

Les 5 - Bouwstenen van een programma

« Vorige les

» Volgende les

Huiswerk

Programmeren voor Beginners - Les 5

Resultaat: 10,0

Bekijk huiswerk

Je hebt 1 vraagmoment. 1 moment over.

Ga naar stel een vraag

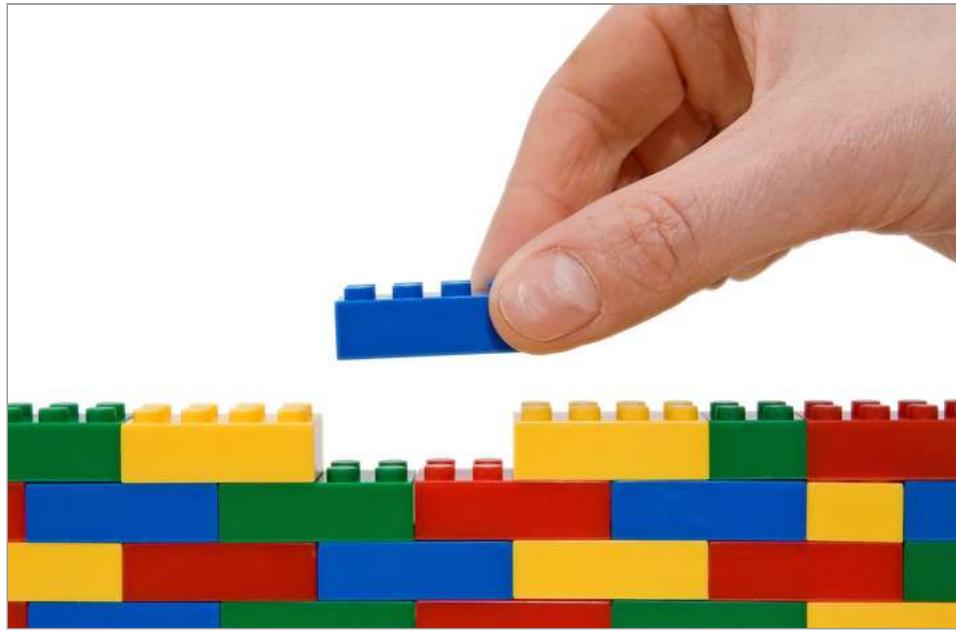
Inhoudsopgave

- 5.1 Inleiding
- 5.2 Sequentiestructuur
- 5.3 Variabelen
- 5.4 Datatypes
- 5.5 Namen van variabelen
- 5.6 Statements en bewerkingen
- 5.7 Print-statement
- 5.8 Loops
- 5.9 Commentaar toevoegen
- 5.10 Samenvatting
- 5.11 Vraagmoment
- 5.12 Oefenopgaven
- 5.13 Huiswerkopgaven

5.1 Inleiding

We hebben in de vorige lessen een flinke aanloop genomen, maar in deze les is het dan eindelijk tijd om ons te richten op de programmeertaal zelf.

In de basis is programmeren een creatief proces. U zou het kunnen vergelijken met het bouwen van een *Lego-kasteel*: dat gebeurt ook steentje voor steentje. Er zijn verschillende soorten steentjes, waarvan u er enkele vaak gebruikt. Bij het programmeren zijn die bouwstenen stukjes code. U zult het ene stukje vaker opnemen in uw programma dan het andere stukje.



Net als Lego werkt programmeren met bouwsteenjes!

In deze les maakt u kennis met de bouwsteenjes van programmeren, vooral met de steentjes die u vaak nodig zult hebben.

Leerdoelen

Aan het einde van deze les weet u:

- wat we bedoelen met een sequentiestructuur;
- wat variabelen zijn en waarom we ze gebruiken;
- dat een variabele van een bepaald datatype is;
- wat *strings* zoal kunnen bevatten;
- dat *booleans* slechts twee waardes kunnen hebben;
- aan welke eisen een variabelenaam moet voldoen;
- wat *statements* en bewerkingen zijn;
- hoe u uitvoer kunt creëren met behulp van een *print-statement*;
- wat *loops* zijn;
- wat het verschil is tussen de *for-* en *while-loop*;
- hoe u commentaar kunt toevoegen aan 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.

5.2 Sequentiestructuur

Veel computerprogramma's beginnen bij het begin en eindigen bij het eind. Dat klinkt eenvoudig en logisch. Toch is dat niet zo vanzelfsprekend als u wellicht zou denken.

