

Les 6 - Werken met gegevens

« Vorige les

» Volgende les

Huiswerk

Programmeren voor Beginners - Les 6

Ingediend

Ingediend op 27 april 2023

[Ga naar huiswerk](#)

Je hebt 1 vraagmoment. 1 moment over.

[Ga naar stel een vraag](#)

Inhoudsopgave

6.1	Inleiding
6.2	Lists
6.3	Dictionaries
6.4	Operatoren
6.5	Samenvatting
6.6	Vraagmoment
6.7	Oefenopgaven
6.8	Huiswerkopgaven

6.1 Inleiding

Er is geen enkel computerprogramma denkbaar dat niet op een of andere manier gebruikmaakt van gegevens (ook wel **data** genoemd). Het is daarom belangrijk dat we in deze cursus aandacht besteden aan **datamanipulatie**.

Als we het hebben over datamanipulatie, is CRUD een belangrijke term. De vier letters van deze afkorting verwijzen naar vier datamanipulatieconcepten:

C	Create	Data creëren
R	Retrieve	Data lezen (of 'ophalen')
U	Update	Data wijzigen
D	Delete	Data verwijderen

Als we deze vier bewerkingen kunnen uitvoeren op onze data, hebben we volledige controle over deze data.

In de vorige les leerde u dat we data (tijdelijk) bewaren in variabelen. Weet u het nog? Variabelen zijn een soort ‘emmers’ waarin we waarden kunnen opslaan.

In beginsel heeft elke variabele één waarde. Het is echter ook mogelijk om meerdere waarden op te slaan in een variabele.

Het opslaan van meerdere waarden in een enkele variabele kan op verschillende manieren. We zullen in deze les twee veelgebruikte manieren aan de orde stellen, namelijk de *list* en de *dictionary*.

We gaan in deze les ook dieper in op het programmeerconcept ‘operator’. U leert wat operatoren zijn en hoe we operatoren gebruiken om data te bewerken.

Leerdoelen

Aan het einde van deze les weet u:

- wat een *list* is en hoe u een *list* creëert;
- wat een index is;
- hoe u een of meer elementen uit deze *list* haalt (en print);
- hoe u *lists* combineert met iteratie;
- hoe u achteraf elementen toevoegt aan deze *list* (of verwijdert);
- wat een *dictionary* is;
- hoe u een *dictionary* creëert;
- wat een sleutel is;
- hoe u een of meer elementen uit dit *dictionary* haalt (en print);
- hoe u *dictionaries* combineert met iteratie;
- hoe u achteraf elementen toevoegt aan een *dictionary* (of verwijdert);
- hoe u elementen van een *dictionary* bewerkt;
- wat operatoren zijn;
- hoe u verschillende soorten operatoren kunt gebruiken in uw code.



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.

6.2 Lists

In de vorige les heeft u gezien hoe u een enkele waarde kunt toekennen aan een variabele. Bijvoorbeeld:

```
mijn_var = "Hallo wereld"
```

Of:

```
i = 10
```

Het is echter ook mogelijk om een reeks van waardes toe te kennen aan een enkele variabele. In de inleiding gaven we aan dat we in deze les twee methodes laten zien, waarvan de *list* er één is.



Wat in Python een *list* wordt genoemd, heet in andere programmeertalen vaak een **array**.

Een *list* ziet er bijvoorbeeld zo uit:

```
my_cart = ["appel", "banaan", "citroen"]
```

Over deze *list* kunnen we het volgende zeggen:

- De variabele is `my_cart`.
- Deze variabele heeft drie waardes. (Maar: het had ook een lege *list* kunnen zijn: `my_cart = []`.)

- In dit geval zijn al deze waardes *strings*. (Maar: het hadden ook *integers*, *floats*, *booleans* of zelfs een combinatie van deze typen kunnen zijn.)

Een *list* is, in tegenstelling tot een *string*, dynamisch. Daarmee bedoelen we dat er elementen aan toegevoegd kunnen worden en uit verwijderd kunnen worden.

In de volgende subparagrafen gaan we op verschillende manieren aan de slag met *lists*.

6.2.1 Een enkel element gebruiken

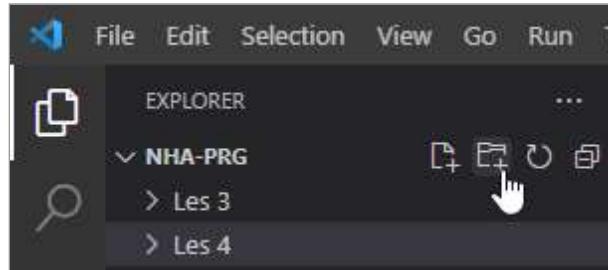
We beginnen met het aanmaken van een *list* en een enkel element uit die *list* halen en printen.

- Open VS Code.
- Open een map die u eerder aanmaakte. (Bijvoorbeeld NHA-prg).



Weet u het nog? Een map openen doet u door in de *Menu Bar* op "File" te klikken, vervolgens te klikken op "Open Folder" en tot slot de map te selecteren.

- Klik op de knop "New Folder".



Klik op de knop "New Folder".

- Geef de map een naam (bijvoorbeeld Les 6).
- Klik in de *Menu Bar* op "File".
- Klik op "Open Folder".
- Selecteer de map die u zojuist als laatste aanmaakte (Les 6).
- 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:

```
mijn_lijs = ["appel", "banaan", "citroen"]
```

Deze *list* voldoet aan de standaardopbouw van een *list*:

- De elementen staan tussen rechte haken: [].
- De verschillende elementen worden gescheiden door komma's.

