

Stroomschema's maken op papier



Een programma direct maken in *Python*, gaat vaak wel goed als het een klein programma is. Als het programma groter en moeilijker is, is het lastig om goed te zien welk commando waar moet komen. Je maakt dan sneller fouten.

Het is ook moeilijker om het programma dan netjes te maken. Een programma dat netjes in elkaar zit, werkt sneller en beter dan een programma dat rommelig is. Zo netjes mogelijk programmeren wordt *gestructureerd programmeren* genoemd. Een structuur is een ander woord voor hoe iets in elkaar zit.

Om je een handje te helpen bij het maken van een goed programma, gebruik je een *stroomschema*. Een stroomschema is een soort tekening. De tekening bestaat uit verschillende vakjes die met pijltjes met elkaar verbonden zijn. In de vakjes staan de commando's die een programma moet uitvoeren.

Met een stroomschema maak je eerst een oefenversie van een programma op papier. Of op de computer in een tekstverwerkingsprogramma, zoals *Word*, of in een tekenprogramma, zoals *Paint*. Omdat alles in losse vakjes staat, is duidelijk te zien hoe het programma werkt. Het stroomschema gebruik je daarna om stap voor stap je programma in *Python* te maken.

In dit aanvullende bestand wordt uitgelegd hoe je een stroomschema op papier maakt.

In dit hoofdstuk leer je:

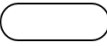
- een stroomschema starten en eindigen;
- pijlen toevoegen;
- een activiteit toevoegen;
- invoer en uitvoer toevoegen;
- een toelichting toevoegen;
- beslissingen toevoegen;
- lussen toevoegen;
- een functie toevoegen.

Een stroomschema starten

In dit voorbeeld wordt het stroomschema gemaakt op papier. Het is handig als dit een groot vel papier is. Anders kan je ook verschillende velletjes aan elkaar plakken of het stroomschema op een aantal losse velletjes maken.

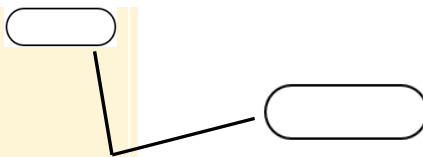
Daarnaast heb je een potlood nodig. Een pen kan ook, maar een potlood is handiger als je iets wilt uitgummen. Bijvoorbeeld wanneer je een fout maakt. Om de pijltjes te tekenen gebruik je een liniaal of een geodriehoek. Er zijn ook speciale plastic sjablonen waarin alle vormen voor een stroomschema zitten, zoals rechthoeken. Daarmee teken je een stroomschema het meest netjes.

Pak een vel papier, een potlood en een liniaal of geodriehoek

Een stroomschema begint altijd met een *start*-commando. Daar gebruik je deze *start*-vorm  voor. Met de start-vorm maak je duidelijk waar het stroomschema, en dus het programma, begint. Het is geen commando dat je verder in *Python* gebruikt bij het programmeren.

Je voegt de start-vorm toe aan je vel papier. Het is slim om dat tussen de linkerkant en het midden van het vel te doen. Je houdt dan genoeg ruimte links en rechts over voor de rest van het stroomschema. De meeste ruimte moet je altijd rechts houden:

Teken de vorm



Je zet ook nog een tekst in de vorm. Daarmee vertel je wat er gebeurt in je programma. Je mag zelf de tekst verzinnen, als hij maar duidelijk genoeg is. In de start-vorm zet je meestal de tekst *Start*:

Schrijf in de vorm:

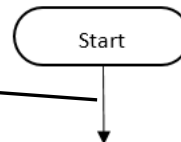
Start



Een pijl toevoegen

Voordat je een volgende vorm in je stroomschema zet, teken je een pijl. Die pijl laat zien in welke volgorde het programma wordt uitgevoerd. Gewoonlijk is dat van boven naar beneden. Onder de pijl zet je dan straks de volgende vorm. Je zet een pijl altijd ongeveer in het midden van een vorm:

 **Teken een pijl naar beneden**

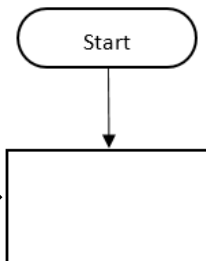


Een activiteit toevoegen

Als het begin van het stroomschema klaar is, voeg je de rest van het programma toe. Gewone commando's, zoals het geven van een waarde aan een variabele, voeg je toe met een *activiteit-vorm* . Die vorm zet je onder de pijl, recht onder de eerste vorm:

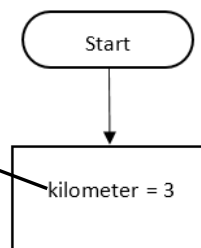
 **Teken de vorm**

De vorm hoort ongeveer midden onder de pijl te staan:



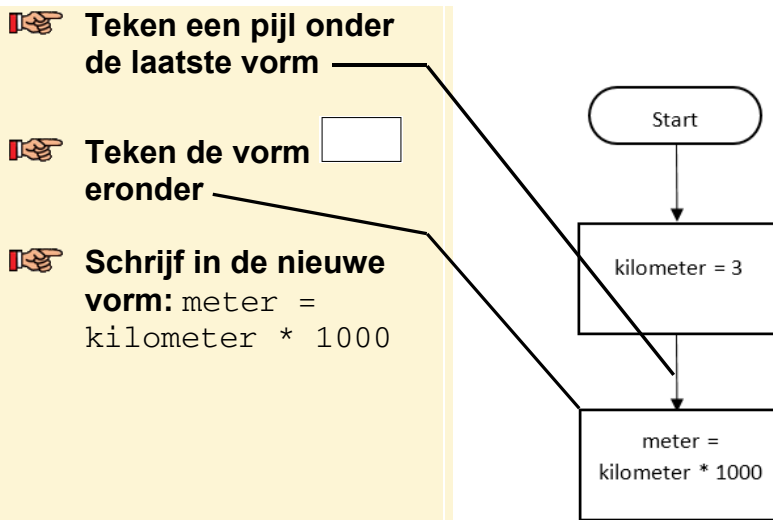
Je zet nu nog een tekst in de activiteit-vorm. Dit is het commando dat in het programma uitgevoerd wordt, bijvoorbeeld `kilometer = 3`. Je hoeft niet helemaal precies de tekst van een commando in *Python* op te schrijven. Het gaat erom dat het duidelijk is wat er in het programma gebeurt:

 **Schrijf in de vorm:**
`kilometer = 3`



Je hebt een activiteit-vorm gemaakt.

Je kunt nog meer activiteit-vormen onder elkaar zetten in je stroomschema. Je voegt een activiteit-vorm toe voor het commando `meter = kilometer * 1000`:



Op dezelfde manier maak je een heel stroomschema voor een eenvoudig programmadeel. In dit voorbeeld heb je de eerste twee commando's van het programma in *paragraaf 2.5 Kilometers omrekenen met numerieke variabelen* in het boek gemaakt.

Uitvoer toevoegen

Invoer (input) en uitvoer (output) van gegevens zijn belangrijk in een programma. Bijvoorbeeld het typen van een getal of het laten zien van de uitkomst van een som.

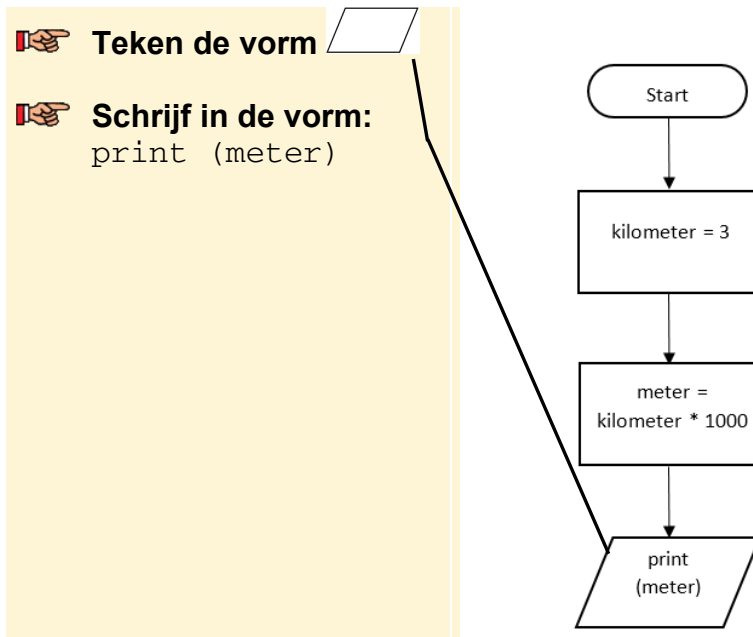
Voor invoer en uitvoer gebruik je dezelfde *invoer/uitvoer-vorm* .