De **sequentiestructuur** is een verzamelnaam voor alle structuren die bestaan in relatie tot programmeren. In les 2 hebben we twee van deze structuren besproken, toen we het verschil tussen lineaire programma's en object-georiënteerde programma's uitlegden. Ter herhaling:

Simpel gezegd lopen lineaire programma's stap voor stap een aantal programmaregels met gegevens door. Bij object-georiënteerde programma's zijn de volgorde en gegevens georganiseerd rond objecten.

De object-georiënteerde manier van werken noemen we ook wel *Object Oriented Programming (OOP)*. Hierbij is er dus geen sprake van een lineaire volgorde van het uitvoeren van opdrachten (lees: beginnen bij het begin en eindigen bij het einde).

Wat we precies bedoelen met een **lineaire volgorde**, leren we u aan de hand van een voorbeeldcode in een fictieve programmeertaal:

A = 1

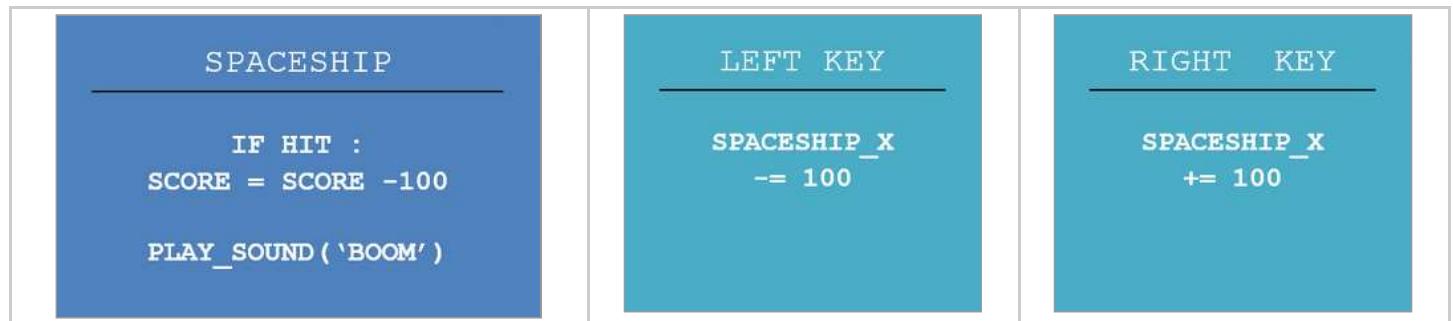
B = 5

```
C = A + B  
PRINT C
```

Als deze code wordt uitgevoerd, zal dat gebeuren in de volgorde waarin u de code zult lezen. Net zoals u deze les leest: u begint linksboven op de eerste regel en leest van links naar rechts. Aan het einde van de regel gaat u verder op de tweede regel (opnieuw van links naar rechts). De uitvoervolgorde van deze code is dus:

1. A = 1
2. B = 5
3. C = A + B
4. PRINT C

Bij veel programma's is het echter niet handig om de code op deze manier in te richten. Bij spellen zien we vaak bijvoorbeeld dit soort structuren:



Binnen elk van deze objecten zullen de commando's lineair worden uitgevoerd. Maar welk object wordt als eerste geactiveerd? Dat is maar de vraag. Dat hangt er helemaal vanaf wat de gebruiker van het programma (hier: het spel) doet.

Hoewel een programma dus niet lineair opgebouwd hoeft te zijn, zullen we ons in deze en de meeste volgende lessen wel beperken tot het programmeren in de lineaire volgorde.

5.3 Variabelen

De eerste bouwsteen die we noemen, zijn de variabelen. Wie programmeert, kan eigenlijk niet om het gebruik van variabelen uit. Maar wat zijn het?

We gaan even terug naar het eerste voorbeeld uit de vorige paragraaf. De eerste regel van deze code zag er als volgt uit:

```
A = 1
```

De A in dit voorbeeld is een variabele: het is een element met een waarde. In dit geval is de waarde 1, maar de waarde kan variëren. Het had dus ook 2 kunnen zijn, maar ook 10, 2918 of -3427. (Of zelfs iets heel anders, maar dat ziet u verderop in deze les.)

U zou een variabele kunnen vergelijken met een emmer. In ons voorbeeld heeft de emmer de naam A. De waarde van de variabele kunnen we zien als de vulling van de emmer, bijvoorbeeld 1 liter verf.

