

Les 8 - Functies

[« Vorige les](#)[» Volgende les](#)

Huiswerk

Programmeren voor Beginners - Les 8

[Ga naar huiswerk](#)

Je hebt 1 vraagmoment. 1 moment over.

[Ga naar stel een vraag](#) [Markeer als deelgenomen aan deze les](#)

Inhoudsopgave

- 8.1** Inleiding
- 8.2** Wat zijn functies?
- 8.3** Door de gebruiker gedefinieerde functies
- 8.4** Structuur van een functie
- 8.5** Een functie met een teruggeefwaarde
- 8.6** Een functie met parameters
- 8.7** Lokale en globale variabelen
- 8.8** Project 'IJssalon'
- 8.9** Samenvatting
- 8.10** Vraagmoment
- 8.11** Oefenopgaven
- 8.12** Huiswerkopgaven

8.1 Inleiding

In deze les staat het onderwerp 'functies' centraal. Dat lijkt wellicht een nieuw onderwerp, maar toch heeft u eerder in deze cursus al verschillende keren met functies gewerkt. Neem bijvoorbeeld de print-functie:

```
print("Hallo wereld")
```

Maar ook het volgende is een functie:

```
l = len(Mijn_lijst)
```

In deze les duiken we dieper in de wereld van functies. Daarnaast gaat u zelf functies maken.

Leerdoelen

Aan het einde van deze les weet u:

- wat een functie is;
 - dat Python vooraf gedefinieerde functies bevat;
 - hoe u zelf functies kunt definiëren;
 - de voordelen van persoonlijke functies;
 - de opbouw van een functie;
 - het doel van het inspringen van code;
 - dat u met functies bewerkingen kunt laten uitvoeren (met bepaalde waarden als uitvoer);
 - dat functies ook parameters kunnen bevatten;
 - hoe u variabelen definieert in een functie;
 - het verschil tussen lokale en globale variabelen;
 - hoe u een functie importeert vanuit een ander bestand.



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

8.2 Wat zijn functies?

U heeft immiddels al de nodige regels code geschreven. Iedere regel die u schreef, had een doel: het uitvoeren van een bepaalde taak.

Functies doen eigenlijk precies hetzelfde. Het grootste verschil is echter dat functies bestaan uit meerdere regels code die bij elkaar horen en die u als programmeur vaak nodig heeft. Door de regels te groeperen in een functie, kunt u ze eenvoudig hergebruiken. In plaats van het schrijven van complete regels code, worden ze aangeroepen met een enkel commando.



In sommige programmeertalen worden functies ook wel 'procedures' genoemd.

Neem de print-functie. Die roept u weliswaar aan met het simpele commando print, maar er zitten meerdere regels code achter dit commando. Terwijl u maar één ding ziet gebeuren (er wordt iets op het scherm getoond), gebeurt er achter de schermen veel meer voordat het zover is.

The screenshot shows a Visual Studio Code interface. On the left is the Explorer sidebar with a file tree containing 'les3.py' under 'LES 3'. The main editor area has the same 'les3.py' file open with the single line of code: `1 print("Hello World")`. Below the editor is the Terminal panel, which is active and displays the command `C:\Users\MSI\Documents\WHA-prg\Les 3>python les3.py` followed by the output `Hello World`. The terminal tab bar also shows 'PROBLEMS', 'OUTPUT', and 'DEBUG CONSOLE'. The status bar at the bottom indicates the environment is 'Python 3.9.4 64-bit'.

Doordat `print()` een functie is, gebeurt er achter de schermen veel meer dan u ziet!

De print-functie is een zogenaamde **voorgedefinieerde functie**. Dat houdt in dat deze functie standaard in Python is ingebakken. Als u Python opstart, bestaat deze functie dus al.

Naast print() bevat Python nog veel meer van zulke voorgedefinieerde functies:

abs()	compile()	format()	isinstance()	object()	setattr()
all()	complex()	frozenset()	issubclass()	oct()	slice()
any()	delattr()	getattr()	iter()	open()	sorted()
ascii()	dict()	globals()	len()	ord()	staticmethod()
bin()	dir()	hasattr()	list()	pow()	str()
bool()	divmod()	hash()	locals()	property()	sum()
bytearray()	enumerate()	help()	map()	range()	super()
bytes()	eval()	hex()	max()	repr()	tuple()
callable()	exec()	id()	memoryview()	reversed()	type()
chr()	filter()	input()	min()	round()	vars()
classmethod()	float()	int()	next()	set()	zip()

Enkele van deze functies hebben we al eens genoemd, zoals abs() en pow(). Enkele andere functies zullen in deze en volgende lessen nog aan bod komen. Voor meer informatie over de andere functies verwijzen we u graag naar www.w3schools.com/python/python_ref_functions.asp.

Veel interessanter dan de voorgedefinieerde functies zijn functies die u zelf heeft gedefinieerd. Daar gaan we in de volgende paragraaf dieper op in.

8.3 Door de gebruiker gedefinieerde functies

Met 'door de gebruiker gedefinieerde functies' bedoelen we dus functies die u zelf heeft toegevoegd. Alleen u kunt ze gebruiken en ze sluiten perfect aan op uw situatie. Maatwerk dus!