Deze *list* bevat drie elementen: appel, banaan en citroen. Ieder element heeft zijn eigen **index**. Dat is een nummer om aan te geven waar in de *list* het element zich bevindt.



De nummering van de elementen begint niet bij 1, maar bij 0!

Dus:

Element	Index
appel	0
banaan	1
citroen	2

- Voeg nu de volgende code toe aan het bestand:

```
keuze = mijn_lijst[1]
print(keuze)
```

Wat denkt u dat er in beeld verschijnt als u dit programma uitvoert?

- Klik in de *Menu Bar* op "File".
- Klik op "Save" om het bestand op te slaan.
- Klik in de *Menu Bar* op "Terminal".
- Klik op "New Terminal".
- Klik ergens in de *Terminal* (om zeker te weten dat deze actief is).
- Bent u een Windows-gebruiker? Typ dan `python test.py` (of de door u gekozen bestandsnaam). Als Mac-gebruiker typt u `python3 test.py`.
- Druk op Enter.

Als alles goed is gegaan, ziet u het volgende in het venster verschijnen:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
banaan

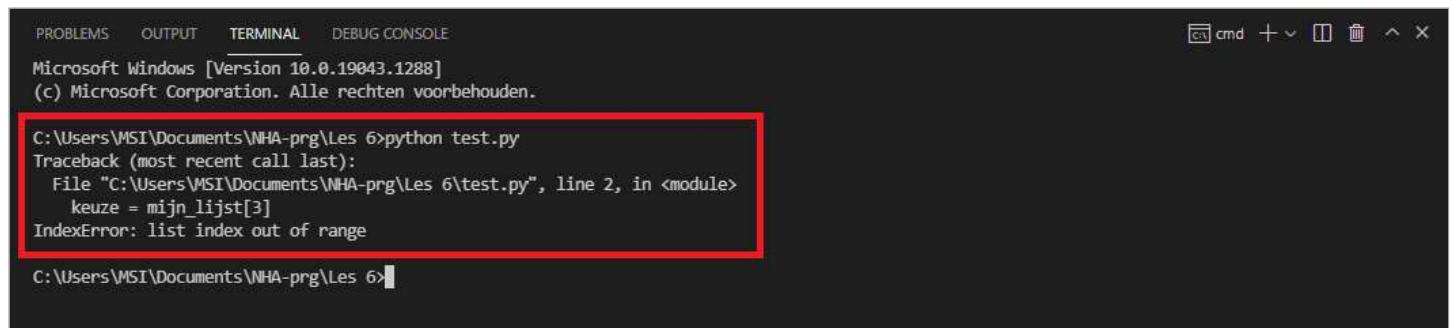
C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Was het resultaat wat u verwachtte?

We hebben de variabele `keuze` gedefinieerd. De waarde van deze variabele is het element uit de *list* `mijn_lijst` waarvan de index 1 is. Hadden we gewild dat `appel` in beeld zou verschijnen, dan hadden we de volgende code moeten invoeren:

```
mijn_lijst = ["appel", "banaan", "citroen"]
keuze = mijn_lijst[0]
print(keuze)
```

Hadden we de derde optie in de *list* willen printen en per ongeluk `keuze = mijn_lijst[3]` ingevoerd, dan hadden we bij de uitvoering van het programma een foutmelding gezien:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Traceback (most recent call last):
  File "C:\Users\MSI\Documents\NHA-prg\Les 6\test.py", line 2, in <module>
    keuze = mijn_lijst[3]
IndexError: list index out of range

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Logisch, want er is geen element met index 3 gedefinieerd.

- Pas de code in het bestand als volgt aan:

```
mijn_lijst = ["appel", "banaan", "citroen"]
keuze = mijn_lijst[-1]
```

```
print(keuze)
```

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

Misschien had u ook nu een foutmelding verwacht. Die krijgt u echter niet te zien:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
citroen

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Met behulp van het minteken kunt u terugstellen. Als u `-1` invoert, krijgt u het laatste item van de *list* te zien. Had u gekozen voor `-2` en `-3`, dan had u banaan en appel in beeld gekregen, bij `-4` was dat een foutmelding geweest.

6.2.2 Een reeks elementen gebruiken

Als een *list* meerdere elementen kan bevatten, is het ook handig als u meer dan een van die elementen kunt gebruiken. Hoe dat werkt, tonen we u in deze subparagraaf.

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

```
mijn_lijst = ["Ma", "Di", "Wo", "Do", "Vr", "Za", "Zo"]
```

- Voeg de volgende code daaronder toe:

```
print(mijn_lijst[2:5])
```

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

Het resultaat ziet er dan als volgt uit:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
['Wo', 'Do', 'Vr']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

U ziet de elementen met de index 2, 3 en 4. U had misschien ook het element met index 5 verwacht, maar dan had u `[2:6]` moeten invoeren. De dubbele punt betekent namelijk 'tot' en niet 'tot en met'.