In ons voorbeeld hadden we ook nog de variabele B:

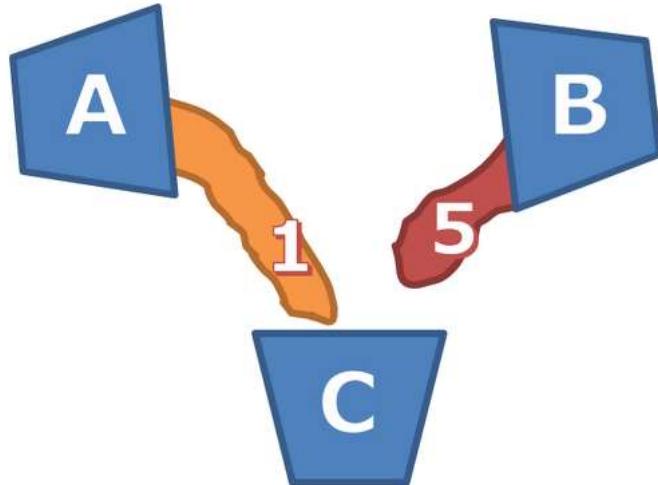
```
B = 5
```

Oftewel: een emmer met de naam B, die gevuld is met 5 liter verf.

Daarnaast was er nog de variabele C:

```
C = A + B
```

De uitvoer van deze regel zou u kunnen vergelijken met het overgieten van de inhoud van emmer A en emmer B in emmer C:



Na de uitvoer van deze actie zit er 6 liter verf in de emmer. De waarde van variabele C is dus 6.

U zult zich misschien afvragen waarom het werken met variabelen handig is. Een voorbeeld kan helpen bij het beantwoorden van deze vraag:

Stel, u krijgt de opdracht om een computerprogramma te ontwikkelen dat een willekeurig getal als invoer accepteert, dat getal verdubbelt en vervolgens het verdubbelde getal als uitvoer teruggeeft.

U zou dan voor elk getal een regel kunnen schrijven:

Als de invoer 1 is, geef dan 2 als uitvoer.

Als de invoer 2 is, geef dan 4 als uitvoer.

Als de invoer 3 is, geef dan 6 als uitvoer.

Als de invoer 4 is, geef dan 8 als uitvoer.

(enzovoorts)

Dat is niet alleen erg omslachtig, maar ook nog eens ondoenlijk. U kunt beter iets schrijven als:

Wat de invoer ook is, verdubbel die waarde en geef de uitkomst terug.

Die formulering is echter weer te menselijk, dat begrijpt een computer niet. (Al zouden recente ontwikkelingen daar wel eens verandering in kunnen brengen, kijkt u maar eens naar [deze video](#) op YouTube).

In de programmeertaal Python zou u deze formulering als volgt leesbaar kunnen maken voor een computer:

```
A = input("Welk getal wilt u verdubbelen?")
B = int(A)
C = B * 2
print(C)
```



Later in deze cursus zult u begrijpen waarom de regel `B = int(A)` nodig is.

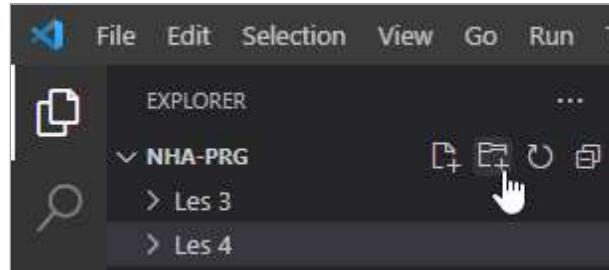
Probeer u het zelf maar eens uit:

- 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 5).
- Klik in de *Menu Bar* op "File".
- Klik op "Open Folder".
- Selecteer de map die u zojuist als laatste aanmaakte (Les 5).
- Klik in de *Activity Bar* op de knop "Explorer".
- 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:

```
A = input("Welk getal wilt u verdubbelen?")
B = int(A)
C = B * 2
print(C)
```

- 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 u ervan te verzekeren 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 de volgende vraag in het venster verschijnen:

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 5>python test.py
Welk getal wilt u verdubbelen?
```

- Voer het getal 10 in.
- Druk op Enter.

De output is inderdaad een verdubbeling van de input (van het getal 10):

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\MSI\Documents\NHA-prg\Les 5>python test.py
Welk getal wilt u verdubbelen?10
20
```

5.4 Datatypes

In de vorige paragraaf leerde u dat een variabele een getal als waarde kan hebben. Er zijn nog meer zogenaamde **datatypeën**, waarvan we er diverse behandelen in deze paragraaf.

5.4.1 Integers en floating-point numbers

Nog even terug naar de getallen. Als het gaat om getallen in Python, zijn er twee belangrijke smaken: *integers* en *floating-point numbers*. (Er is nog een derde smaak, maar die wordt vrijwel nooit gebruikt.)

