden te weinig op om er veel tijd in te steken. Desondanks blijft *debuggen* natuurlijk wel een aanrader.

9.3 Verschillende soorten fouten

Nu u weet wat we verstaan onder een softwarefout, is het tijd om in te zoomen op zulke fouten. We kunnen ze verdelen in een aantal categorieën, die we in de volgende subparagrafen behandelen. Per type software fout geven we voorbeelden en laten we zien hoe deze fout doorgaans opgelost kan worden.

9.3.1 Syntaxfout

De eerste foutsoort die we behandelen, is de syntaxfout. Met **syntax** doen we op de taalregels die gelden wanneer u programmeert, oftewel de grammatica van de programmeertaal die u gebruikt.

Om uit te leggen wat een syntaxfout is, kunnen we het beste de vergelijking maken met syntaxfouten in ons dagelijkse taalgebruik. Neem de volgende zinnen:

Ik wordt programmeur.

De kat ligt in de mandje.

Beide zinnen bevatten een grammaticafout: *wordt* in de eerste zin moet *word* zijn en *de* in de tweede zin moet *het* zijn.

Dit type fouten komt ook voor in programmeertalen. Bekijkt u maar eens deze regel code:

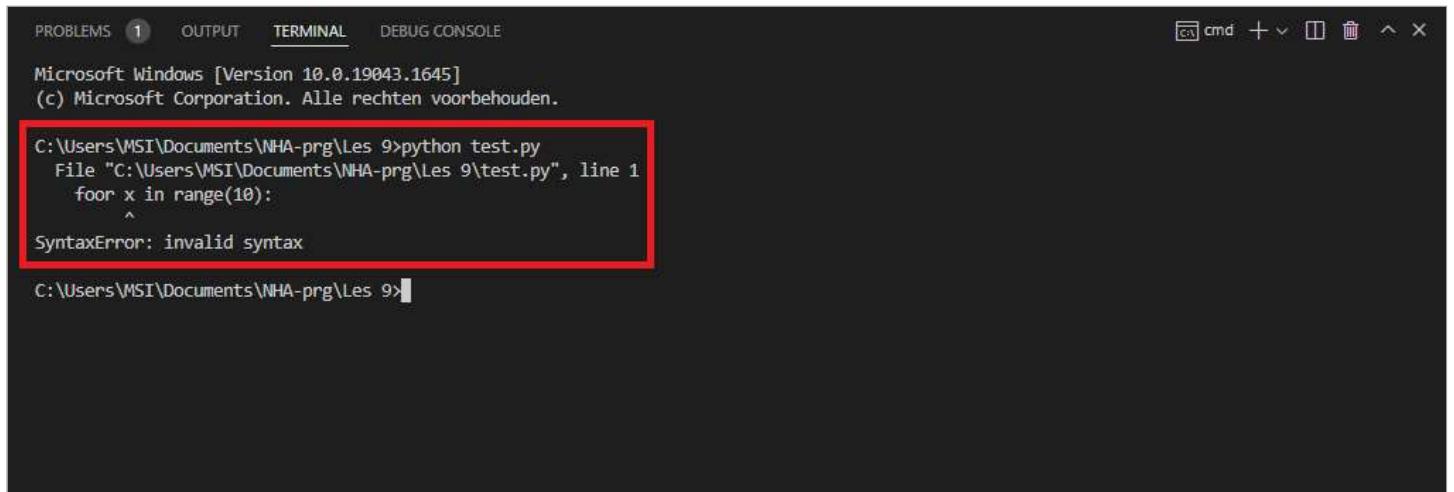
foor x in range(10):

Misschien heeft u al direct door wat de fout in deze regel code is, maar we gaan hem toch invoeren in VS Code.

- Open VS Code.
- Open een map die u eerder aanmaakte, bijvoorbeeld NHA-prg).
- Klik op de knop "New Folder".

- Geef de map een naam (bijvoorbeeld Les 9).
- Klik in de *Menu Bar* op "File".
- Klik op "Open Folder".
- Selecteer de map die u zojuist als laatste aanmaakte (Les 9).
- Klik in de *Side Bar* op de knop "New File".
- Geef het bestand een naam. Eindig met de extensie .py om aan te geven dat het om een Python-bestand gaat (bijvoorbeeld test.py).
- Typ de regel code in het bestand.
- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U krijgt dan een foutmelding te zien:



The screenshot shows a terminal window with the following text:

```
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
File "C:\Users\MSI\Documents\NHA-prg\Les 9\test.py", line 1
    foor x in range(10):
        ^
SyntaxError: invalid syntax

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

A red box highlights the error message: "SyntaxError: invalid syntax".

De foutmelding geeft aan dat het om een `SyntaxError` gaat, oftewel een syntaxfout.

Het pijltje tussen de regelcode en de foutmelding geeft aan waar de fout zit. Maar: in dit geval wijst het pijltje naar de x. Dat is echter niet de plek waar de fout zit!

Helaas is dit iets wat vaak voorkomt bij foutmeldingen en dat maakt het voor u niet gemakkelijker. De *interpreter* doet zijn best om aan te geven waar de fout precies zit, maar vaak zit hij er – letterlijk – naast: hij wijst de code aan die direct na de fout komt. Dat is ook hier het geval: `foor` is fout, dat had `for` moeten zijn.

Dit fenomeen is lastig uit te leggen. U kunt dit wellicht het beste begrijpen als u bedenkt dat `foor` een functie of variabele had kunnen zijn. Pas als er geen = of () achter `foor` verschijnt, zal de *interpreter* constateren dat er iets mis is met de code.

Syntaxfout oplossen

Als u de melding krijgt dat er een syntaxfout is opgetreden, kunt u dus het beste de regel controleren waarvan de *interpreter* aangeeft dat deze een syntaxfout bevat, maar ook de regel ervoor. De kans is groot dat u een typefout aantreft, of dat u een haakje, komma of dubbele punt vergeten bent.

9.3.2 Delen door nul-fout

We leerden het allemaal al op jonge leeftijd:

Delen door nul is flauwekul!

Oftewel, het is niet mogelijk om een getal door nul te delen. Dat we dit weten, wil echter niet zeggen dat we zo'n deling nooit uitvoeren. Zeker in de programmeerwereld komt zo'n fout weleens voor, omdat het getal 0 vaak niet letterlijk in de berekening zelf staat. Bekijk maar eens het volgende voorbeeld:

- Vervang de code in het bestand door de volgende code:

```
x = 0
print(5/x)
```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

Ook nu krijgt u een foutmelding te zien:

A screenshot of a terminal window titled "TERMINAL". The window shows the following text:

```
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
Traceback (most recent call last):
  File "C:\Users\MSI\Documents\NHA-prg\Les 9\test.py", line 2, in <module>
    print(5/x)
ZeroDivisionError: division by zero

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

The error message "ZeroDivisionError: division by zero" is highlighted with a red box.

De foutmelding geeft aan dat het om een `ZeroDivsionError` gaat, oftewel een fout die wordt veroorzaakt door delen door nul (`division by zero`).

Delen door nul-fout oplossen

Het oplossen van deze fout is simpel: verander de nul door een ander getal. Is dat niet mogelijk, zorg er dan voor dat er geen bewerkingen met dit getal worden uitgevoerd (bijvoorbeeld met een `if`-statement).

9.3.3 Importeefout

U heeft in de vorige les al eens een module geïmporteerd. Dat gebeurt meestal aan het begin van het computerprogramma. In de volgende les gaan we hier nog dieper op in.

Uiteraard kunnen ook bij het importeren van modules, functies en variabelen fouten gemaakt worden.

- Vervang de code in het bestand door de volgende code:

```
from math import division_by_2
```

- Sla de wijzigingen op.
- Voer het programma uit in de `Terminal`.

U krijgt dan de volgende foutmelding te zien:

A screenshot of a terminal window titled "TERMINAL". The window shows the following text:

```
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
Traceback (most recent call last):
  File "C:\Users\MSI\Documents\NHA-prg\Les 9\test.py", line 1, in <module>
    from math import division_by_2
ImportError: cannot import name 'division_by_2' from 'math' (unknown location)

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

The error message "ImportError: cannot import name 'division_by_2' from 'math' (unknown location)" is highlighted with a red box.

Deze foutmelding bestaat uit een aantal delen:

- `ImportError` geeft aan dat het om een importeefout gaat.
- De foutmelding geeft ook aan wat er niet geïmporteerd kan worden. In dit geval is dat `division_by_2`.
- Tot slot wordt er ook nog een oorzaak vermeld: hier is dat een `unknown location` (onbekende locatie).

Importfout oplossen

In tegenstelling tot de twee eerdere fouten zullen we hier niet uitleggen hoe u dit type fout kunt oplossen. Dat leert u in een latere les (waarin u ook meer leert over het importeren zelf).

9.3.4 Bereikfout (scope)

De volgende fout die we behandelen, heeft betrekking op het bereik van een variabele. In de programmeerwereld wordt dit ook wel de **scope** van de variabele genoemd.

De *scope* geeft aan hoever een variabele kan reiken, oftewel: tot waar in de code u de variabele kunt gebruiken. Dat dit bereik niet oneindig is, laten we u zien aan de hand van een voorbeeld:

- Vervang de code in het bestand door de volgende code:

```
leeftijd = 10

def over_2_jaar():
    uitvoer = leeftijd + 2
    return uitvoer

print(over_2_jaar())
```

- Sla de wijzigingen op.
 - Voer het programma uit in de *Terminal*.

De waarde van de variabele bevindt zich in de hoofdcode. Die kunnen we zonder problemen uitlezen; de uitvoer is 12.

Dat wordt anders wanneer we waarde van de variabele aanpassen binnen de functie:

- Wijzig de code op de volgende manier:

```
leeftijd = 10

def over_2_jaar():
    leeftijd = leeftijd + 2
    return leeftijd

print(over_2_jaar())
```

- Sla de wijzigingen op.
 - Voer het programma uit in de *Terminal*

U krijgt dan de volgende foutmelding te zien:

Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
Traceback (most recent call last):
  File "C:/Users/MSI/Documents/NHA-prg/Les 9/test.py", line 7, in <module>
    print(over_2_jaar())
  File "C:/Users/MSI/Documents/NHA-prg/Les 9/test.py", line 4, in over_2_jaar
    leeftijd = leeftijd + 2
UnboundLocalError: local variable 'leeftijd' referenced before assignment
```

We nemen deze relatief uitgebreide foutmelding stap voor stap met u door:

- Net als de eerdere foutmeldingen begint ook deze foutmelding met Traceback (most recent call last):..

- Daarna geeft de *interpreter*, net als in de eerdere foutmeldingen, aan welke regel(s) een fout bevat. In ons geval worden regel 4 en regel 7 genoemd:
 - Regel 7 is het moment waarop het fout gaat. Dat is de regel waarop de functie wordt aangeroepen. (Wanneer u deze regel zou verwijderen, kunt u de functie uitvoeren zonder foutmelding, maar dan krijgt u ook geen 12 als uitvoer.)
 - De daadwerkelijke fout zit echter in regel 4.
- Welke fout er te vinden is in regel 4, wordt daarna aangegeven:

```
UnboundLocalError: local variable 'leeftijd' referenced before assignment
```

Zonder deze zin al te veel te ontleden, zal u duidelijk worden dat er iets mis is met de variabele `leeftijd`. De foutmelding geeft aan dat we variabele gebruiken waarvan de waarde niet bekend is.