Iedere moderne programmeertaal maakt het mogelijk om zelf functies te definiëren, zo ook Python. Kijkt u maar eens naar dit voorbeeld:

- 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 8).
- Klik in de *Menu Bar* op "File".
- Klik op "Open Folder".
- Selecteer de map die u zojuist als laatste aanmaakte (Les 8).
- 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 volgende code in het bestand:

```
def print_10():
    for i in range(10):
        print(i + 1)

print_10()
```

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

Het resultaat ziet er dan als volgt uit:

The screenshot shows a terminal window with the following interface elements:

- Top bar: PROBLEMS, OUTPUT, TERMINAL (underlined), DEBUG CONSOLE
- Right side: cmd, +, -, ^, X

The terminal window displays the following text:

```
C:\Users\MSI\Documents\NHA-prg\Les 8>python test.py
1
2
3
4
5
6
7
8
9
10
```

Below the terminal window, the command prompt is shown again: C:\Users\MSI\Documents\NHA-prg\Les 8>

De getallen 1 tot en met 10 zijn een voor een uitgeprint, telkens met een tussenstap van + 1.

De eerste drie regels van uw code vormen de functie:

```
def print_10():
    for i in range(10):
        print(i + 1)
```

Met de vierde regel roept u de functie aan:

```
print_10()
```

Dat aanroepen werkt voor iedere functie op dezelfde manier: u begint met de naam van de functie, gevolgd door een haakje openen en een haakje sluiten: naamvandefunctie() dus. Dit lijkt misschien een omweg, omdat u twee stappen moet zetten: eerst de functie definiëren en daarna de functie oproepen. U zult zich afvragen: waarom laten we het niet gewoon bij de eerste drie regels en voeren we die uit in de *Terminal*? Het antwoord is simpel: **de kracht van herhaling**.

Eerder in deze les gaven we al aan dat u regels groepeert in een functie om ze op een gemakkelijke manier te kunnen hergebruiken. Zou u de getallen van 1 tot en met 10 maar één keer willen printen in uw hele code, dan is het inderdaad zonde om er een functie van te maken. Heeft u deze getallenreeks vaker nodig, dan is het juist zonde om deze drie regels iedere keer opnieuw te typen en komt de functie print_10() van pas. We zullen dit bewijzen aan de hand van een ander voorbeeld.

Tot nu printten we iedere keer vrij korte tekstjes op het scherm. Maar stelt u zich eens voor dat dit erg lange teksten zouden zijn: dan zou de uitvoer naar het scherm uiteindelijk ook erg lang zijn. Het plaatsen van een scheidingslijn tussen iedere uitvoer zorgt dan voor wat overzicht. De regels code die ervoor zorgen dat de scheidingslijn in beeld komt, zou u iedere keer opnieuw kunnen programmeren:

```
print()
print(-----)
print()
```

U kunt er echter ook een functie van maken:

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

```
def devider():
    print()
    print("-----")
    print()
```

- Typ daaronder de code waarin u deze functie wilt gebruiken:

```
print("Dit is mijn eerste code")
devider()
print("Dit is mijn tweede code")
devider()
print("Dit is mijn derde code")
```

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

De uitvoer ziet er dan zo uit:

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 8>python test.py
Dit is mijn eerste code
-----
Dit is mijn tweede code
-----
Dit is mijn derde code

C:\Users\MSI\Documents\NHA-prg\Les 8>
```

Het maakt voor de uitvoer niet uit of u de persoonlijke functie `devider()` gebruikt of op die plek de drie regels code typt die achter deze functie schuilgaan. Toch zal iedere ervaren programmeur de voorkeur geven aan het gebruik van de functie. Waarom?

- In dit geval lijkt de uitvoer overzichtelijk, maar wat als de teksten langer zijn dan `Dit is mijn ___ code`? En wat als er meer dan drie teksten worden toegevoegd?

Het volledig uitschrijven van de code (in plaats van het gebruik van functies) neemt niet alleen tekstruimte in, maar ook geheugenruimte. In veel gevallen zorgt het gebruik van functies er daardoor voor dat een programma sneller is.

- Het 'opschalen van code' (lees: het toevoegen van meer paragrafen) is niet alleen veel gemakkelijker als u een functie gebruikt. Het zorgt er ook voor dat er minder kans is op het maken van fouten.
- Stel dat u op een later moment besluit geen streepjes als scheidingstekenen te gebruiken, maar `--`-tekens. Als u een functie heeft gebruikt, hoeft u de code slechts op één plek aan te passen. Opnieuw geldt: dit is sneller, maar ook minder foutgevoelig.
- Zodra u een handige functie heeft gedefinieerd, kunt u die ook importeren naar een eventueel volgend programma dat u gaat coderen. U hoeft de code achter een functie in feite dus maar één keer helemaal uit te typen.

Dit zijn slechts enkele voordelen die verbonden zijn aan het gebruik van functies. Er zijn er nog veel meer, maar die vallen buiten het bestek van deze cursus.

8.4 Structuur van een functie

In de vorige paragraaf definieerde we al twee persoonlijke functies. Dat deden we telkens op dezelfde manier:

1. Het commando om een functie te starten (`def`), gevolgd door een spatie
2. De naam van de functie
3. Haakje openen
4. Haakje sluiten
5. Dubbele punt

Dus:

1	2	3	4	5
<code>def</code>	<code>print_10</code>	<code>(</code>	<code>)</code>	<code>:</code>
<code>def</code>	<code>devider</code>	<code>(</code>	<code>)</code>	<code>:</code>

Na de dubbele punt volgen er enkele regels met een inspringing, bijvoorbeeld:

```
def devider():
    print()
```

```
print("-----")
print()
```

Als u in VS Code werkt, wordt die inspringing automatisch toegevoegd wanneer u na de dubbele punt op Enter drukt.

Deze inspringing heeft een belangrijke betekenis:

Elke regel na de dubbele punt die deze inspringing heeft, is onderdeel van de functie.

Met andere woorden: de eerste regel die deze inspringing niet heeft, maakt geen deel meer uit van de functie (maar behoort tot de 'reguliere' code).

```
def voorbeeld_functie():
    print("Onderdeel van de functie")
    print("Onderdeel van de functie")
    print("Onderdeel van de functie")
    print("Onderdeel van de functie")

    print("Geen onderdeel van de functie, maar reguliere code.")
```

Alles wat na de dubbele punt inspringt, hoort bij de functie.

8.5 Een functie met een teruggeefwaarde

De twee functies die we hiervoor definiereiden, zorgden ervoor dat er een bepaalde bewerking werd uitgevoerd (in beide gevallen: het printen van een bepaalde tekst).

U kunt een functie ook gebruiken om een bepaalde waarde te laten 'teruggeven'. Daarvoor gebruikt u het sleutelwoord **return**. Bijvoorbeeld:

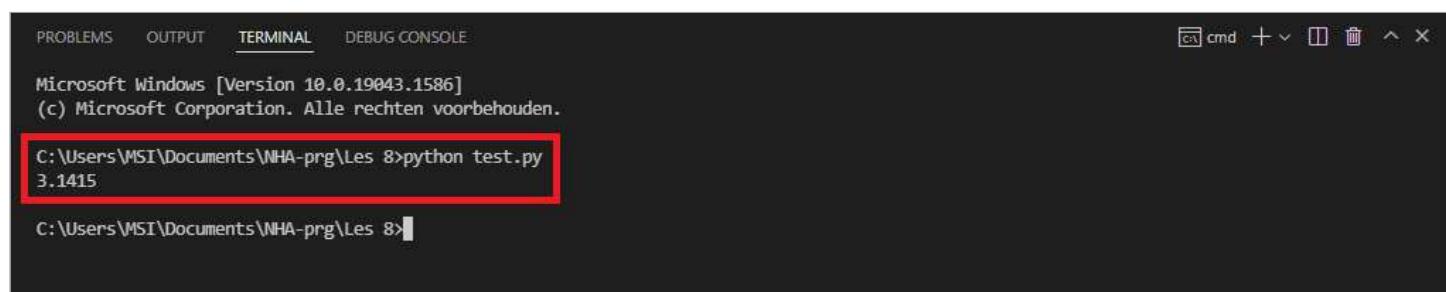
- Vervang de code in het bestand door de volgende code:

```
def ongeveer_pi():
    return 3.1415

print(ongeveer_pi())
```

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

Het resultaat is dan:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
cmd + ×

Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 8>python test.py
3.1415
C:\Users\MSI\Documents\NHA-prg\Les 8>
```

U ziet dat de functie niets anders doet dan het getal dat we hebben opgegeven, 'teruggeven'. (In dit geval is het een benadering van het getal 'pi'.)

Hier kozen we ervoor om **return** te gebruiken voor het teruggeven van een getal, maar u kunt nog veel meer teruggeven: *strings*, *lists*, *dictionaries*, enzovoorts.

8.6 Een functie met parameters

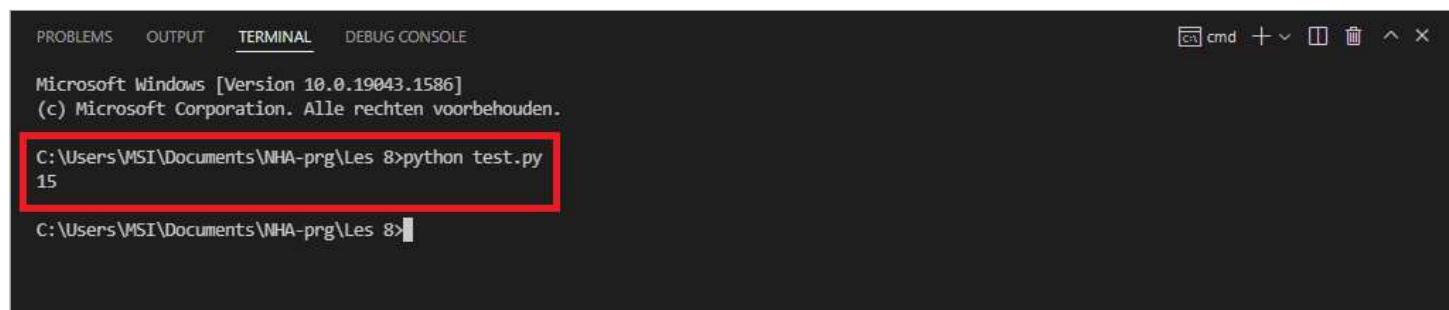
Achter de naam van de functie staan twee haakjes (haakje openen en haakje sluiten). Die haakje staan er met een reden, ze hebben een functie: tussen deze haakjes kunt u waarden plaatsen op het moment dat u de functie aanroeft. Dat kan echter alleen als de functie daarop rekent (lees: als u dat vooraf heeft aangegeven). Bijvoorbeeld:

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

```
def tel_op(a,b):  
    return a + b  
  
totaal = tel_op(5,10)  
  
print(totaal)
```

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

Het resultaat is dan:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE  
Microsoft Windows [Version 10.0.19043.1586]  
(c) Microsoft Corporation. Alle rechten voorbehouden.  
C:\Users\MSI\Documents\WHA-prg\Les 8>python test.py  
15  
C:\Users\MSI\Documents\WHA-prg\Les 8>
```

De letters a en b, die tussen haakjes staan, noemen we **parameters**. Zodra we de functie aanroepen met waarden, noemen we die waarden **argumenten** (hier: 5 en 10).

Zoals u ziet, gebruikt u parameters in een functie op dezelfde manier als 'gewone' variabelen. U kunt er allerlei bewerkingen mee uitvoeren.

Om te laten zien dat we niet alleen maar getallen als argument mee kunnen geven, geven we nog een voorbeeld:

```
def info(naam, leeftijd, in_dienst):  
    if in_dienst:  
        text_1 = "en nog altijd in dienst van onze firma."  
    else:  
        text_1 = "en niet meer bij ons in dienst."  
  
    uitvoer = f"Beste {naam}, u bent {leeftijd} jaar " + text_1  
    return uitvoer  
  
print(info("Harry", 54, True))  
print(info("Magda", 73, False))
```

De uitvoer zal dan zijn:

```
Beste Harry, u bent 54 jaar en nog altijd in dienst van onze firma.  
Beste Magda, u bent 73 jaar en niet meer bij ons in dienst.
```

Foutieve invoer

Als u aangeeft dat er sprake is van parameters, rekent de functie daarop. Vergeet u echter de argumenten door te geven tijdens het aanroepen van de functie, dan zult u in de *Terminal* een foutmelding krijgen:

Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 8>python test.py
Traceback (most recent call last):
  File "C:\Users\MSI\Documents\NHA-prg\Les 8\test.py", line 4, in <module>
    totaal = tel_op()
TypeError: tel_op() missing 2 required positional arguments: 'a' and 'b'
```

```
C:\Users\MSI\Documents\NHA-prg\Les 8>
```

De foutmelding geeft aan hoeveel argumenten er missen (hier: 2) en welke argumenten er precies missen (hier 'a' en 'b'). U zult een vergelijkbare foutmelding krijgen als u de functie aanroeft met maar één argument.

Maar... u hoeft de argumenten niet in alle gevallen door te geven bij het aanroepen van de functie. U zou ze ook al kunnen doorgeven bij het definiëren van de functie. Bijvoorbeeld:

```
def tel_op(a=1,b=2):
    return a + b

totaal = tel_op()

print(totaal)
```

Bij deze methode kent u standaardwaarden toe aan de parameters. Zodra u de functie aanroeft zonder argumenten, worden deze standaardwaarden gebruikt. De uitkomst zal dan 3 zijn (en geen foutmelding).

Iets vergelijkbaars gebeurt er als u standaardwaarden opgeeft en u de functie aanroeft met één argument. Bijvoorbeeld:

```
def tel_op(a=1,b=2):
    return a + b

totaal = tel_op(20)

print(totaal)
```

Het ene argument dat u opgeeft (20), wordt toegekend aan parameter a. Hiermee wordt de standaardwaarde van a (1) overschreven.

U heeft geen tweede argument opgegeven, dus voor b wordt de standaardwaarde gebruikt (2). Het resultaat is dus 22 (en geen foutmelding).

8.7 Lokale en globale variabelen

Een functie kan dus waarden meekrijgen in de vorm van argumenten en een waarde teruggeven. Maar let op:

De variabelen die u definieert binnen in een functie, kunt u daarbuiten niet meer gebruiken!

Bekijkt u maar eens het volgende voorbeeld:

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

```
totaal = 0

def mijn_functie():
    a = 2

    totaal = totaal + a
    print(totaal)
```

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

Het resultaat is dan een foutmelding:

The screenshot shows a terminal window with the following text:

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\WHA-prg\Les 8>python test.py
Traceback (most recent call last):
  File "C:/Users/MSI/Documents/WHA-prg/Les 8/test.py", line 6, in <module>
    totaal = totaal + a
NameError: name 'a' is not defined

C:\Users\MSI\Documents\WHA-prg\Les 8>
```

A red box highlights the error message: "NameError: name 'a' is not defined".

De foutmelding geeft aan dat de a die genoemd wordt in regel 6, niet gedefinieerd is. Dat lijkt niet te kloppen, want op regel 4 staat toch gewoon a = 2? De foutmelding is echter toch terecht, want a = 2 is ingesprongen (en dus onderdeel van een functie). Regel 6 is niet ingesprongen en hoort dus niet meer bij de functie. Omdat u a al gebruikt heeft in een functie (regel 4), kunt u a niet meer gebruiken buiten de functie (regel 6).

Andersom kan wel:

De variabelen die u gebruikt in de 'hoofdcode', kunt u wél gebruiken in een functie.

U moet dan echter wel aangeven dat deze variabele global is. Daarmee geeft u aan dat de variabele zowel buiten als binnen de functie 'zichtbaar' is en gebruikt kan worden. Een voorbeeld:

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