- Een **integer** is een geheel getal, zoals -291, -5, -2, 0, 1, 12 of 2124. De waarden in ons voorgaande voorbeeld waren allemaal **integers**. Integer wordt vaak afgekort als **int**.
- Een **floating-point number** is een kommagetal, zoals 1.22145, 3.12 of -0.121. Als u meer precisie nodig heeft (bijvoorbeeld omdat u met financiële data werkt), gebruikt u een kommagetal. *Floating-point number* wordt vaak afgekort als **float**.



In Nederland en België gebruiken we een komma om aan te geven dat alles wat erachter komt, decimalen zijn.

In de VS echter gebruikt men hiervoor een punt, vandaar dat dit in programmeertalen ook wordt gedaan.

U vraagt zich misschien af waarom dit onderscheid nodig is. We kunnen toch altijd met kommagetallen werken en voor gehele getallen .0 gebruiken? Waarom we dat toch niet doen, is een tamelijk technisch verhaal. Het heeft te maken met hoe de computer met data omgaat. Mocht u het interessant vinden om te weten hoe dit zit, dan kunt u op [deze website](#) terecht voor meer informatie.

In dit voorbeeld ziet u hoe we in Python float's en int's gebruiken:

```
cornetto_prijs = 1.50
raketje_prijs = 0.75

aantal_cornettos = 2
aantal_raketjes = 3
```

Python herkent automatisch of het om een float of een int gaat. Dat is helaas niet bij elke taal het geval: soms moet u aangeven om wat voor soort getal het gaat. Bij de taal C doet u dat bijvoorbeeld als volgt:

```
float cornetto_prijs = 1.50;
float raketje_prijs = 0.75;

int aantal_cornettos = 2;
int aantal_raketjes = 3;
```

5.4.2 Strings

Een ander datatype is *string*. We zouden simpelweg kunnen zeggen dat dit een tekst is, maar dat dekt niet de hele lading. Een *string* kan namelijk niet alleen letters bevatten, maar ook cijfers, leestekens en symbolen. Daarnaast kan een *string* leeg zijn.

Enkele voorbeelden:

```
string1 = ""
string2 = "A"
string3 = "123ABC"
string4 = "Python is een taal die begin jaren 90 ontworpen werd."
string5 = " @&*()!_)_@*#(!_!"
```

De eerste regel code is een lege *string*, de andere vier regels laten zien hoe gevareerd de inhoud kan zijn wanneer de *string* wel gevuld is.

We kunnen vele bewerkingen uitvoeren op *strings*, dat zullen we u later laten zien. Voor nu geven we slechts een simpel voorbeeld:

```
str1 = "Vanille"
str2 = "ijs"
str3 = str1 + " " + str2
```

```
print(str3)
```

De uitvoer van dit programma is een samenvoeging van de drie variabelen `Vanille`, een spatie en `ijs`, oftewel `Vanille ijs`.

Zoals u ziet, kunt u een *string* creëren door dubbele aanhalingstekens om een aantal tekens te plaatsen. Een lege *string* typt u door alleen de twee dubbele aanhalingstekens na elkaar te plaatsen.



Als u in Python programmeert, maakt het niet uit of u enkele aanhalingstekens (') of dubbele quotes (") gebruikt. Beide tekens zullen werken.

5.4.3 Booleans

Een ander veelgebruikt datatype is de *boolean*. Dit datatype kan slechts twee waarden hebben: `True` of `False` (beide met een hoofdletter). Bijvoorbeeld:

```
mijn_variabele = True  
mijn_andere_variabele = False
```

U kunt zich voorstellen dat dit soort variabelen zeer weinig geheugenruimte innemen. Dat kan een groot voordeel zijn, zeker als de geheugenruimte beperkt is.

We zullen u later in deze les en de rest van de cursus laten zien wanneer en hoe u *booleans* gebruikt. We kunnen u nu alvast vertellen dat u ze vaker nodig heeft dan u wellicht denkt!

5.5 Namen van variabelen

Eerder zag u dat uw variabelen allerlei namen kunnen hebben, zoals `A`, `cornetto_prijs`, `str1` of `mijn_variabele`.