- Pas de code in het bestand als volgt aan:

```
mijn_lijst = ["Ma", "Di", "Wo", "Do", "Vr", "Za", "Zo"]
print(mijn_lijst[5:])
```

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

U krijgt dan het volgende resultaat:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
['Za', 'Zo']
```

C:\Users\MSI\Documents\NHA-prg\Les 6>

Door na de dubbele punt geen getal in te voeren, hebben we aangegeven dat we alle elementen die na het element met index 5 komen, willen printen. Dat kan ook andersom: `print(mijn_lijst[:4])` oftewel: print alle elementen die voor het element met index 4 in de *list* komen:

```
['Ma', 'Di', 'Wo', 'Do']
```

Samengevat voor `mijn_lijst = ["Ma", "Di", "Wo", "Do", "Vr", "Za", "Zo"]`:

<code>mijn_lijst[3]</code>	<code>['Do']</code>
<code>mijn_lijst[-1]</code>	<code>['Zo']</code>
<code>mijn_lijst[5:]</code>	<code>['Za', 'Zo']</code>
<code>mijn_lijst[:4]</code>	<code>['Ma', 'Di', 'Wo', 'Do']</code>

6.2.3 Iteratie

Lists kunt u combineren met iets wat u eerder geleerd heeft, namelijk iteratie, oftewel lussen (*loops*). U kunt door een *list* *lopen* en zo een voor een alle elementen van de *list* 'pakken' en er iets mee doen.

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

```
mijn_lijst = ["Wafels", "Softijs", "Schepijs", "Pannenkoeken"]
```

- Voeg de volgende code daaronder toe:

```
for item in mijn_lijst:
    print(f"Wij verkopen", item)
```

Herhaling: De `f` voor onze *string* die geprint wordt, zorgt ervoor dat de Python-*interpreter* begrijpt dat alle elementen die tussen de accolades (`{}`) staan, variabelen zijn.

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

Het resultaat ziet er dan als volgt uit:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Wij verkopen Wafels
Wij verkopen Softijs
Wij verkopen Schepijs
Wij verkopen Pannenkoeken
```

C:\Users\MSI\Documents\NHA-prg\Les 6>

Door het combineren van *lists* en iteratie kunt u de code logisch en leesbaar houden, iets waar Python erg goed in is.

6.2.4 Een element toevoegen

Eerder in deze paragraaf gaven we al aan dat *lists* dynamisch zijn. Dat betekent dat het mogelijk is om er later nog elementen aan toe te voegen of ze juist te verwijderen.

In deze subparagraph richten we ons op het toevoegen van elementen. Daarvoor gebruikt u de methode `append`.

Een **methode** is een soort functie die u vastmaakt aan een object (in dit geval een *list*). Dat klinkt ingewikkeld, maar ziet er in de codeerpraktijk eigenlijk vrij simpel uit. Bijvoorbeeld:

```
mijn_lijst.append("Muffins")
```

Achtereenvolgens ziet u:

1. De naam van de <i>list</i>	<code>mijn_lijst</code>
2. Een punt	<code>.</code>
3. De methode	<code>append</code>
4. Haakje openen	<code>(</code>
5. Het element dat u wilt toevoegen	<code>"Muffins"</code>
6. Haakje sluiten	<code>)</code>

- Verwijder alle code in het bestand, behalve de bovenste regel.
- Voeg de volgende code toe:

```
mijn_lijst.append("Muffins")
for item in mijn_lijst:
    print(f"Wij verkopen", item)
```

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

En zie hier het resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\WHA-prg\Les 6>python test.py
Wij verkopen Wafels
Wij verkopen Softijs
Wij verkopen Schepijs
Wij verkopen Pannenkoeken
Wij verkopen Muffins

C:\Users\MSI\Documents\WHA-prg\Les 6>
```

U ziet dat u met `append()` een element kunt toevoegen aan het einde van een *list*.

Wilt u een element op een andere plaats in de *list* invoegen? Dan gebruikt u niet de methode `append`, maar de methode `insert`.

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

```
alfabet = ["A", "B", "D", "E", "F"]
```

- Voeg de volgende code daaronder toe:

```
print(alfabet)
```

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

Dit is het resultaat:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
['A', 'B', 'D', 'E', 'F']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Zoals u ziet, zijn we de C vergeten in deze *list*. Die willen we toevoegen tussen de B en D.

Maar eerst kijken we naar de index van onze elementen:

Element	Index
A	0
B	1
C	2
E	3
F	4

Het nieuwe element moet dus geplaatst worden tussen het element met index 1 en het element met index 2. Daarvoor gebruiken we de volgende code:

```
alfabet.insert(2, "C")
```

Het eerste deel van de code kent u al: de naam van de *list*, gevolgd door een punt en de naam van de methode en een haakje openen. Daarna volgt 2, "C". Hiermee zegt u tegen de computer: "Voeg waarde "C" in vóór het element met index 2."

- Pas de code in het bestand als volgt aan:

```
alfabet = ["A", "B", "D", "E", "F"]
alfabet.insert(2, "C")
print(alfabet)
```

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

Het resultaat:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
['A', 'B', 'C', 'D', 'E', 'F']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

U ziet het: "C" is precies op de juiste plaats ingevoegd!

6.2.5 Een element verwijderen

We gaven het al aan: u kunt niet alleen elementen toevoegen, maar ook elementen verwijderen.

Ook nu behandelen we twee elementen waarmee u dit zou kunnen doen: *pop* en *remove*. We beginnen met *pop*.

Met *pop* verwijdert u een element uit een *list*. Om welke element het gaat, definieert u door de index van dat element tussen de haakjes te typen.

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

```
fruit = ["appel", "banaan", "kers"]
```

- Voeg de volgende code daaronder toe:

```
print("Voor pop()", fruit)
fruit.pop(1)
print("Na pop()", fruit)
```

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

Dit zou het resultaat moeten zijn:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Voor pop() ['appel', 'banaan', 'kers']
Na pop() ['appel', 'kers']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Zowel het resultaat voor als na het 'poppen' wordt getoond. We hebben het element met index 1 verwijderd.

Met `remove()` kunt u hetzelfde doen. U plaatst dan echter niet de index van het te verwijderen element tussen de haakjes, maar het element zelf:

- Verwijder alle code in het bestand.
- Voeg de volgende code toe:

```
fruit = ["druif", "sinaasappel", "perzik", "kiwi", "druif"]
print("Voor remove()", fruit)
fruit.remove("perzik")
print("Na remove()", fruit)
```

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

En zie hier het resultaat:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Voor remove() ['druif', 'sinaasappel', 'perzik', 'kiwi', 'druif']
Na remove() ['druif', 'sinaasappel', 'kiwi', 'druif']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Heeft u gezien dat het element `druif` tweemaal voorkomt in de *list*? Wat zou er gebeuren als we `remove("druif")` toepassen?

- Verwijder alle code in het bestand, behalve de bovenste regel.
- Voeg de volgende code toe:

```
print("Voor remove()", fruit)
fruit.remove("druif")
print("Na remove()", fruit)
```

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

U ziet dat alleen het eerste element, dat gelijk is aan de waarde tussen de haakjes, wordt verwijderd:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Voor remove() ['druif', 'sinaasappel', 'perzik', 'kiwi', 'druif']
Na remove() ['sinaasappel', 'perzik', 'kiwi', 'druif']

C:\Users\MSI\Documents\NHA-prg\Les 6>
```



U kunt bij `.remove()` slechts één parameter angeven. Wilt u beide exemplaren van het woord "druif" gebruiken, dan zult u dus twee keer `.remove()` moeten uitvoeren of een `loop` moeten gebruiken.

6.2.6 Overzicht met beschikbare methodes

Ten slotte geven we u een compleet overzicht van alle methodes die u kunt toepassen op een *list*:

Methode	Beschrijving
<code>append()</code>	Voegt een element toe aan het eind van de <i>list</i> .
<code>clear()</code>	Verwijdert alle elementen uit de <i>list</i> .
<code>copy()</code>	Maakt een kopie van de <i>list</i> .
<code>count()</code>	Telt het aantal elementen in de <i>list</i> met een bepaalde waarde.
<code>extend()</code>	Voegt een <i>list</i> toe aan de bestaande <i>list</i> .
<code>index()</code>	Geeft de index van het eerste element dat gelijk is aan een bepaalde waarde.
<code>insert()</code>	Voegt een element toe vóór de plaats met de gespecificeerde index.
<code>pop()</code>	Verwijdert het element met de gespecificeerde index.
<code>remove()</code>	Verwijdert het eerste element met de gespecificeerde waarde.
<code>reverse()</code>	Draait de volgorde van de <i>list</i> om.
<code>sort()</code>	Sorteert de <i>list</i> alfabetisch.

6.3 Dictionaries

Naast *lists* bestaan er ook *dictionaries*. Dit is net als een *list* een verzameling elementen. In plaats van een index hebben de elementen van een *dictionary* een **sleutel** (key).

U weet al hoe het zit met *lists*. Een voorbeeld ter herhaling:

```
mijn_lijst = ["Jan", "Feb", "Mar", "Apr", "Mei"]
```

Element	Index
Jan	0
Feb	1
Mar	2
Apr	3
Mei	4

Bij een *dictionary* ziet het er bijvoorbeeld zo uit:

```

mijn_dictionary = {
    "merk" : "Mitsubishi",
    "model" : "Colt",
    "bouwjaar" : 2010,
    "kleur" : "blauw",
    "staat" : "gebruikt"
}

```

De sleutels en hun bijbehorende waarde zijn dan:

Sleutel	Waarde
merk	Mitsubishi
model	Colt
bouwjaar	2010
kleur	blauw
staat	gebruikt

De sleutels in een *dictionary* moeten uniek zijn. U kunt in één *dictionary* dus niet twee keer dezelfde sleutel gebruiken.



Zo'n paar (sleutel + waarde) wordt in vaktermen ook wel een **key-value pair** genoemd.

U ziet dat in een *dictionary* de elementen tussen accolades staan: {}. De elementen worden gescheiden door komma's, een dubbele punt wordt gebruikt om de sleutel van de bijbehorende waarde te scheiden.

6.3.1 Een enkel element gebruiken

Eerder lieten we u zien hoe u in het geval van een *list* een enkel element kon printen. Een voorbeeld ter herhaling:

```

mijn_lijst = ["boter", "kaas", "eieren"]
print(mijn_lijst[1])

```

Bij *dictionaries* gaat dat vrijwel net zo. Het enige verschil is dat we nu niet kunnen werken met de index, maar dat we de sleutel moeten gebruiken.

- Verwijder alle code in het bestand.
- Voeg de volgende code toe:

```

mijn_dictionary = {
    "product" : "softijs",
    "aantal" : 101,
    "smaak" : "vanille"
}

```



Als u een *dictionary* gaat invoeren, zult u merken dat de sleutels en hun waarden worden ingesprongen. Daardoor blijft het overzichtelijk als we iedere sleutel op een aparte regel zetten.