```
b = 2
def mijn_functie():
    global b
    b = b + 10
    print("binnen: ", b)

print("buiten: ", b)
mijn_functie()
print("buiten: ", b)
```

Deze code is als volgt opgebouwd:

Regel	Betekenis
1	U geeft variabele b de waarde 2.
2 - 6	U definieert de functie mijn_functie.
7	U print de waarde van b.
8	U roept de functie aan.
9	U print opnieuw de waarde van b.

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

Het resultaat is dan:

A screenshot of a terminal window titled 'cmd'. The window shows the following text:

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 8>python test.py
buiten: 2
binnen: 12
buiten: 12

C:\Users\MSI\Documents\NHA-prg\Les 8>
```

Omdat in de functie wordt aangegeven dat b een global-variabele is, kan de waarde ervan uitgelezen worden en veranderd worden (en dat gebeurt in ons voorbeeld ook: we tellen er 10 bij op). Daardoor is de waarde van b de eerste keer dat u hem print 2 (regel 7), door het aanroepen van de functie, veranderd in 12 (regel 8). Die waarde wordt door de functie geprint, wat wordt herhaald door de code op regel 9.

Het werken met variabelen is in de praktijk niet gemakkelijk. Hoe meer u oefent met zowel lokale als globale variabelen, hoe meer u doorkrijgt hoe ze werken en hoe gemakkelijker u ze kunt toepassen.

8.8 Project 'IJssalon'

Nu u weet wat functies zijn en hoe u ze kunt gebruiken in uw code, is het tijd om ze toe te passen in een casus. Daarbij zal ook duidelijk worden dat u reeds gedefinieerde functies kunt importeren naar andere bestanden.

8.8.1 Bestaande code in een functie plaatsen

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

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

```
1 prijzen = {
2     "aardbei" : 3,
3     "vanille" : 4,
4     "chocolade" : 5
5 }
6
7 aanbieding = prijzen["aardbei"] * 0.8
8
9 reclame_tekst = f"Vandaag in de aanbieding: vanille-ijs, 1 liter - slechts € {aanbieding}"
10
11 reclame_tekst2 = reclame_tekst[:63]
12
13 reclame_tekst3 = reclame_tekst2.upper()
14
15 reclame_tekst4 = reclame_tekst3.split()
16
17 for el in reclame_tekst4:
18     if len(el) > 4:
19         print(el.upper())
20     else:
21         print(el.lower())
```

- Plaats de cursor voor prijzen (regel 1).
- Druk op Enter.
- Plaats de volgende code op de nu lege regel 1:

```
def print_aanbieding():
```

- Selecteer de rest van de code.
- Druk op de Tab-toets.

Doordat u op de Tab-toets drukt, springt de geselecteerde code in. Dat betekent dat deze code nu dus binnen de functie valt!

```
1 def print_aanbieding():
2     prijzen = {
3         "aardbei": 3,
4         "vanille": 4,
5         "chocolade": 5
6     }
7
8     aanbieding = prijzen["aardbei"] * 0.8
9
10    reclame_tekst = f"Vandaag in de aanbieding: vanille-ijs, 1 liter - slechts € {aanbieding}"
11
12    reclame_tekst2 = reclame_tekst[:63]
13
14    reclame_tekst3 = reclame_tekst2.upper()
15
16    reclame_tekst4 = reclame_tekst3.split()
17
18    for el in reclame_tekst4:
19        if len(el) > 4:
20            print(el.upper())
21        else:
22            print(el.lower())
```

De gehele code valt nu binnen de functie print_aanbieding.

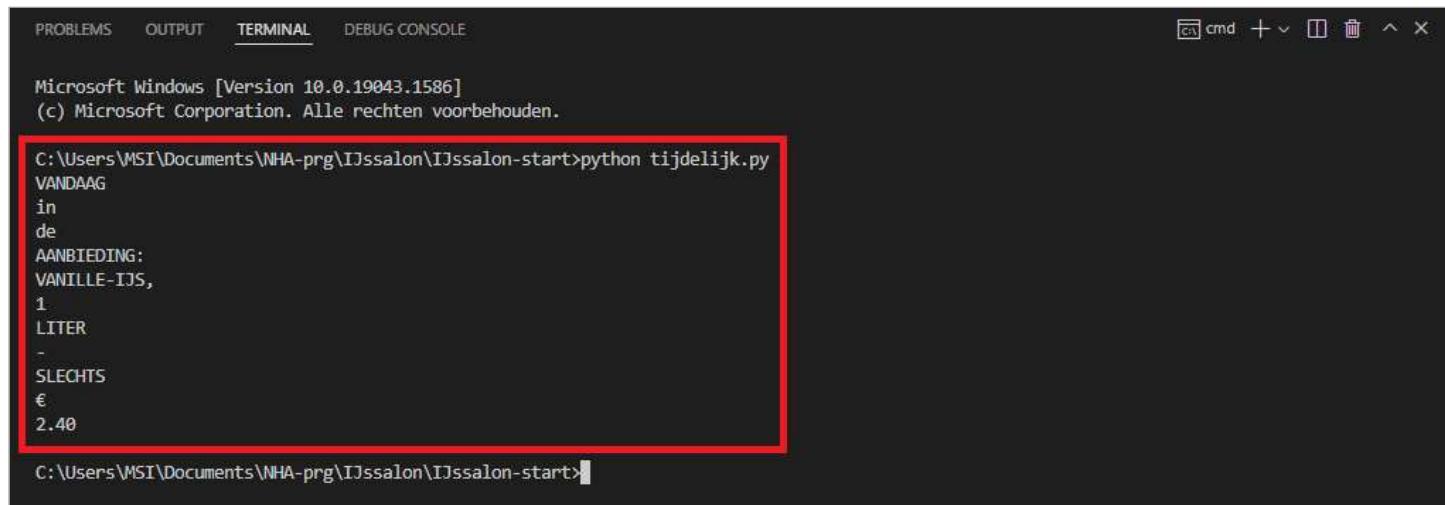
- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