Toch bent u niet helemaal vrij in het kiezen van een variabele naam: er zijn regels waaraan u zich moet houden. Dit zijn de regels:

- Een variabelenaam mag alleen alfanumerieke karakters bevatten, oftewel letters van het alfabet (hoofd- of kleine letter) of de cijfers 0 tot en met 9. Daarnaast is het gebruik van underscores (_) toegestaan.
- Een variabelenaam moet beginnen met een letter of een *underscore*. Een variabele mag dus wel een getal bevatten, maar mag niet met een getal beginnen.
- Zogenoemde *reserved keywords* mogen niet gebruikt worden in de naam. Om welke namen het precies gaat in Python, kunt u achterhalen door het volgende programma te schrijven en uit te voeren in de *Terminal*:

```
import keyword  
print(keyword.kwlist)
```

Er verschijnen dan woorden als `True`, `and`, `with`, `not` en `else`.



U hoeft deze lijst niet te onthouden als u *VS Code* gebruikt. De editor zal u namelijk waarschuwen als u een van deze woorden toch gebruikt in uw naam.

U kunt uw variabele een korte naam geven, zoals we eerder in deze les deden met `A`, `B` en `C`. In ons voorbeeld was dat prima, maar we zullen u aan de hand van een voorbeeld laten zien dat dit niet aan te raden is:

Stel, u verkoopt setjes wenskaarten. Ieder setje kost € 2,95. U wil het aantal sets dat u heeft verkocht, kunnen invoeren en het totaalbedrag als uitvoer te zien krijgen. U schrijft daarom het volgende programma:

```
a = 5  
p = 2.95  
print(a * p)
```

Ten tijde van het invoeren van de code zult u snappen dat `a` staat voor 'aantal' en `p` voor 'prijs'. Maar weet u dit ook nog als u de code na enkele maanden nog eens bekijkt om wijzigingen aan te brengen? Of wat als u – in een langere code – de variabele `a` tien regels verderop

per ongeluk nog eens gebruikt voor iets anders? Het is daarom beter om logische namen te kiezen, zoals:

```
aantal = 5  
prijs = 2.95  
print(aantal * prijs)
```

Het is wat meer typewerk, maar u doet er uzelf (en eventuele andere programmeurs) een groot plezier mee als uw code duidelijk is.

5.6 Statements en bewerkingen

We gaan even terug naar ons eerste voorbeeld:

```
A = 1  
B = 5  
C = A + B  
PRINT C
```

Op basis van de voorgaande informatie zouden we kunnen zeggen dat we hier drie variabelen zien (met bijbehorende waarde) en dat we de computer de opdracht geven iets met de variabele C te doen. We zouden deze vier regels echter ook vier **statements** kunnen noemen. Het zijn namelijk allemaal instructies die door de Python-interpreter kunnen worden uitgevoerd:

Statement	Instructie
A = 1	"Onthoud dat de waarde van A gelijk is aan 1"
B = 5	"Onthoud dat de waarde van B gelijk is aan 5"
C = A + B	"Onthoud dat de waarde van C gelijk is aan A + B"
PRINT C	"Toon de waarde van C op het scherm"

Een bijzondere vorm van een **statement**, is een **bewerking**. Dit is een **statement** waarbij een waarde of variabele wordt veranderd. Er zijn verschillende soorten bewerkingen, zoals een wiskundige bewerking:

```
a = b + 1
```

Er is ook een **string**-bewerking:

```
mijn_tekst = "Hallo"  
mijn_tekst = mijn_tekst + "Wereld"
```

En een **boolean**-bewerking:

```
mijn_variabele = True  
mijn_andere_variabele = not(mijn_variabele)
```

Het **print**-statement is het eenvoudigste **statement** om uitvoer te creëren. U heeft het al een aantal keren in deze cursus gebruikt. Daarover gaat de volgende paragraaf.

5.7 Print-statement

De eenvoudigste vorm van dit **statement** is het rechttoe rechtaan printen van een getal of **string**, bijvoorbeeld:

Input	Output
print(2)	2
print("Hallo")	Hallo

U heeft in deze cursus echter ook al gezien dat we variabelen kunnen printen. Bijvoorbeeld:

Input	Output
a = 10 print(a)	10
mijn_string = "Computer" print(mijn_string)	Computer

Een combinatie is ook mogelijk, bijvoorbeeld:

Input	Output
a = 10 print("resultaat:", a)	resultaat: 10
mijn_string = "Computer" print("Woord van de dag: " + mijn_string)	Woord van de dag: Computer



Merk op dat de komma in het eerste voorbeeld een spatie genereert tussen het eerste en tweede element. In het tweede voorbeeld is die spatie handmatig toegevoegd achter dag::.

Het `print`-statement heeft een aantal variaties en extra mogelijkheden. Twee ervan zullen we in de volgende subparagrafen introduceren.