Bereikfout oplossen

Deze fout zou u moeten kunnen oplossen met de kennis die u opdeed in de vorige les. In plaats van het antwoord hier gewoon neer te zetten, geven we u dus eerst de kans om het zelf te formuleren.

Controleer uw oplossing in de voetnoot.

9.3.5 Logicafout

De laatste foutsoort die we bespreken, is een lastige fout. Dat komt doordat de Python-*interpreter* deze fout niet kan ontdekken. Het is namelijk een fout in uw denkwijze. We laten zien hoe dit zit aan de hand van een voorbeeld:

- Vervang de code in het bestand door de volgende code:

```
getal1 = input("Wat is getal 1? ")
getal2 = input("Wat is getal 2? ")
uitkomst = getal1 + getal2

print(f"{getal1} + {getal2} = {uitkomst}")
```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.
- Beantwoord de vraag `Wat is getal 1?` met 10 en druk op Enter.
- Beantwoord de vraag `Wat is getal 2?` met 10 en druk op Enter.

U krijgt dan de uitkomst 1010 als uitvoer:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
Wat is getal1? 10
Wat is getal2? 10
10 + 10 = 1010

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

De programmeur is hier vergeten dat de standaardfunctie `input` een *string* als waarde aan variabele `getal1` zal toekennen. Datzelfde geldt voor de variabele `getal2`. Als deze *strings* worden opgeteld, zullen ze aan elkaar worden geplakt (zoals u al eens eerder in deze cursus zag). In plaats van 20, ziet u dus 1010. Dit noemen we een **logicafout**.

Zoals gezegd zult u bij dit type fout helaas geen foutmelding te zien krijgen. Dat de uitkomst niet is wat u zou verwachten en dat u geen foutmelding te zien krijgt, moeten u aan het denken zetten: er is sprake van een logicafout.

Logicafout oplossen

Hoe u deze specifieke logicafout moet oplossen, leerde u al in een eerdere les: u moet de *strings* veranderen in *integers* (met `int()`).

Heeft u er moeite mee om te achterhalen waar het fout gaat? Probeer dan eens gebruik te maken van de **rubber duck method**. Deze methode houdt in dat u een badeendje koopt en dat naast uw computer zet. Leg vervolgens iedere code uit aan het badeendje.

Doorloop uw code regel voor regel en leg uit wat iedere regel zou moeten doen. Na het uitleggen van een regel gaat u na of er daadwerkelijk gebeurt wat er zou moeten gebeuren. Dat 'nagaan' kan door nog eens goed te bedenken wat de waarde van elke variabele nou werkelijk zal zijn, of - als dat lastig is of niets oplevert - die variabele tijdelijk uit te printen (waarover later meer).



De code uitleggen aan een badeend klinkt misschien onnozel, maar het is in feite een krachtige foutopsporingsmethode.



Ga naar Plaza en bekijk een (Engelstalige) video over deze methode.

9.4 Debug-strategieën

De *rubber duck method* is een voorbeeld van een zogenaamde *debug-strategie*. Deze strategie kunt u gebruiken om softwarefouten uit uw code te halen.

Er zijn nog meer strategieën die het vermelden waard zijn. We bespreken ze in de volgende subparagrafen.

9.4.1 Verdeel en heers

We beginnen deze subparagraaf met een situatieschets:

U krijgt een foutmelding die aangeeft dat er een probleem is in regel 130. Als u regel 130 bekijkt, kunt u echter niets vinden dat een fout kan veroorzaken. Maar waar zit de fout dan wel?

De volledige code uitleggen aan uw badeend gaat vrij veel tijd kosten, net als het printen van iedere variabele.

Een alternatief is delen van uw code veranderen in commentaar, zodat u kunt achterhalen in welk deel van de code de fout zich bevindt.

In een eerdere les leerde u dat u commentaar op twee manieren kunt toevoegen:

1. Met behulp van een hashtag

De *hashtag (#)* geeft aan dat alles wat er achter getypt wordt, commentaar is, en dus niet door de *interpreter* uitgevoerd dient te worden. Bijvoorbeeld:

```
# prijzen van ijsjes
cornetto_prijs = 1.50
raketje_prijs = 0.75
```

```
# aantal ijsjes
aantal_cornettos = 2
```

```
aantal_raketjes = 3
```

De rode code wordt niet uitgevoerd door de interpreter.

2. Met behulp van *triple quotes*

Deze methode komt van pas wanneer u meerdere regels met commentaar wilt toevoegen. Alles tussen de *triple quotes* (oftewel: drie enkele aanhalingstekens) negeert de interpreter. Bijvoorbeeld:

...

Dit is een commentaar van meerdere regels.

Dit kan handig zijn als u veel informatie wilt geven.

...

Bij deze *debug-strategie* kunt u beide methodes gebruiken. De tweede methode is echter het gemakkelijkst: dan hoeft u maar twee keer drie aanhalingstekens te plaatsen, terwijl u bij de eerste methode voor iedere regel die u niet wilt uitvoeren, een *hashtag* moet plaatsen.

Door telkens een ander deel van uw code uit te sluiten, kunt u nagaan in welk deel van de code de fout zich bevindt. Deze strategie komt vooral van pas als uw code erg lang is.