U ziet dat er geen uitvoer plaatsvindt. Geen paniek, dat komt doordat we u bewust een fout lieten maken. U heeft namelijk wel een functie gedefinieerd, maar u heeft deze functie niet aangeroepen! Zolang dat niet gebeurt, zal de functie nooit worden uitgevoerd. Dat is nu eenmaal hoe functies werken.

We gaan de functie `alsnog` aanroepen.

- Plaats de cursor na het laatste haakje sluiten op regel 22.
- Druk twee keer op Enter.
- Druk drie keer op de Backspace-toets.
- Typ `print_aanbieding()` om de functie aan te roepen.
- Sla de wijzigingen op.
- Voer het programma uit in de *Terminal*.

Nu worden de aanbiedingen wél uitgeprint:



8.8.2 Een functie gebruiken in een ander bestand

De functie die u heeft gedefinieerd in het bestand 'tijdelijk.py' gaan we gebruiken in het bestand 'helper.py'.

- Open het bestand ‘helper.py’.

Op de eerste regel van dit bestand staat nu nog deze code:

tekst="header"

- Vervang deze regel code door:

```
def decoreer(tekst=""):
```

- Selecteer de rest van de code.
 - Druk op de Tab-toets.

Doordat u op de Tab-toets drukt, springt de geselecteerde code in. Dat betekent dat deze code nu dus binnen de functie valt!

```
1 def decoreer(tekst=""):
2     lengte = len(tekst) + 4
3     print()
4     print(lengte * "*")
5     print(f"* {tekst} *")
6     print(lengte * "*")
7     print()
```

De gehele code valt nu binnen de functie `decorree`.

We willen deze functie def decoreer(tekst""), die zich bevindt in het bestand 'helper.py', gebruiken in het bestand 'tijdelijk.py'. Daarvoor moeten we de functie importeren (in het bestand 'tijdelijk.py').

- Open het bestand ‘tijdelijk.py’.

Het importeren van functies of variabelen vanuit een ander bestand naar het huidige wordt altijd gedaan aan het begin van een bestand. We maken daarom allereerst ruimte om de functie toe te voegen (twee regels is genoeg).

- Plaats de cursor voor def (regel 1).
 - Druk twee keer op Enter.
 - Plaats de volgende code op de nu lege regel 1:

```
from helper import decoreer
```

Door het toevoegen van deze code kunt u de functie decoreer gebruiken waar u maar wilt (in het bestand 'tijdelijk.py').

- Plaats de cursus voor de code `print_aanbieding()`: (op regel 26).

- Druk op Enter.

- Plaats de volgende code op de nu lege regel 26:

```
decorer("Aanbieding")
```

- Sla de wijzigingen op.

- Voer het programma uit in de *Terminal*.

U ziet dat er dankzij de functie `decorer` een fraaie *header* boven de reclameteksten is gegenereerd:

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\IJssalon\IJssalon-start>python tijdelijk.py

*****
* Aanbieding *
*****

VANDAAG
in
de
ANBIEDING:
VANILLE-IJJS,
1
LITER
-
SLECHTS
€
2.40

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

In dit geval wellicht niet het meest functionele voorbeeld, maar u weet nu in elk geval wel hoe u functies uit andere bestanden kunt gebruiken.

8.8.3 Versie bijwerken en pushen naar GitHub

Nu u weer wat code heeft toegevoegd en veranderd, is het zaak een nieuwe lokale versie van het project aan te maken.

- Ga naar een nieuwe regel in de *Terminal*.
- Typ `git add .` en druk op Enter.
- Typ `git commit -m "functies toegevoegd"` en druk op Enter.
- Typ `git push origin main` en druk op Enter.

Na die laatste opdracht krijgt u een reeks regels met informatie te zien. Ziet u hier geen foutmeldingen, dan is uw nieuwe versie succesvol geüpdateerd en naar *GitHub* gestuurd. U kunt dat eventueel nog controleren op *GitHub*:

- Typ `git config --get remote.origin.url` en druk op Enter.