5.7.1 Printen met een separator

Op de eerste plaats is het mogelijk om twee elementen te printen met een **separator**. Dit is een karakter (of serie karakters) die tussen de elementen in wordt geprint. Bijvoorbeeld:

Input	Output
a = "Maandag" b = "Dinsdag" c = "Woensdag" print(a,b,c, sep="-")	Maandag-Dinsdag-Woensdag

5.7.2 Formatted string

Het is ook mogelijk om variabelen te integreren in een `print`-statement. We noemen dit een **formatted string**.

Een voorbeeld:

Input	Output
a = "Maandag" b = "Dinsdag" c = "Woensdag" print(f"Eerst komt {a}, dan komt {b} en dan komt {c}.")	Eerst komt Maandag, dan komt Dinsdag en dan komt Woensdag.

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. De *interpreter* vult tijdens het printen de waarden van deze variabelen in.

5.8 Loops

We beginnen deze paragraaf met een voorbeeld:

Stel, u werkt voor NASA en moet een aftelprocedure programmeren voor een raket die binnenkort gelanceerd zal worden.



Uw code (en de bijbehorende output) zou er dan als volgt kunnen uitzien:

Input	Output
print("10")	10
print("9")	9
print("8")	8
print("7")	7
print("6")	6
print("5")	5
print("4")	4
print("3")	3
print("2")	2
print("1")	1
print("Lift off!")	Lift off !

Dit programma functioneert zoals het zou moeten functioneren. Het schrijven van deze code is echter nogal wat (repetitief) werk. Dat moet anders kunnen, zeker omdat computers erom bekendstaan repetitief werk efficiënt te kunnen uitvoeren.

Bij deze code hadden we beter gebruik kunnen maken van een **iteratie**. Deze term geeft aan dat iets kan worden uitgevoerd met een bepaald zich herhalend proces.

In de programmeerwereld slaat de term 'iteratief' meestal op het programmeren van herhaling met gebruik van zogenaamde **lussen** (Engels: *loops*). Er zijn verschillende soorten *loops* die we kunnen gebruiken in Python. In de volgende subparagrafen laten we er twee zien.

5.8.1 for-loop

De eerste *loop* is de **for-loop**. We zullen deze *loop* gebruiken in ons NASA-voorbeeld:

Input	Output
<pre>for i in range (10, 0, -1): print(i) print("Lift off!")</pre>	10 9 8 7 6 5 4 3 2 1 Lift off !

Om hetzelfde resultaat te krijgen, waren er nu slechts drie regels code nodig. We zullen regel voor regel uitleggen wat de betekenis is:

1. `for i in range (10, 0, -1):`

Vrij vertaald zegt u met deze regel het volgende:

Voor variabele i, die gaat van 10 tot 0, met stapjes van -1, voer het volgende uit:

Tussen haakjes staan drie getallen, gescheiden door komma's. Achtereenvolgens zijn dit: het begin van de reeks, het eind van de reeks en de stapwaarde. De dubbele punt (:) geeft aan dat er een of meer statements uitgevoerd zullen worden, elke keer als we door de loop gaan.

2. `print(i)`

De tweede regel komt na de dubbele punt. Een regel na een dubbele punt is altijd ingesprongen. Iedere ingesprongen regel (hier: één regel) wordt uitgevoerd bij elke iteratie. In dit geval betekent dit dat iedere keer variabele i wordt geprint.

3. `print("Lift off!")`

Als er geen iteratie meer mogelijk is (omdat het eind van de reeks is bereikt), zal de tweede regel niet meer worden uitgevoerd. Het is dan de derde regel die wordt uitgevoerd, in dit geval het printen van de tekst Lift off!.

Graag ook uw aandacht voor het volgende voorbeeld:

Input	Output
<pre>for i in range (1,3): print("enkel:", i) print("dubbel:", i*2) print("En we zijn weer uit de loop")</pre>	enkel: 1 dubbel: 2 enkel: 2 dubbel: 4 En we zijn weer uit de loop

Wellicht vraag u zich na het zien van deze input en output het volgende af:

- **Waarom staan er tussen de haakjes geen drie getallen maar twee?**

Als de stapwaarde 1 is, kunnen we deze waarde weglaten uit de for-loop. De getallen 1 en 3 geven hier dus het begin en het eind van de reeks aan.

- **Waarom genereert de iteratie vier regels en geen zes regels?**

Het begin van de reeks is dus 1, het einde van de reeks 3. U zou dus verwachten dat we deze drie getallen itereren en de output er dus zo uit zou zien:

```
enkel: 1
dubbel: 2
enkel: 2
dubbel: 4
enkel: 3
dubbel: 6
En we zijn weer uit de loop
```

De eindwaarde van de reeks (hier: 3) wordt niet als iteratie meegenomen. Dat is wellicht duidelijker te zien in het NASA- voorbeeld, waarin we aftellen van 10 naar 0: de eindwaarde 0 wordt nooit bereikt. Misschien niet logisch, maar dat is nu eenmaal hoe de **for-loop** werkt in Python.