In het voorbeeld voeg je een uitvoer-vorm toe voor `print (meter)` in *paragraaf 2.5 Kilometers omrekenen met numerieke variabelen*.

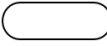
📌 Teken een pijl omlaag onder

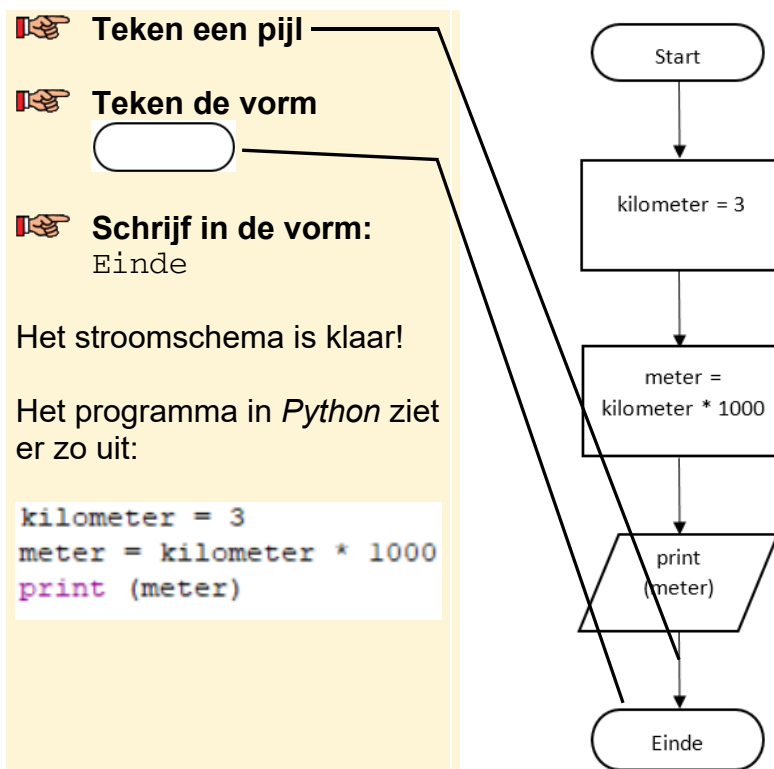
```
meter =
kilometer * 1000
```

Je zet een tekst in de uitvoer-vorm. Dat is het commando *print (meter)*:



Het einde toevoegen

Net zoals je laat zien waar het programma start, laat je ook zien waar het programma stopt. Daar gebruik je de *einde-vorm*  voor:



Je kunt het stroomschema nu gebruiken om je programma mee te maken.



Tip

Een programma ergens anders stoppen

Je hoeft een programma niet altijd onderaan te stoppen. Je kunt het ook op een

Einde

andere plek stoppen met het commando `sys.exit()` in *Python*. Je zet dan op die plek in het stroomschema. Een stroomschema kan zo meerdere

Einde

vormen hebben.

Commentaar toevoegen

Soms is het handig om wat extra informatie bij een vorm in een stroomschema te zetten. Bijvoorbeeld omdat de tekst niet helemaal in de vorm past of omdat de tekst niet duidelijk genoeg is. Die extra informatie zet je in een *commentaar* of een *toelichting*. In *Python* zet je daarvoor een hekje `#` voor een tekst. Je gebruikt in een stroomschema de vorm `/` .

In dit voorbeeld zet je een commentaar bij . Daarin zet je dat het programma kilometers omrekent naar meters.