9.4.2 Variabelen uitprinten

De volgende strategie die we bespreken, noemden we al eens in de voorgaande paragraaf: het tijdelijk uitprinten van iedere variabele. Hoe dit werkt, laten we zien aan de hand van een voorbeeld:

- Vervang de code in het bestand door de volgende code:

```
def pi():
    return 3.1415

radius = 2
oppervlakte = pi * (radius ** 2)

print(oppervlakte)
```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U krijgt dan deze foutmelding te zien:

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
```

Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
Traceback (most recent call last):
  File "C:\Users\MSI\Documents\NHA-prg\Les 9\test.py", line 5, in <module>
    oppervlakte = pi * (radius ** 2)
TypeError: unsupported operand type(s) for *: 'function' and 'int'

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

A red box highlights the error message in the terminal output.

Uit deze melding blijkt dat de fout zich op regel 5 bevindt. Maar wat is dan de fout?

Om dat te kunnen achterhalen, kunt u de variabele `pi` printen (voordat u de berekening op regel 5 maakt).

- Wijzig de code op de volgende manier:

```
def pi():
    return 3.1415

radius = 2
print(pi)
#oppervlakte = pi * (radius ** 2)

#print (oppervlakte)
```



We gebruiken in deze code niet alleen de strategie 'variabelen printen', maar ook de strategie 'verdeel en heers'. We zorgen er namelijk voor dat de laatste twee regels als commentaar worden gezien (met behulp van hashtags).

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U ziet dan dit als uitvoer te zien:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
<function pi at 0x0000018DE537F040>
C:\Users\MSI\Documents\NHA-prg\Les 9>
```

Hiermee probeert de *interpreter* u duidelijk te maken dat hij *pi* ziet als een functie, niet als een variabele (althans: niet een variabele zoals wij dachten). U moet hieruit kunnen afleiden dat we de haakjes achter *pi* vergeten zijn (in regel 5).

- Haal de twee *hashtags* weg (regel 6 en 8).
- Verwijder de regel *print(pi)*.
- Voeg de haakjes toe, zodat regel 5 er zo komt uit te zien:

```
oppervlakte = pi() * (radius ** 2)
```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U krijgt dan de juiste uitvoer te zien:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python test.py
12.566
C:\Users\MSI\Documents\NHA-prg\Les 9>
```

9.4.3 Hard coding

Ten slotte kunnen we het hardcoden van variabelen nog als een strategie beschouwen. Hiermee bedoelen we dat we de waarde van een bepaalde variabele zelf invoeren, in plaats van die te laten bepalen door een formule, functie of gebruikersinvoer.

Een voorbeeld:

- Vervang de code in het bestand door de volgende code:

```

import math

def discriminant(a,b,c):
    # voor: ax^2 + bx + c
    D1 = (-b + math.sqrt(b**2 - 4*a*c))/(2 * a)
    D2 = (-b - math.sqrt(b**2 - 4*a*c))/(2 * a)

    uitvoer = [D1,D2]
    return uitvoer

print("Voor de formule ax^2 + bx + c, geef a , b en c:")
a = int(input("Wat is a? "))
b = int(input("Wat is b? "))
c = int(input("Wat is c? "))

uitkomst = discriminant(a,b,c)

D1 = uitkomst[0]
D2 = uitkomst[1]

print(f"De discriminant(en) voor {a}x^2 + {b}x + {c} zijn :")
print(f"{D1} en {D2}")

```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.
- Beantwoord de vraag *Wat is a?* met 0 en druk op Enter.
- Beantwoord de vraag *Wat is b?* met 0 en druk op Enter.
- Beantwoord de vraag *Wat is c?* met 0 en druk op Enter.

U krijgt dan een foutmelding te zien:

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python fouten.py
Voor de formule ax^2 + bx + c, geef a , b en c:
Wat is a? 0
Wat is b? 0
Wat is c? 0
Traceback (most recent call last):
  File "C:/Users/MSI/Documents/NHA-prg/Les 9/fouten.py", line 17, in <module>
    uitkomst = discriminant(a,b,c)
  File "C:/Users/MSI/Documents/NHA-prg/Les 9/fouten.py", line 5, in discriminant
    D1 = (-b + math.sqrt(b**2 - 4*a*c))/(2 * a)
ZeroDivisionError: float division by zero
C:\Users\MSI\Documents\NHA-prg\Les 9>

```

Dat komt doordat 'de gebruiker' (in dit geval *uzelf*) rare waarden heeft ingevoerd.

Om toch te kunnen testen of onze formule werkt, kunnen we *tijdelijk* de 'input' vervangen door 'harde waarden' :

- Wijzig de code op de volgende manier:

```

print("Voor de formule ax^2 + bx + c, geef a , b en c:")
a = -3 #int(input("Wat is a? "))
b = 2 #int(input("Wat is b? "))
c = 5 #int(input("Wat is c? "))

uitkomst = discriminant(a,b,c)

D1 = uitkomst[0]

```

```
D2 = uitkomst[1]

print(f"De discriminant(en) voor {a}x^2 + {b}x + {c} zijn :")
print(f"{D1} en {D2}")
```



We gebruiken in deze code niet alleen de strategie ‘*hard coding*’, maar ook de strategie ‘verdeel en heers’. We zorgen er namelijk voor dat de gebruiker geen input mag leveren door de drie vragen te definiëren als commentaar (met behulp van hashtags).

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

Door het wijzigen van de code heeft u de gebruikers invoer uitgeschakeld. In plaats daarvan heeft u de drie dezelfde waardes zelf ingevoerd.

Nu krijgt u wel een oplossing te zien en kunt u controleren of de formule klopt.