De **for-loop** heeft nog meer mogelijkheden, maar voorlopig is dit het laatste voorbeeld:

Input	Output
<pre>mijn_string = "Vanille" for i in mijn_string: print(i)</pre>	V a n i l l e

Dit voorbeeld laat zien dat u een **for-loop** ook kunt gebruiken om letter voor letter door een *string* te *lopen*.

5.8.2 while-loop

Een tweede *loop* in Python is de **while-loop**. Ook deze *loop* leggen we uit aan de hand van een voorbeeld:

Input	Output
<pre>i = 0 while i < 6: print(i) i = i + 1 print("eind")</pre>	0 1 2 3 4 5 eind

Opnieuw zullen we u regel voor regel uitleggen wat de betekenis is:

1. **i = 0**

In dit voorbeeld is eerst de variabele **i** gedefinieerd. Deze variabele heeft de waarde 0.

2. **while i < 6**

Vrij vertaald zegt u met deze regel het volgende:

Zolang variabele i kleiner is dan 6, voer het volgende uit:

De dubbele punt (:) geeft ook bij deze *loop* aan dat er een of meer *statements* uitgevoerd zullen worden, elke keer als we door de *loop* gaan.

3. **print(i)**

4. **i = i + 1**

Deze regels komen na de dubbele punt. Regels na een dubbele punt zijn altijd ingesprongen. U weet inmiddels dat al deze ingesprongen regels worden uitgevoerd bij elke iteratie. In dit geval betekent dit dat iedere keer variabele **i** wordt geprint en de waarde van variabele **i** met 1 wordt opgehoogd. Zouden we de vierde regel niet hebben toegevoegd, dat zouden we in een oneindige *loop* uitkomen, want dan zou **i** altijd kleiner zijn dan 6.

5. **print("eind")**

Zodra **i** niet langer kleiner is dan 6, wordt de iteratie gestaakt en gaat het programma verder met het uitvoeren van de eerste niet-ingesprongen regel. Hier betekent dit dat de tekst **eind** wordt geprint. (Merk overigens op dat de waarde 6 niet wordt meegenomen in de iteratie!)

Het grote verschil tussen de **for-** en **while-loop** is dus dat u bij de **for-loop** aangeeft hoe vaak iets herhaald moet worden, terwijl een **while-loop** stopt als er aan een door u gedefinieerde eis is voldaan. U gebruikt dus een **for-loop** als u op voorhand al weet hoe vaak u een bepaald stuk code wil herhalen. Als het aantal *loops* afhankelijk is van een bepaalde voorwaarde en u weet nog niet precies hoe vaak u de code wil herhalen, dan gebruikt u een **while-loop**.

5.9 Commentaar toevoegen

Misschien bent u iemand die zichzelf of anderen graag aan iets herinnert of ergens op wijst door opmerkingen achter te laten, bijvoorbeeld op de volgende manier:



Als u een programma gaat schrijven, zult u deze wens waarschijnlijk ook vaak hebben. Losse briefjes zijn dan niet erg handig. Gelukkig kunt u uw opmerkingen direct in de code plaatsen.

Opmerkingen die u in uw code plaatst, ook wel **commentaar** genoemd, worden niet uitgevoerd door de *interpreter*. Ze zijn vooral bedoeld om de code uit te leggen of om latere werkzaamheden aan het bestand te vergemakkelijken.

Ook kunnen opmerkingen nuttig zijn als het nodig is uw code te testen: u kunt bijvoorbeeld toelichten wat u bij een bepaalde test heeft gedaan of delen van uw code tijdelijk verbergen voor de *interpreter*.



Het is niet verplicht om opmerkingen toe te voegen, maar het is wel een goede gewoonte. Als u de namen van uw variabelen (en later: functies) goed kiest, zouden er echter niet veel commentaarregels nodig moeten zijn om uw code begrijpelijk te laten zijn.

Opmerkingen kunt u op twee manieren 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 wil toevoegen. Alles tussen de *triple quotes* (oftewel: drie enkele aanhalingstekens) wordt genegeerd door de *interpreter*. Bijvoorbeeld:

...

Dit is een commentaar van meerdere regels.

Dit kan handig zijn als u veel informatie wil geven.

...