Dit levert u een URL op.

- Open een browser (bijvoorbeeld *Google Chrome*).
- Bezoek de URL die u terugkreeg in de *Terminal*.

U ziet dan dat de *repository* is bijgewerkt naar de laatste versie.

8.9 Samenvatting

In deze les maakte u kennis met functies, een belangrijk onderdeel van programmeertalen. Functies zijn zelfs zo belangrijk, dat het moeilijk is om een programma te schrijven waarin geen functies worden gebruikt.

U kunt voorprogrammeerde functies gebruiken, maar ook zelf functies definiëren. In het laatste geval bouwt u in feite zelf commando's, waarmee u een bepaalde bewerking kunt laten uitvoeren. Ook kunt u een bepaalde waarde laten teruggeven. Wat het ook wordt: in deze les leerde u hoe u een functie opbouwt.

U zag ook dat functies parameters kunnen accepteren. Door argumenten mee te geven wanneer u een functie aanroeft, voegt u extra informatie toe aan de functie. De functie zal deze input dan gebruiken voor de berekening van een teruggeefwaarde of anderszins gebruikmaken van deze data.

Vervolgens hebben we ingezoomd op het gebruik van variabelen in een functie. U heeft gezien dat de variabelen die u in een functie gebruikt, standaard lokale variabelen zijn. Dat betekent dat ze buiten de functie niet bekend zijn. Daar kunt u verandering in brengen met behulp van het commando `global`.

Ten slotte heeft u uw kennis van functies toegepast in het overkoepelende project 'IJssalon'.

8.10 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 spreekballoon.

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.

8.11 Oefenopgaven

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

Hieronder ziet u drie keer een poging om een functie te definiëren. Geef per poging aan wat er niet klopt.

- a. `function mijn_functie():`
- b. `def mijn_functie:`
- c. `def mijn_functie(`:

[Bekijk antwoord](#)

Bekijk de volgende code:

```
b = 15
def mijn_functie():
    global b
    b = b + 10
    print("binnen: ", b)

print("buiten: ", b)
mijn_functie()
print("buiten: ",b)
```

Op welke regel begint de functie? Op welke regel staat de laatste regel van de functie?

→ Bekijk antwoord

Bekijk de volgende code:

```
b = 15
def mijn_functie():
    global b
    b = b + 10
    print("binnen: ", b)

print("buiten: ", b)
mijn_functie()
print("buiten: ", b)
```

Wat zal de uitvoer zijn wanneer deze code wordt uitgevoerd?

→ Bekijk antwoord

Definieer een functie met de naam `mijn_functie`, die één parameter bevat (met de naam `mijn_int`). De functie verdubbelt de parameter en geeft de verdubbelde waarde als teruggeefwaarde wanneer de functie wordt aangeroepen.

(U mag ervan uitgaan dat als argument voor `mijn_int` een `integer` wordt meegegeven.)

→ Bekijk antwoord

Definieer een functie met de naam `mijn_functie`, die één parameter bevat (met de naam `mijn_lijst`). De functie loopt door de `list` loopt, telt alle elementen bij elkaar op en geeft het totaal als teruggeefwaarde.

(U mag ervanuit gaan dat als argument voor `mijn_lijst` een `list` met `integers` wordt meegegeven.)

→ Bekijk antwoord

Definieer een functie met de naam `mijn_functie`, die twee parameters bevat (met de namen `string1` en `string2`). De teruggeefwaarde van deze functie zou een `string` moeten zijn die alle letters van `string1` bevat die ook voorkomen in `string2`.

(U mag ervan uitgaan dat de argumenten die bij het aanroepen van de functie worden meegegeven, inderdaad `strings` zijn met kleine letters, zonder interpunctie).

→ Bekijk antwoord

Bekijk de volgende code:

```
a = 10

def maal_drie(a):
    print(a)
    return a * 3
```

Wat zal de uitvoer zijn wanneer deze code wordt uitgevoerd?

→ Bekijk antwoord

Bekijk de volgende code:

```
a = 10
b = 20

def maal_drie(b):
    return b * 3

print(maal_drie(a))
```

Wat zal de uitvoer zijn wanneer deze code wordt uitgevoerd?

→ Bekijk antwoord

Bekijk de volgende code:

```
a = 10
b = 20

def maal_drie(a):
    global b
    return b * 3

print(maal_drie(20))
```

Wat zal de uitvoer zijn wanneer deze code wordt uitgevoerd?

→ Bekijk antwoord

Bekijk de volgende code:

```
a = 10

def maal_drie(a):
    return a * 3

def mijn_functie(a):
    a = maal_drie(a)
    uitvoer = maal_drie(a)
    return uitvoer * 2

print(mijn_functie(20))
```

Wat zal de uitvoer zijn wanneer deze code wordt uitgevoerd?

→ Bekijk antwoord

8.12 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. 1. Voeg een nieuw bestand toe aan de map 'IJssalon-start'. Geef deze map de naam `algemene_functies.py`.
2. Bekijk de volgende tabel:

Argumenten	Teruggeefwaarde
2	4
4	16
10	100
12	144

Plaats in het aangemaakte bestand een functie genaamd `mijn_functie_1()`, die wanneer een argument uit de eerste kolom van de tabel wordt gebruikt, het bijbehorende getal uit de tweede kolom teruggeeft.