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 9>python fouten.py
Voor de formule ax^2 + bx + c, geef a , b en c:
De discriminant(en) voor -3x^2 + 2x + 5 zijn :
-1.0 en 1.6666666666666667

C:\Users\MSI\Documents\NHA-prg\Les 9>
```

Nu is het wel mogelijk om te controleren of de formule klopt.

9.5 Overige hulpmiddelen

De besproken oplossingen en strategieën hadden, behalve het doel, nog iets gemeen: u moest de fout zelf oplossen. Daarmee hebben we waarschijnlijk bij u de indruk gewekt dat u er alleen voorstaat wanneer uw code een fout bevat. In deze paragraaf zullen we u laten zien dat dit niet het geval is.

We werken in deze lessenserie met Python, omdat het voor beginners een perfecte taal is. Dat is niet alleen doordat deze gebruiksvriendelijk en eenvoudig te leren is, maar ook vanwege zijn populariteit: er zijn veel mensen die deze taal gebruiken. Anders gezegd: Python heeft een grote *community*.

Deze grote *community* zorgt niet alleen voor een grote hoeveelheid toegankelijke en bruikbare bronnen, maar kan ook dienen als vraagbaak. Dat is handig als u behoefte heeft aan ondersteuning, hulp bij het leren van Python of tegen ontwikkelproblemen of uitdagingen aanloopt.

In de volgende subparagrafen bekijken we hoe u de hulp van anderen kunt inschakelen.

9.5.1 Google

De snelste manier om hulp te vinden, is het gebruiken van een online zoekmachine. In deze subparagraaf gebruiken we de bekendste: *Google*.

We gaan in ons voorbeeld uit van een foutmelding die u eerder kreeg, namelijk `ValueError: math domain error`. Door *hard coding* heeft u achterhaald dat deze foutmelding werd veroorzaakt door de gebruikersinvoer, maar u weet niet welke waarde er fout was en waarom.

- Ga naar www.google.com.
- Typ de foutmelding in de zoekbalk.
- Druk op Enter.

U krijgt dan verschillende zoekresultaten te zien. Het eerste zoekresultaat ziet er vaak iets anders uit dan de rest van de resultaten. *Google* toont u hier namelijk, in vette tekst, het antwoord waarnaar u waarschijnlijk zoekt.



ValueError: math domain error

[Alle](#)[Afbeeldingen](#)[Video's](#)[Shopping](#)[Nieuws](#)[Meer](#)[Tools](#)

Ongeveer 3.300.000 resultaten (0,59 seconden)

The ValueError: math domain error is raised when you perform a mathematical function on a negative or zero number which cannot be computed. To solve this error, make sure you are using a valid number for the mathematical function you are using. 23 feb. 2021

<https://careerkarma.com> > Blog > Python Tutorials[Python ValueError: math domain error Solution - Career Karma](#)[Over samenvattingen](#) • [Feedback](#)<https://stackoverflow.com> > questions < Vertaal deze pagina[ValueError: math domain error - python - Stack Overflow](#)

27 jan. 2015 — you are getting math domain error for either one of the reason : either you are trying to use a negative number ...

5 antwoorden · Topantwoord: Your code is doing a log of a number that is less than or equal to...

How to fix 'ValueError: math domain error' in python? - Stack ... 26 mei 2019

python math domain errors in math.log function - Stack Overflow 30 sep. 2013

Why do I get ValueError : math domain error? - python - Stack ... 30 nov. 2012

Python,how to solve ValueError: math domain error - Stack ... 20 jan. 2020

Meer resultaten van stackoverflow.com

Het eerste zoekresultaat ziet er vaak iets anders uit dan de rest van de resultaten. Google toont u hier namelijk, in vette tekst, het antwoord waarnaar u waarschijnlijk zoekt.

In ons geval is dat ook inderdaad wat we willen weten: de fout treedt op doordat we een wiskundige functie uitvoeren met een negatief getal. Dat kan niet!



Doordat de foutmelding geschreven is in het Engels, toont Google u ook een Engelstalig antwoord. De kans is groot dat ook de overige zoekresultaten u zullen leiden naar Engelstalige websites. U kunt Google vragen om uitsluitend Nederlandstalige zoekresultaten te tonen:

- Klik onder de zoekbalk op "Tools".
- Klik op "Elke taal".
- Klik op "Zoeken in pagina's in het Nederlands".

De kans is groot dat u nu ook Nederlandstalige pagina's te zien krijgt, maar dat zullen er veel minder zijn dan Engelstalige pagina's.

9.5.2 Stack Overflow

Onder het eerste resultaat (of niet ver daarvandaan), vindt u een interessante website als resultaat: *Stack Overflow*. Dit is dé plek waar u moet zijn als het gaat om debugging.

Veel programmeurs nemen de tijd om anderen te helpen: beginners, maar ook professionals die aan complexe taken werken. Op dit populaire platform vindt u vragen en antwoorden, ongeacht uw onderwerp of niveau.

- Klik op het zoekresultaat dat u leidt naar www.stackoverflow.com.

Wat u nu in beeld ziet, noemen we een **thread**. De *thread* begint met de vraag, daaronder staan de eventuele antwoorden. Meestal bevat zo'n antwoord niet alleen de oorzaak van een fout, maar ook de oplossing. Let wel: de voertaal op het platform is Engels.

U kunt vragen stellen of antwoorden geven als u een account heeft. Voor het lezen van bestaande vragen en antwoorden is dat niet nodig.

stackoverflow About Products For Teams Search... Log in Sign up

Home PUBLIC Questions Tags Users Companies COLLECTIVES Explore Collectives TEAMS Stack Overflow for Teams – Start collaborating and sharing organizational knowledge. Create a free Team Why Teams?

ValueError: math domain error

Asked 9 years, 1 month ago Modified 5 months ago Viewed 420k times

I was just testing an example from *Numerical Methods in Engineering with Python*.