5.10 Samenvatting

In deze les stonden bouwstenen centraal. Zo zag u dat iemand die programmeert, niet onder het gebruik van variabelen uitkomt. Een variabele heeft een naam en een bijbehorende waarde. Die waarde kan een geheel getal zijn, maar bijvoorbeeld ook een kommagetal, *string* of *boolean*.

Uw code wordt gevormd door *statements*, oftewel instructies die door de *interpreter* kunnen worden uitgevoerd (zoals ‘onthoud dit’ of ‘toon dat’). Een bijzondere vorm van een *statement* is een bewerking: een *statement* waarbij een waarde of variabele wordt veranderd. We hebben in deze les vooral gekeken naar het *print*-*statement* in zijn simpelste vorm en enkele variaties.

Daarnaast maakte u in deze les kennis met *loops*. Die kunnen ervoor zorgen dat u veel minder hoeft te coderen en het codeerwerk minder repetitief is. We hebben aandacht besteed aan de *for*- en *while-loop*. Het grote verschil tussen deze *loops* is wanneer de herhaling stopt: na een x-aantal keer of zodra aan een eis is voldaan.

Tot slot leerde u hoe u aan uw code commentaar kunt toevoegen. Dat kan op twee manieren: met een *hashtag* of met *triple quotes*.

5.11 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.

5.12 Oefenopgaven

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

Wat bedoelen we met ‘opdrachten worden in lineaire volgorde uitgevoerd’?

→ Bekijk antwoord

Wat is een variabele?

→ Bekijk antwoord

Geef aan of de volgende variabelennamen geldig of ongeldig zijn in Python.

- a. mijn_ijsje1
- b. 1e_getal
- c. prijs-invoer

d. dit_is_mijn_extra_lange_variabele

e. for

→ Bekijk antwoord

Schrijf een programma dat voldoet aan de volgende eisen:

- Het programma bevat drie variabelen.
- De eerste variabele heeft de naam prijs1 en de waarde € 1,95.
- De tweede variabele heeft de naam prijs2 en een waarde € 2,25.
- De derde variabele heeft de naam totaal. De waarde van deze variabele is een optelling van de eerste twee waarden.

Benoem tevens of de waarde van de variabele totaal een int of een float is.

→ Bekijk antwoord

Waarom is het beter om een variabele leeftijd te noemen in plaats van te kiezen voor de kortere naam l?

→ Bekijk antwoord

Wat is een *statement*?

→ Bekijk antwoord

Wat is een bewerking?

→ Bekijk antwoord

Wat wordt in de programmeerwereld bedoeld met 'iteratie'?

→ Bekijk antwoord

Wat is het verschil tussen de **for**- en **while-loop**?

→ Bekijk antwoord

Hoe kunt u gemakkelijk zien welke regels behoren tot een *loop*?

→ Bekijk antwoord

Welk karakter zorgt ervoor dat de Python-*interpreter* alle tekst erachter niet uitvoert?

→ Bekijk antwoord

Bij welke *loop* loopt u het risico dat u een 'oneindige loop' creëert: de **for**- of **while-loop**?

→ Bekijk antwoord

Schrijf een programma dat de volgende output genereert:

```
100  
90  
80  
70  
60  
50  
40  
30  
20  
10
```

Maak gebruik van een *loop*.

→ Bekijk antwoord

Bekijk het volgende voorbeeld:

```
# we printen de getallen 2 tot en met 7  
for i in range(2,8):  
    print(i)
```

Waarom hebben we in dit voorbeeld geen stapwaarde gedefinieerd?

→ Bekijk antwoord

5.13 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. Schrijf een programma dat voldoet aan de volgende eisen:

- Het programma bevat drie variabelen.
- De eerste variabele heeft een logische naam naar keuze en de waarde '10'.
- De tweede variabele heeft een logische naam naar keuze en de waarde '0.50'.
- De derde variabele heeft de naam **totaal**. De waarde van deze variabele is een vermenigvuldiging van de eerste twee waarden.

2. Schrijf een programma dat de volgende output zo efficiënt mogelijk genereert:

```
10 maal 2 is 20
11 maal 2 is 22
12 maal 2 is 24
13 maal 2 is 26
14 maal 2 is 28
15 maal 2 is 30
16 maal 2 is 32
17 maal 2 is 34
18 maal 2 is 36
19 maal 2 is 38
```

3. Schrijf een programma dat begint met de volgende code:

```
leeftijd = 30
naam = "Deborah"
```

Voeg vervolgens een regel toe die ervoor zorgt dat de output er als volgt uitziet:

Haar naam is Deborah, ze is 30 jaar oud.