Kijk goed naar de rekenkundige bewerking die u moet uitvoeren om de teruggeefwaarde als uitkomst te krijgen.

3. Bekijk de volgende tabel:

Argumenten	Teruggeefwaarde
12,3	[15, 9, 36, 4]
12,2	[14, 10, 24, 6]
10,5	[15, 5, 50, 2]
100,20	[120, 80, 2000, 5]

Plaats in het bestand 'algemene_functies.py' een functie genaamd `mijn_functie_2()`, die wanneer een argument uit de eerste kolom van de tabel wordt gebruikt, het bijbehorende getal uit de tweede kolom als teruggeefwaarde toont.



Kijk goed naar de rekenkundige bewerking die u moet uitvoeren om de teruggeefwaarde als uitkomst te krijgen.

4. Voeg een nieuw bestand toe aan de map 'IJssalon-start'. Geef deze map de naam `reclame.py`.

5. Plaats in het bestand 'reclame.py' een functie genaamd `aanbieding_1()`, die drie parameters bevat: `smaak`, `prijs` en `korting`. Wanneer deze functie wordt aangeroepen met de argumenten `aardbei`, `4` en `0.1`, zou de teruggeefwaarde van de functie moeten zijn:

Vandaag in de aanbieding: emmertje ijs (1 liter) in de smaak aardbei, van 4 euro voor 3,60 euro.

6. Plaats in het bestand 'reclame.py' een functie genaamd `inkomsten_totaal()`, die één parameter bevat, genaamd `inkomsten`. Als argument wordt een lijst met zeven waarden meegegeven, namelijk de (fictieve) inkomsten per dag van deze week (bijvoorbeeld `[220, 430, 125, 160, 205, 90, 345]`). De teruggeefwaarde moet het totaal van alle bedragen zijn.

7. Verander de functie `inkomsten_totaal()` zodanig, dat een extra argument meegegeven kan worden (btw). Btw is een `float` (bijvoorbeeld `0.09` voor 9% btw).

De teruggeefwaarde zou een `string` moeten zijn met de volgende inhoud:

Het totaal van alle inkomsten van deze week is <totaal> euro, waarover <bedrag> euro btw betaald dient te worden.

Waar nu <totaal> staat, moet het totaal van deze week worden weergegeven. Waar nu <bedrag> staat, komt de btw in euro's.

8. Plaats in het bestand 'reclame.py' een functie genaamd `laag_en_hoog()`, die één parameter bevat, genaamd `mijn_lijst`. Als argument wordt een lijst met zeven waarden meegegeven, namelijk de (fictieve) inkomsten per dag van deze week (bijvoorbeeld [220, 430, 125, 160, 205, 90, 345]). De teruggeefwaarde moet een lijst zijn met slechts twee elementen: de hoogste en de laagste waarde.



Gebruik hiervoor de standaardfuncties `max()` en `min()`. Meer informatie over deze functies vindt u op www.w3schools.com/python/python_ref_functions.asp.

9. Plaats in het bestand 'reclame.py' een functie genaamd `gemiddelde()`, die één parameter bevat, genaamd `mijn_lijst`. Als argument wordt een lijst met zeven waarden meegegeven, namelijk de (fictieve) inkomsten per dag van deze week (bijvoorbeeld [220, 430, 125, 160, 205, 90, 345]). De teruggeefwaarde moet het gemiddelde van deze zeven waarden zijn.

10. Verander de functie `gemiddelde()` zodanig, dat de teruggeefwaarde een *string* is met de volgende inhoud:

De gemiddelde inkomsten deze week zijn <bedrag> euro.

Waar nu <bedrag> staat moet het totaalbedrag van deze week worden weergegeven.

11. Plaats in het bestand 'reclame.py' een functie genaamd `meervoudig()`, die één parameter bevat, genaamd `invoer_lijst`. Als argument wordt een lijst van tussen de vijf en tien integers lang meegegeven (bijvoorbeeld [10,5,3,2,1,2,9]).

Als teruggeefwaarde geeft de functie `meervoudig()` een lijst terug van de hoogste waarde in de lijst en de laagste waarde. U dient hierbij gebruik te maken van de functie `hoog_en_laag()`.

12. Importeer op de eerste regel van het bestand 'aanbieding.py' de functie `mijn_functie_2` uit het bestand 'algemene_functies.py'.

Plaats in het bestand 'reclame.py' een functie genaamd `combinatie()`, die één parameter bevat, genaamd `invoer_lijst_2`. In deze functie roept u de functie `meervoudig()` aan, met als argument de parameter `invoer_lijst_2`.

De teruggeefwaarde van deze functie dient te worden opgeslagen in variabele `korte_lijst`. Deze korte lijst dient als argument gebruikt te worden bij het aanroepen van functie `mijn_functie_2`. De teruggeefwaarde die hierdoor gegenereerd wordt, is de teruggeefwaarde van functie `combinatie()`.

13. Update uw *remote repo* op GitHub.

14. Stuur de URL van uw *remote repo* op GitHub via Plaza naar uw docent. Die kan dan niet alleen het eindresultaat zien, maar ook alle tussentijdse wijzigingen.