```
from numpy import zeros, array
from math import sin, log
from newtonRaphson2 import *
def f(x):
    f = zeros(len(x))
    f[0] = sin(x[0]) + x[1]**2 + log(x[2]) - 7.0
    f[1] = 3.0*x[0] + 2.0**x[1] - x[2]**3 + 1.0
    f[2] = x[0] + x[1] + x[2] - 5.0
    return f
x = array([1.0, 1.0, 1.0])
print newtonRaphson2(f,x)
```

When I run it, it shows the following error:

```
File "example_NR2method.py", line 8, in f
    f[0] = sin(x[0]) + x[1]**2 + log(x[2]) - 7.0
ValueError: math domain error
```

I have narrowed it down to the log as when I remove log and add a different function, it works. I assume it is because of some sort of interference with the base, I can't figure out how. Can anyone suggest a solution?

[python](#) [runtime-error](#) [logarithm](#)

Share Improve this question Follow edited Feb 18, 2014 at 13:26 asked Apr 8, 2013 at 22:58 André Stamenk 7,603 ● 32 ● 49 ramanunni.pm 1,429 ● 2 ● 10 ● 8

Add a comment

The Overflow Blog Software is adopted, not sold (Ep. 441) An unfiltered look back at April Fools' 2022 Featured on Meta Staging Ground: Reviewer Motivation, Scaling, and Open Questions Overhauling our community's closure reasons and guidance WE LOVE THE POSITION! Relay42 Artificial Intelligence We have great benefits! Flexible working hours and workspace Pioneering entrepreneurial atmosphere Pension scheme Travel allowance + 5 more benefits Learn more

Voorbeeld van een thread op www.stackoverflow.com.

9.5.3 Python.org

Lukt het niet met behulp van de twee eerder genoemde hulpmiddelen, dan is het bezoeken van de officiële website van Python een optie.

- Ga naar www.python.org.
- Klik op "Documentation".

Op deze pagina vindt u allerlei documentatie over Python. De diverse documenten zijn onder meer gecategoriseerd op niveau:

Python PSF Docs PyPI Jobs Community

python™ Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

Browse the docs online or download a copy of your own.

Python's documentation, tutorials, and guides are constantly evolving. Get started here, or scroll down for documentation broken out by type and subject.

Python Docs See also [Documentation Releases by Version](#)

Beginner

- Beginner's Guide
- Python FAQs

Moderate

- Python Periodicals
- Python Books

Advanced

- Python Packaging User Guide
- In-development Docs
- Guido's Essays

General

- PEP Index
- Python Videos
- Developer's Guide

In plaats van het aanklikken van een van de links, kunt u er ook voor kiezen om de zoekbalk rechtsboven in beeld te gebruiken.

9.5.4 W3Schools

Ook W3Schools biedt u documentatie op het gebied Python. U vindt er echter nog veel meer:

HTML and CSS	JavaScript	Server Side	Data Analytics
Learn HTML	Learn JavaScript	Learn SQL	Learn AI
Learn CSS	Learn jQuery	Learn MySQL	Learn Machine Learning
Learn RWD	Learn React	Learn PHP	Learn Data Science
Learn Bootstrap	Learn AngularJS	Learn ASP	Learn NumPy
Learn W3.CSS	Learn JSON	Learn Node.js	Learn Pandas
Learn Colors	Learn AJAX	Learn Raspberry Pi	Learn SciPy
Learn Icons	Learn AppML	Learn Git	Learn Matplotlib
Learn Graphics	Learn W3.JS	Learn AWS Cloud	Learn Statistics
Learn SVG	Programming	Web Building	Learn Excel
Learn Canvas			Learn Google Sheets
Learn How To			XML Tutorials
Learn Sass			
	Learn Python	Create a Website <small>NEW</small>	
	Learn Java	Where To Start	
	Learn C	Web Templates	
	Learn C++	Web Statistics	
	Learn C#	Web Certificates	
	Learn R	Web Development	
	Learn Kotlin	Code Editor	Learn XML Schema
	Learn Go	Test Your Typing Speed	Learn XSLT
	Learn Django	Play a Code Game	Learn XPath
	Learn TypeScript	Cyber Security	Learn XQuery
		Accessibility	

De documentatie over Python vindt u op www.w3schools.com/python.

9.5.5 Overige communities

Op het internet vindt u ook diverse **communities**, oftewel groepen mensen met dezelfde interesses. Dit kan bijvoorbeeld gaan om een groep programmeurs die elkaar helpt en nuttige tips deelt.

Als u op platformen als *Facebook* en *Discord* zoekt op 'Python', zult u zeker groepen vinden die u verder kunnen helpen bij uw ontwikkeling als programmeur. (Wellicht zelfs een groep waarin Nederlands de voertaal is!)

9.6 Project 'IJssalon'

Omdat we in deze les eigenlijk geen nieuwe concepten hebben behandeld, zullen we niet veel toevoegen aan ons project. Om de werklast over de lessen te verdelen, zullen we in deze les toch iets doen: we voegen aan het bestand 'helper.py' een functie toe die Ellie helpt met het verdelen van de fooienpot over de medewerkers in haar ijssalon.

- Ga naar VS Code.
- Open de map die u eerder aanmaakte voor de uitwerking van deze casus (bijvoorbeeld **IJssalon-start**).
- Open het bestand 'helper.py' (dat u ook eerder aanmaakte).

Dit bestand bevat, als het goed is, de volgende code:

```
def decoreer(tekst=""):
    lengte = len(tekst) + 4
    print()
    print(lengte * "*")
    print(f"* {tekst} *")
    print(lengte * "*")
    print()
```

Anders dan we in vorige lessen deden, zullen we u niet letterlijk aangeven welke code u moet toevoegen. We plaatsen een beschrijving, op basis waarvan u de code uitbreidt. Verderop in deze les tonen we u wat de juiste code had moeten zijn.

De beschrijving van de toe te voegen functie is als volgt:

- De naam van de functie is `fooi_pp()`.
- De functie heeft twee parameters:
 - `bedrag`: het totaalbedrag van alle fooi, oftewel het bedrag dat verdeeld moet worden;
 - `personen`: het aantal personen waarover het bedrag verdeeld dient te worden.
- Op de eerste regel van de functie zet u een variabele met de naam `bedrag_pp`, die als waarde de berekening 'bedrag gedeeld door personen' krijgt.
- Deze variabele geeft een uitvoer met behulp van het `return`-statement, waarbij u via een *formatted string* een gebruikersvriendelijke uitvoer genereert in de trant van `Het bedrag per persoon is 10 euro` (of iets dergelijks).
- Onder de functie (dus buiten de functies, in de hoofdcode) roept u de functie als volgt aan:

```
b = int(input("Welk bedrag zit er in de fooienpot? "))
p = int(input("Over hoeveel mensen moet de pot verdeeld worden? "))

print(fooi_pp(b,p))
```



Deze laatste regels kunt u het beste verwijderen nadat u klaar bent met deze les. Het zal geen fouten veroorzaken wanneer u de regels niet verwijdert, het staat alleen een beetje slordig.

Schrijf de code en probeer deze uit door het bestand 'helper.py' uit te voeren in de *Terminal*. Voer verschillende waarden in om te kijken of uw code goed werkt. Zo niet: probeer de *debug*-strategieën uit voordat u spekt wat de juiste code had moeten zijn.

De juiste code had moeten zijn:

```
def fooi_pp(bedrag, personen):
    bedrag_pp = bedrag/personen
    return f"Het bedrag per persoon is {bedrag_pp} euro"

b = int(input("Welk bedrag zit er in de fooienpot? "))
p = int(input("Over hoeveel mensen moet de pot verdeeld worden? "))

print(fooi_pp(b,p))
```

Wanneer u deze code uitvoert in de *Terminal*, krijgt u de volgende uitvoer (bij het invoeren van 99 als bedrag en 10 als aantal personen):

A screenshot of a terminal window titled 'cmd'. The window shows the command 'python helper.py' being run. The output of the program is displayed in a red box:

```
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\IJssalon\IJssalon-start>python helper.py
Welk bedrag zit er in de fooienpot?99
Over hoeveel mensen moet de pot verdeeld worden?10
Het bedrag per persoon is 9.9 euro

C:\Users\MSI\Documents\NHA-prg\IJssalon\IJssalon-start>
```

U weet inmiddels dat u, wanneer u de tweede vraag (het aantal personen) had beantwoord met 0, een foutmelding zou krijgen:

```
ZeroDivisionError: division by zero
```

In dat geval krijgt de gebruiker dus helemaal geen 'normale' uitvoer te zien, het programma stopt abrupt.

Eerder in deze les zag u ook al een fout die werd veroorzaakt door de gebruiker. We gaven toen het volgende aan:

Doordat u weet dat de foutmelding wordt veroorzaakt door de gebruikersinvoer, weet u dat u moet bedenken hoe u voorkomt dat de gebruiker een foutieve waarde invoert (of het op een chiquer manier oplost).

We laten u nu zo'n 'chiquere manier' zien:

- Wijzig de code op de volgende manier:

```
def fooi_pp(bedrag, personen):
    try:
        bedrag_pp = bedrag/personen
    except:
        bedrag_pp = "??"
    return f"Het bedrag per persoon is {bedrag_pp} euro"

b = int(input("Welk bedrag zit er in de fooienpot? "))
p = int(input("Over hoeveel mensen moet de pot verdeeld worden? "))