- Voeg daaronder de volgende code toe:

```
print(mijn_dictionary["aantal"])
```

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

Dit geeft het volgende resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
101

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

6.3.2 Meerdere elementen gebruiken

Het kan voorkomen dat u alle sleutels van een *dictionary* aan één variabele wilt toekennen. Dat doet u bijvoorbeeld op de volgende manier:

```
keys = mijn_dictionary.keys()
```

Achtereenvolgens ziet u:

1. Aangeven wat u wilt definiëren	keys
2. Gelijkteken	=
3. Naam van de dictionary	mijn_dictionary
4. Punt	.
5. Verwijzing naar de keys in de dictionary	keys
6. Haakje openen	(
7. Haakje sluiten)

- Verwijder de laatste regel in het bestand.
- Voeg de volgende code toe:

```
keys = mijn_dictionary.keys()
print(keys)
```

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

Dit geeft het volgende resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
dict_keys(['product', 'aantal', 'smaak'])

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Hetzelfde kunnen we doen met de waarden van de *dictionary*. In dat geval gebruikt u:

```
values = mijn_dictionary.values()
```

We voegen een extra *print*-statement toe, zodat u kunt zien wat nu precies wat is:

- Pas de code in het bestand op de volgende manier aan:

```
mijn_dictionary = {
    "product" : "softijs",
    "aantal" : 101,
    "smaak" : "vanille"
```

```
}

keys = mijn_dictionary.keys()
values = mijn_dictionary.values()
print("keys:", keys)
print("values:", values)
```

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

Dit geeft het volgende resultaat:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
keys: dict_keys(['product', 'aantal', 'smaak'])
values: dict_values(['softijs', 101, 'vanille'])

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

U kunt natuurlijk ook de *dictionary* in zijn geheel printen. Daarvoor gebruikt u `print(mijn_dictionary)`. Het resultaat is dan:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
{'product': 'softijs', 'aantal': 101, 'smaak': 'vanille'}

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

6.3.3 Iteratie

Ook *dictionaries* kunt u combineren met iteratie. U kunt op de volgende manier door ons voorbeeld-*dictionary* loopen:

```
for item in mijn_dictionary:
    print(item)
```

U krijgt dan de sleutels van de *dictionary* te zien:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
product
aantal
smaak

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Wilt u in plaats van door de sleutels door de waarden itereren, dan heeft u de volgende code nodig:

```
for item in mijn_dictionary:
    print(mijn_dictionary[item])
```

De uitvoer zal er dan zo uitzien:

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
softijs
101
vanille
C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Dan is er nog de methode `.items()`. Deze methode kunt u bijvoorbeeld op de volgende manier gebruiken:

```
for k, v in mijn_dictionary.items():
    print(k, v)
```



De k staat voor *key*, de v staat voor *value*.

De iteratie zal alle *key-value pairs* uitprinten:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
product softijs
aantal 101
smaak vanille
C:\Users\MSI\Documents\NHA-prg\Les 6>
```

6.3.4 Een element toevoegen

Ook een *dictionary* is dynamisch, waardoor u achteraf nog elementen kunt toevoegen. Hoe dat werkt, laten we zien aan de hand van een voorbeeld.

- Verwijder alle code in het bestand.
- Voeg de volgende code toe:

```
mijn_dictionary = {
    "Voornaam" : "Harry",
    "Geboortedatum" : "31-maart-1939",
    "Registratienummer" : "AA18891"
}
```

- Voeg daaronder de volgende code toe:

```
print()
for k, v in mijn_dictionary.items():
    print (k, v)
```

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

Dit geeft het volgende resultaat:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
```

Voornaam Harry
Geboortedatum 31-maart-1939
Registratienummer AA18891

```
C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Stel, we willen het element Achternaam toevoegen. Dat doen we door voor het `print`-statement deze regel code toe te voegen:

```
mijn_dictionary = {  
    "Voornaam" : "Harry",  
    "Geboortedatum" : "31-maart-1939",  
    "Registratienummer" : "AA18891"  
}  
mijn_dictionary["Achternaam"] = "de Vries"  
print()  
for k, v in mijn_dictionary.items():  
    print (k, v)
```

De uitvoer is dan:

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

```
C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
```

Voornaam Harry
Geboortedatum 31-maart-1939
Registratienummer AA18891
Achternaam de Vries

```
C:\Users\MSI\Documents\NHA-prg\Les 6>
```

U definieert dus een nieuw element door de naam van de `dictionary` te typen, gevolgd door rechte haken met daartussen de sleutel. U eindigt met een gelijkteken en de waarde. Het nieuwe element voegt u achteraan in de `dictionary` toe.



Het is gebruikelijk om als sleutel een *string* te gebruiken ("Achternaam" in dit geval), maar het is ook mogelijk om een *integer* of zelfs een *float* te gebruiken.

Lengte van de `dictionary`

De `dictionaries` die we tot nu toe als voorbeeld gebruikten, waren door hun beperkte hoeveelheid elementen erg overzichtelijk. In de praktijk is dat vaak anders. Wilt u achterhalen wat de lengte van een `dictionary` is (lees: uit hoeveel elementen de `dictionary` bestaat), dan gebruikt u de methode `len`.

Bijvoorbeeld:

```
for k, v in mijn_dictionary.items():  
    print (k, v)  
  
print()  
print(len(mijn_dictionary))
```

Uitvoer:

```
Voornaam Harry  
Geboortedatum 31-maart-1939  
Registratienummer AA18891
```

6.3.5 Een element verwijderen

Er zijn verschillende manieren om een element uit een *dictionary* te verwijderen. We behandelen er enkele, te beginnen met `pop`.

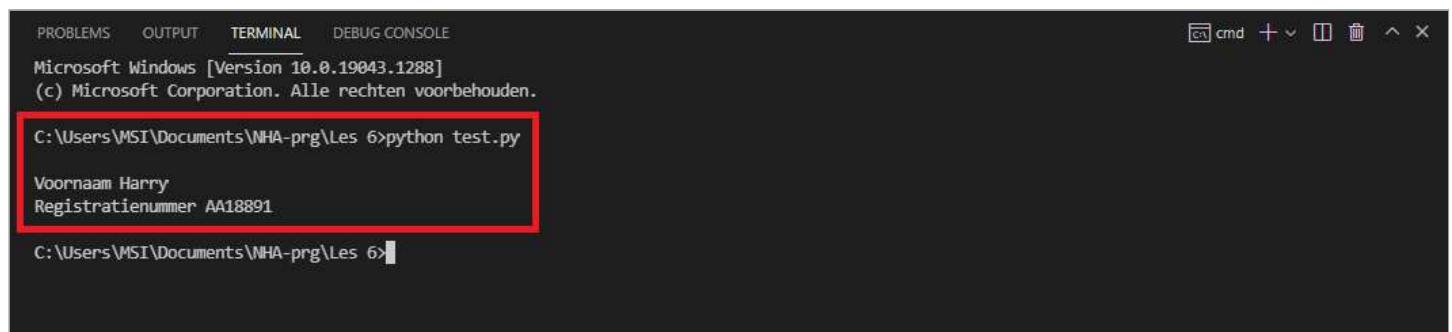
We gaan het geboortejaar verwijderen. Dat doet u als volgt:

- Verwijder alle code uit het bestand, behalve de *dictionary* zelf.
- Voeg daaronder de volgende code toe:

```
mijn_dictionary.pop("Geboortedatum")
print()
for k, v in mijn_dictionary.items():
    print (k, v)
```

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

Dit geeft het volgende resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Voornaam Harry
Registratienummer AA18891

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Als u de methode `pop` wilt gebruiken, typt u dus tussen ronde haakjes achter de sleutel van het element dat u wilt verwijderen.

Hetzelfde resultaat kunt u bereiken met de methode `del`:

```
del mijn_dictionary["Geboortejaar"]
```

Met de methode `popitem` verwijdert u het laatst toegevoegde element:

- Verwijder alle code uit het bestand, behalve de *dictionary* zelf.
- Voeg daaronder de volgende code toe:

```
mijn_dictionary.popitem()
print()
for k, v in mijn_dictionary.items():
    print (k, v)
```

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

De uitvoer ziet er dan als volgt uit:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
Voornaam Harry
Geboortedatum 31-maart-1939

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Wilt u de *dictionary* helemaal leegmaken? Dan gebruikt u de volgende methode:

```
mijn_dictionary.clear()
```

Als u de *dictionary* helemaal wilt verwijderen, typt u:

```
del mijn_dictionary
```

6.3.6 Elementen bewerken

U kunt de elementen van een *dictionary* op verschillende manieren bewerken. Neem bijvoorbeeld het volgende *dictionary*:

```
mijn_dictionary = {
    "product" : "softijs",
    "aantal" : 101,
    "smaak" : "vanille"
}
```

U verandert het getal 101 op de volgende manier in 150:

```
mijn_dictionary["aantal"] = 150
```

Voor het veranderen van een element in een *dictionary* kunt u ook de methode *update* gebruiken:

- Verwijder alle code uit het bestand, behalve de *dictionary* zelf.
- Voeg daaronder de volgende code toe:

```
mijn_dictionary.update({"aantal" : 250})
print(mijn_dictionary["aantal"])
```

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

De uitvoer ziet er dan als volgt uit:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
250

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

6.4 Operatoren

U heeft gezien dat variabelen erg handig zijn bij het programmeren. Ook heeft u al wat sleutelwoorden en -tekens gezien die variabelen manipuleren (zoals de + in `c = a + b`). Dit soort sleutelwoorden/-tekens noemen we **operatoren**.

Er zijn verschillende soorten operatorcategorieën. De belangrijkste daarvan behandelen we in de volgende subparagrafen.

6.4.1 Rekenkundige operatoren

We hebben er al een beetje mee geoefend, maar gaan nu eens kijken welke rekenkundige operatoren we kunnen toepassen in Python.

- Verwijder alle code uit het bestand.
- Voeg de volgende code toe:

```
print(12 + 4)
print(12 - 4)
print(12 * 4)
print(12 / 4)
```

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

Dit geeft het volgende resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
16
8
48
3.0
```

We hadden we waardes ook eerst in een variabele kunnen stoppen:

```
b = 12
c = 4

print (b + c)
print (b - c)
print (b * c)
print (b / c)
```

Dat had hetzelfde resultaat gegeven.



Was het u opgevallen? De eerste drie uitkomsten zijn *integers*, de laatste is een *float*! U kunt met behulp van type zelf controleren wat voor type een waarde heeft. Bijvoorbeeld:

```
b = 12
c = 4
i = b/c
print("uitkomst:", i)
print("type:", type(i))
```

Het resultaat is dan:

```
uitkomst 3.0
type: <class 'float'>
```

Naast het standaard optellen, aftrekken, vermenigvuldigen en delen, kunnen we ook machtsverheffen en de modulus berekenen:

- Verwijder alle code uit het bestand.
- Voeg de volgende code toe:

```
b = 12
c = 4
i = b**c
```

```
j = b%c
print("machtsverheffen:", i)
print("modulus:", j)
```

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

Dit geeft het volgende resultaat:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
machtsverheffen: 20736
modulus: 0

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

Naast deze rekenkundige operatoren zijn er nog tal van wiskundige functies die standaard meegeleverd worden met Python. Anders gezegd: functies die in de *standard library* van Python zitten. Bijvoorbeeld:

Functie	Resultaat
abs()	Toont de absolute waarde
pow()	Machtsverheffen (twee waardes vereist)

Als u de module `math` importeert, kunt u zelfs nog meer functies gebruiken. Dit importeren doet u door de code te beginnen met:

```
import math
```

Op www.w3schools.com/python/module_math.asp vindt u een overzicht van de functies die daarvoor bruikbaar worden. Bijvoorbeeld:

```
b = 64
c = 1.4
print("worteltrekken:", math.sqrt(b))
print("afronden naar boven:", math.ceil(c))
```

Uitvoer:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
worteltrekken: 8.0
afronden naar boven: 2

C:\Users\MSI\Documents\NHA-prg\Les 6>
```

6.4.2 Relationale operatoren

Relationale operatoren zijn operatoren waarmee u zaken kunt vergelijken:

Is a kleiner dan b?	<code>a < b</code>
Is a groter dan b?	<code>a > b</code>
Is a gelijk aan b?	<code>a == b</code>



We gebruiken voor 'gelijk aan' een dubbel gelijkteken (==), omdat het enkele gelijkteken (=) al gereserveerd is voor het toekennen van een waarde, bijvoorbeeld aan een variabele (bijvoorbeeld a = 10).

Een voorbeeld:

- Verwijder alle code uit het bestand.
- Voeg de volgende code toe:

```
a = 10
b = 4
print (a > b)
print (b > a)
print(b == a)
```

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

Dit geeft het volgende resultaat:

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
True
False
False
```

The output of the Python code is highlighted with a red rectangle.



Relationele operatoren worden vaak gebruikt in combinatie met een if-statement. Bijvoorbeeld:

```
if a>b
print('a is groter dan b')
```

Andere relationele operatoren zijn:

Operator	Betekenis	Voorbeeld
!=	is niet gelijk aan	a != b
>=	is groter of gelijk aan	a >= b
<=	is kleiner of gelijk aan	a <= b

6.4.3 Logische operatoren

Ten slotte zijn er nog drie logische operatoren: and, or en not. Hiermee kunt u condities definiëren, oftewel voorwaarden waaraan moet worden voldaan. We geven daarvan een eerste voorbeeld:

- Verwijder alle code uit het bestand.
- Voeg de volgende code toe:

```
leeftijd = 19
rijbewijs = "B"
if leeftijd>17 and rijbewijs != "":
    print("U mag autorijden")
else:
    print("U mag geen autorijden")
```

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

Dit geeft het volgende resultaat:

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 6>python test.py
U mag autorijken

C:\Users\MSI\Documents\NHA-prg\Les 6>

```

Het *if-statement* in dit voorbeeld heeft twee voorwaarden:

1. *leeftijd* is groter dan 17;
2. *rijbewijs* is niet leeg.

Er is gebruikgemaakt van de operator *and* (Nederlands: *en*). Als aan voorwaarde 1 EN voorwaarde 2 voldaan wordt, zal *U mag autorijken* geprint worden. Als niet voldaan wordt aan een of geen van beide voorwaarden, zal *U mag geen autorijken* geprint worden. In ons voorbeeld is het eerste het geval.



Als de *leeftijd* groter is dan 17 en u bij *rijbewijs* bijvoorbeeld nee invult, zult u toch de melding *U mag autorijken* krijgen. Dat is in de werkelijkheid natuurlijk niet de bedoeling, maar nemen we in dit 'beginnersvoorbeeld' voor lief.

Waar bij *and* voldaan moet worden aan beide voorwaarden, is het bij de logische operator *or* (Nederlands: *of*) voldoende als er aan voorwaarde 1 OF voorwaarde 2 wordt voldaan. Bijvoorbeeld:

```

het_regent = False
tijdstip = "nacht"
if het_regent or tijdstip == "nacht":
    print("Ik ga niet naar buiten")
else:
    print("Ik ga naar buiten")

```



Merk op dat we in de eerste voorwaarde volstaan met *het_regent*. We schrijven niet *het_regent == True*.

De code zou wel werken, maar het is niet nodig. Als u als voorwaarde een variabele aanlevert, zal Python eenvoudigweg checken of die voorwaarde *True* (waar) is.

6.4.4 Toewijzingsoperatoren

De laatste categorie die we bespreken, is die van de toewijzingsoperatoren. Deze operatoren worden gebruikt om waarden aan variabelen toe te wijzen.

De bekendste toewijzingsoperator is *=*. Deze operator heeft u al verschillende keren gezien. In deze les bijvoorbeeld:

```

mijn_var = "Hallo wereld"
i = 10

```

Daarnaast zijn er nog een aantal operatoren waarmee u een waarde aan een variabele kunt toewijzen. De operator wordt dan vooral ingezet om de toewijzing op een kortere (en dus snellere) manier te kunnen schrijven:

Operator	Voorbeeld	
	Korte schrijfwijze	Volledige schrijfwijze
=	x = 3	x = 3
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Vooral de korte wijze met += zult u in de komende lessen nog regelmatig voorbij zien komen.

6.5 Samenvatting

In deze les heeft u leren werken met *lists* en *dictionaries*. U werkt ermee als u meerdere waarden wilt opslaan in één variabele:

- Een **list** is een geordende datastructuur. Dit houdt in dat u (met behulp van de index) precies op een plek kunt vinden wat op een bepaalde plaats staat.
- Niet alle dataverzamelingen zijn geordend. Werkt u met een ongeordende datastructuur, dan komen **dictionaries** van pas. Om hierin een bepaald element te vinden, moet u de sleutel (*key*) van het element kennen.

In deze les heeft u geleerd hoe u een *list* en een *dictionary* samenstelt:

List	Dictionary
naam = []	naam = { sleutel : waarde }
Komma's scheiden elementen.	Komma's scheiden elementen. Dubbele punt scheidt sleutel en waarde.

Waar ieder element in een *list* een bepaalde index heeft (waarvan de nummering begint bij 0!), heeft ieder element in een *dictionary* een sleutel (met een bepaalde waarde).

U leerde hoe u een of meerdere elementen van een *list* of *dictionary* kunt printen, maar ook hoe u uw kennis over *lists* en *dictionaries* kunt combineren met iteratie (zie vorige les).

Daarnaast leerde u diverse bewerkingen uit te voeren met behulp van methodes:

Bewerking	List	Dictionary
Element toevoegen	append	naam[] = sleutel
	insert	
Element verwijderen	pop	pop
	remove	popitem
Elementen bewerken	-	naam[sleutel] = nieuwe waarde
	-	update

Tot heeft u kennismegemaakt met vier soorten operatoren:

- Met **rekenkundige operatoren** kunt u rekenkundige bewerkingen uitvoeren, zoals optellen of machtsverheffen.
- Met **relationele operatoren** kunt u zaken vergelijken.
- Met **logische operatoren** kunt u voorwaarden definiëren.
- Met **toewijzingsoperatoren** kunt u een waarde aan een variabele toekennen.

Kortom: u heeft weer een hoop basiskennis opgedaan. Genoeg om in de volgende les eindelijk starten met het maken van de webshop van de ijssalon.

6.6 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.



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.

6.7 Oefenopgaven

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

In welke situatie gebruikt u een *list* of *dictionary*?

→ Bekijk antwoord

Waaraan herkent u een *list*?

→ Bekijk antwoord

Bekijk de volgende *list*:

```
smaken = ["aardbei", "mango", "banaan"]
```

U voert het commando uit op deze *list*:

```
print(smaken[1])
```

Wat is het resultaat?

→ Bekijk antwoord

Bekijk de volgende *list*:

```
smaken = ["aardbei", "mango", "banaan"]
```

U voert het commando uit op deze *list*:

```
print(smaken[-1])
```

Wat is het resultaat?

→ Bekijk antwoord

Bekijk de volgende *list*:

```
smaken = ["aardbei", "mango", "banaan"]
```

Welk commando moet u op deze *list* uitvoeren om de eerste twee elementen van deze *list* te printen?

→ Bekijk antwoord

Bekijk de volgende *list*:

```
smaken = ["aardbei", "mango", "banaan"]
```

Welk commando moet u op deze *list* uitvoeren om de laatste twee elementen van deze *list* te printen?

→ Bekijk antwoord

Waaraan herkent u een *dictionary*?

→ Bekijk antwoord

Bekijk het volgende *dictionary*:

```
mijn_dictionary = {  
    "naam" : "Dolf",  
    "leeftijd" : 26,  
    "auto" : "Fiat Punto"  
}
```

Welk commando moet u op dit *dictionary* uitvoeren om Dolfs leeftijd te printen?

→ Bekijk antwoord

Bekijk het volgende commando:

```
print(2**3)
```

Wat is het resultaat als u dit commando uitvoert?

→ Bekijk antwoord

Bekijk de volgende code:

```
A = 10  
B = 12  
if A > B:  
    print("A is groter dan B")
```

Wat is het resultaat als u dit programma uitvoert?

→ Bekijk antwoord

Bekijk de volgende code:

```
totaal += element
```

Wat is de volledige schrijfwijze van deze code?

→ Bekijk antwoord

6.8 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.

1. a. Maak een *list* die bestaat uit de volgende elementen:

Toyota	Mazda	Volkswagen	Jeep
--------	-------	------------	------

Voeg daaronder het commando toe waarmee deze *list* wordt geprint.

1. b. Voeg met behulp van een commando het element *Suzuki* toe aan de *list*. Voeg daarna opnieuw het commando toe waarmee deze *list* geprint kan worden.

2. a. Maak een *dictionary* dat bestaat uit de volgende elementen:

Voornaam	Harry
Achternaam	van Winkel
Geboortedatum	27-3-1939

Voeg daaronder het commando toe waarmee dit *dictionary* wordt geprint.

2. b. Print met behulp van een commando het element met de sleutel *voornaam* uit.

2. c. Verander met behulp van een commando de voornaam in *Henrikus*. Voeg daarna opnieuw het commando toe waarmee dit *dictionary* geprint kan worden.

3. Definieer twee variabelen: *a* (met waarde 10) en *b* (met waarde 2). Maak daarna een derde variabele (*c*), met als waarde *a* gedeeld door *b*. Gebruik hiervoor de rekenkundige operator voor delen.

4. Definieer twee variabelen: a (met waarde 2) en b (met waarde 3). Maak daarna een derde variabele (c), met als waarde a tot de macht b. Gebruik hiervoor de rekenkundige operator voor machtsverheffen.
5. Definieer twee variabelen: a (met waarde 4) en b (met waarde 5). Maak daarna een derde variabele (c), die True als waarde heeft als a en b gelijk zijn en False als a en b niet gelijk zijn. Gebruik hiervoor een relationele operator.