print(fooi_pp(b,p))
```

- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U ziet dat het programma nu niet abrupt stopt (met een foutmelding), maar dat de uitvoer er redelijk normaal uitziet. Het enige verschil is dat u in plaats van een getal, twee vraagtekens te zien krijgt. Daarvoor hebben we gebruikgemaakt van de try...except-constructie:

- Het eerste deel is `try`, wat Engels is voor 'proberen'. U vraagt de interpreter de berekening te proberen uit te voeren.
- Het tweede deel is `except`, wat Engels is voor 'behalve'. Daarmee geeft u aan dat de uitvoer `Het bedrag per persoon is {bedrag_pp} euro` teruggegeven mag worden, behalve wanneer de berekening uitgevoerd kan worden.

Delen door nul is flauwekul, dus de berekening kan niet uitgevoerd worden. In dat geval wordt de uitvoer getoond die u definieerde in het except-deel van de code, namelijk `??`.



In dit geval hoeven we de constructie maar één keer te gebruiken, maar een code mag meerdere van deze constructies bevatten!

9.7 Samenvatting

In deze les hebben we stilgestaan bij het fenomeen 'softwarefouten', in de programmeerwereld ook wel *bugs* genoemd.

Iedereen die gaat programmeren, zal vroeg of laat een fout maken. U heeft gezien dat dit verschillende soorten fouten kunnen zijn, ieder met zijn eigen oorzaak en passende oplossing.

Soms kunt u fout gemakkelijk opsporen, maar het zal u soms ook de nodige tijd kosten. We bespraken in deze les diverse tactieken die u kunt gebruiken in uw zoektocht, zoals de code uitleggen aan een badeendje, variabelen printen of delen van de code uitsluiten door ze tijdelijk in commentaar te veranderen.

Wil het dan nog niet lukken, dan kunt u de hulp van andere programmeurs inschakelen. U zoekt allereerst of er al documentatie over dit probleem bestaat, daarna stelt u uw vraag eventueel aan de Python-community (bijvoorbeeld via *Stack Overflow*).

Tot slot hebben we u laten zien hoe u de try...except-constructie toepast.

9.8 Vraagmoment

Heeft u vragen over deze les? Dan kunt u die stellen via een vraagmoment.

U herkent een vraagmoment aan het spreekballonnetje dat op Plaza in het overzicht "Lessen" bij een les staat. Door op deze spreekballoon te klikken, komt u op een nieuwe pagina, waar u uw vraag in het tekstvak kunt typen.



Het tekstvak mag maximaal vijfhonderd tekens bevatten. Zorg er dus voor, dat u uw vraag/vragen bondig formuleert.

Nadat u op "Verstuur vraag" heeft geklikt, wordt uw vraag automatisch naar uw docent verstuurd. Zodra uw docent de vraag heeft beantwoord, verschijnt er op uw dashboard bij de opleiding een spreekballon.

U kunt per les één keer vragen stellen. Verzamel dus uw vragen over de les, voordat u deze instuurt. Als u gebruik heeft gemaakt van het vraagmoment, verdwijnt het spreekballonnetje bij de les.

9.9 Oefenopgaven

De volgende opgaven werkt u als oefenopgaven voor uzelf uit. De uitwerkingen van deze opgaven kunt u meteen zelf controleren.

Wat is een softwarefout?

→ Bekijk antwoord

Welke term gebruiken programmeurs om een softwarefout aan te duiden?

→ Bekijk antwoord

Wat is *debuggen*?

→ Bekijk antwoord

Wat heeft u fout gedaan als u een syntaxfout maakt?

→ Bekijk antwoord

Wat is de scope van een variabele?

→ Bekijk antwoord

Hoe kunt u een bereikfout meestal oplossen?

→ Bekijk antwoord

Welke foutmelding krijgt u te zien wanneer uw code een logicafout bevat?

→ Bekijk antwoord

Bekijk de volgende code:

```
eerste_string_var = "Eerste String"
eerste_int_var = 1
totaal = 1 + 2 * 3
```

Welk type foutmelding krijgt u als u deze code uitvoert? Waardoor wordt deze fout veroorzaakt?

→ Bekijk antwoord

Bekijk de volgende code:

```
a = 10

def dubbel():
    a = a * 2

print(dubbel())
```

Welk type foutmelding krijgt u als u deze code uitvoert? Waardoor wordt deze fout veroorzaakt?

→ Bekijk antwoord

Wat doet u als u de *rubber duck method* toepast?

→ Bekijk antwoord

Bekijk de volgende code:

```
try:
    print(10/0)
except:
    print("0/10")
```

Wat krijgt u te zien als u deze code uitvoert?

→ Bekijk antwoord

9.10 Huiswerkopgaven

Maak deze huiswerkopgaven en stuur ze via Plaza naar uw docent. Deze opgaven worden nagekeken en voorzien van een cijfer. Mocht u een onvoldoende cijfer behalen, dan krijgt u een herkansing.

De volgende stappen vormen samen de huiswerkopgaven:

1. Werk in de map die hoort bij les 9, niet bij het project 'IJssalon'. Maak in deze map een bestand aan met de naam 'fouten.py'.
2. Importeer op de eerste regel van dit bestand de module `math` op de volgende manier:

```
import math
```

(In de volgende les zullen we meer uitleg geven over `import`.)

3. Maak vervolgens een functie met de naam `discriminant`. Deze functie heeft drie parameters: `a`, `b` en `c`.

4. Definieer in de functie `discriminant` twee variabelen, genaamd `D1` en `D2`. Deze variabelen bevatten de volgende formules:

```
D1 = (-b + math.sqrt(b**2 - 4*a*c))/(2 * a)
D2 = (-b - math.sqrt(b**2 - 4*a*c))/(2 * a)
```

5. Definieer in de functie `discriminant` een *list* met de naam `uitvoer`. Vul deze *list* met de variabelen `D1` en `D2`. Geef deze *list* terug als return-waarde voor de functie.

6. Print (in de hoofdcode) de volgende uitvoer:

Voor de formule $ax^2 + bx + c$, geef a, b en c:

7. Zorg ervoor dat de gebruiker de waarden van a, b en c kan invoeren. Gebruik hiervoor het `input`-statement.



Vergeet niet de `input`-string om te vormen naar een `integer`!

8. Definieer een variabele met de naam `uitkomst` en stel deze gelijk aan `discriminant(a,b,c)`.

9. Stel `D1` gelijk aan het eerste element in de *list* `uitkomst` en `D2` aan het tweede element in de *list* `uitkomst`.

10. Print een *formatted string* die als uitvoer zowel de waarden voor a, b en c als de waarde voor `D1` en `D2` geeft.

11. Bedenk een `try...except`-constructie die voorkomt dat er een foutmelding optreedt als de waarden voor a, b en c een fout in de berekening in de functie 'discriminant' veroorzaakt. (Dat zal bijvoorbeeld optreden wanneer u a de waarde 1 geeft b de waarde 2 en c de waarde 3.) In beide gevallen moeten `D1` en `D2` de waarde geen `oplossing` krijgen.

Stuur het bestand 'fouten.py' via Plaza naar uw docent.