Je zet een toelichting rechts naast de vorm waar hij bij hoort:

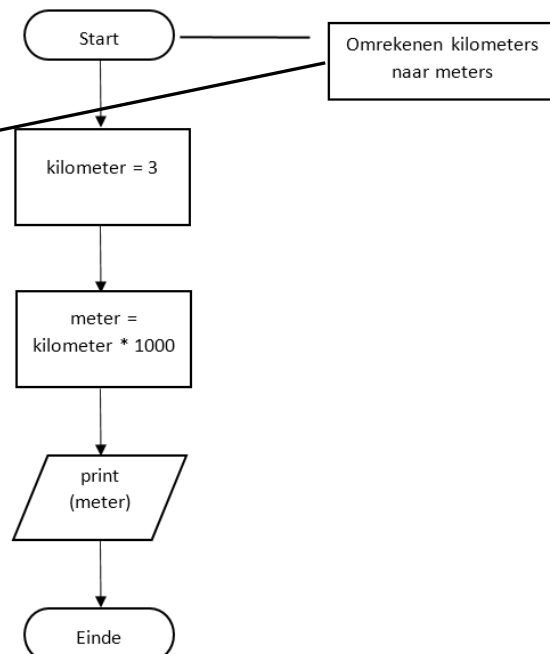
Teken de vorm

`/` rechts naast

Start

Schrijf in de vorm:

Omrekenen
kilometers naar
meters

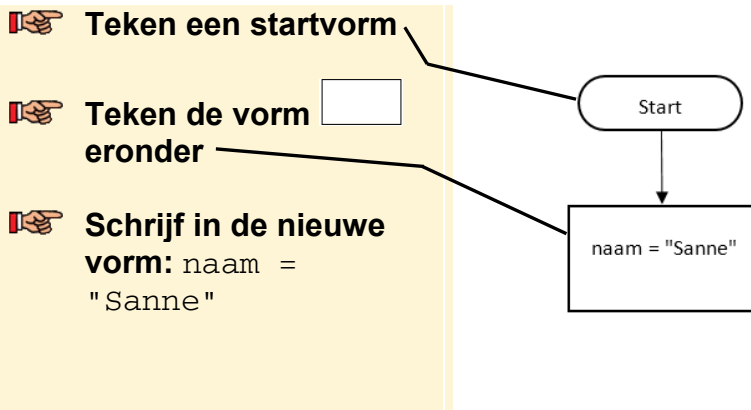


Het commentaar en je stroomschema is klaar.

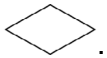
Een beslissing-commando toevoegen

Je maakt een nieuw stroomschema voor een ander programma. Daarvoor kan je bijvoorbeeld op de achterkant van het papier verdergaan of je pakt een nieuw vel papier.

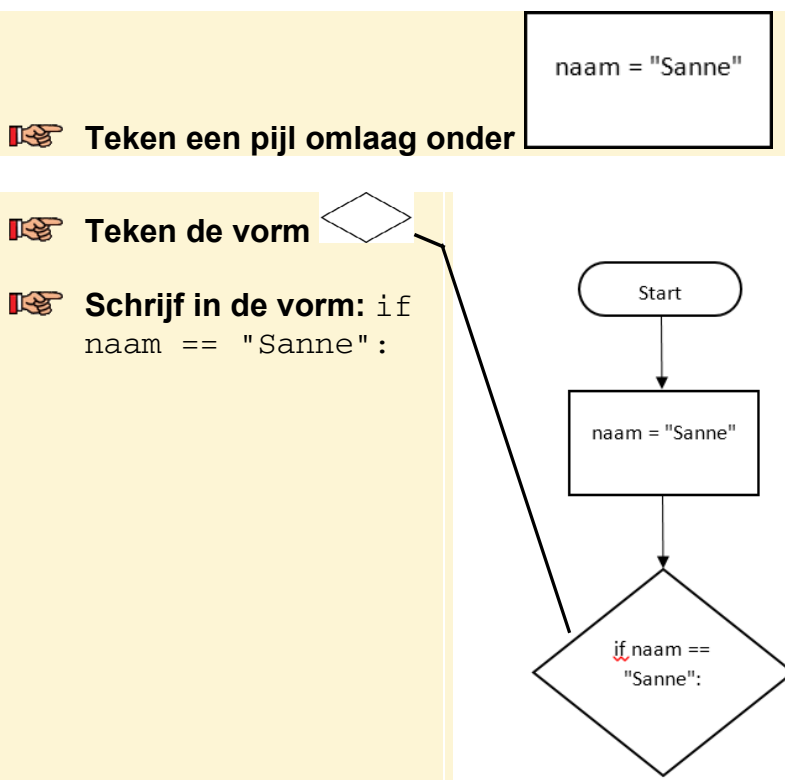
In dit stroomschema maak je het programma in *paragraaf 3.2 Beslissen met het if commando* in het boek:



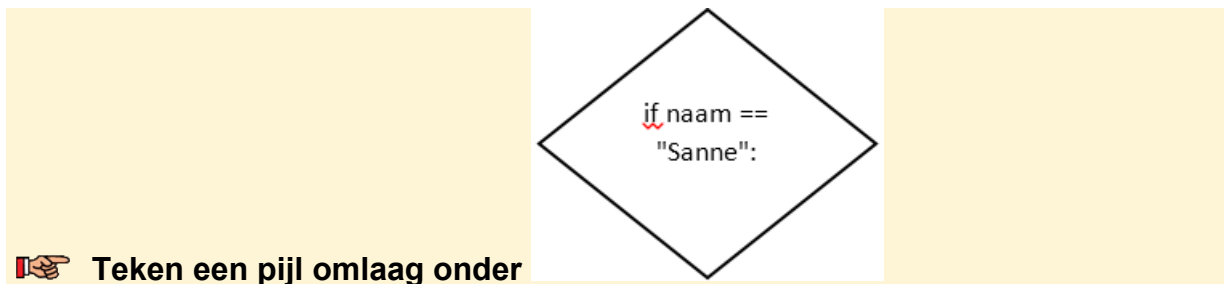
In een programma gebruik je vaak beslissing-commando's. In *Python* zijn dit *if*-commando's. Om die in je stroomschema te zetten, gebruik je de *beslissing-vorm*



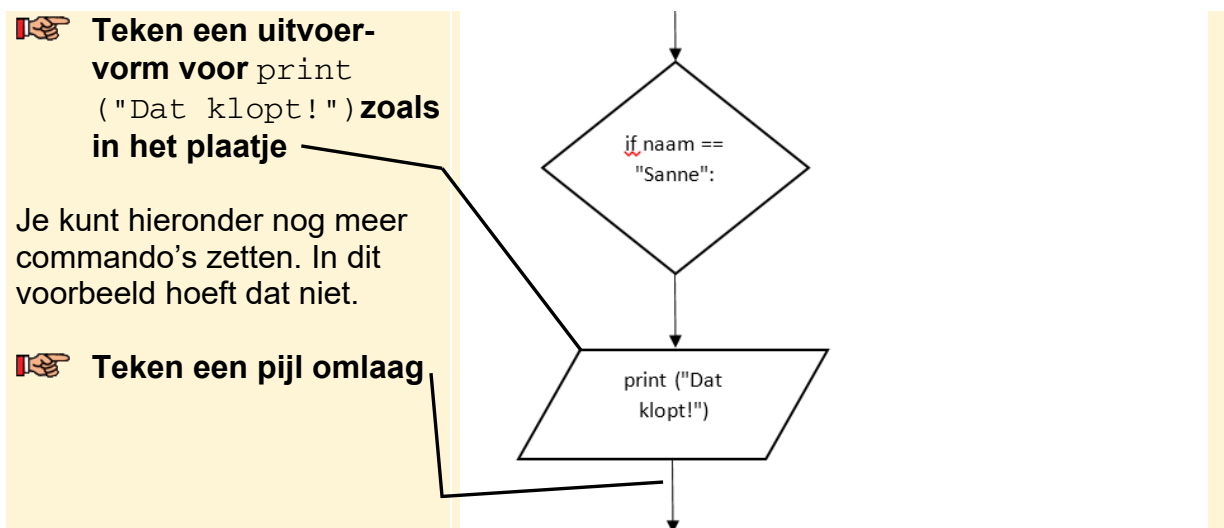
Je zet de beslissing-vorm onder aan het stroomschema en zet er een tekst in. Dat is de vergelijking waar de beslissing aan moet voldoen. In dit voorbeeld is dat het commando *if naam == "Sanne"*:



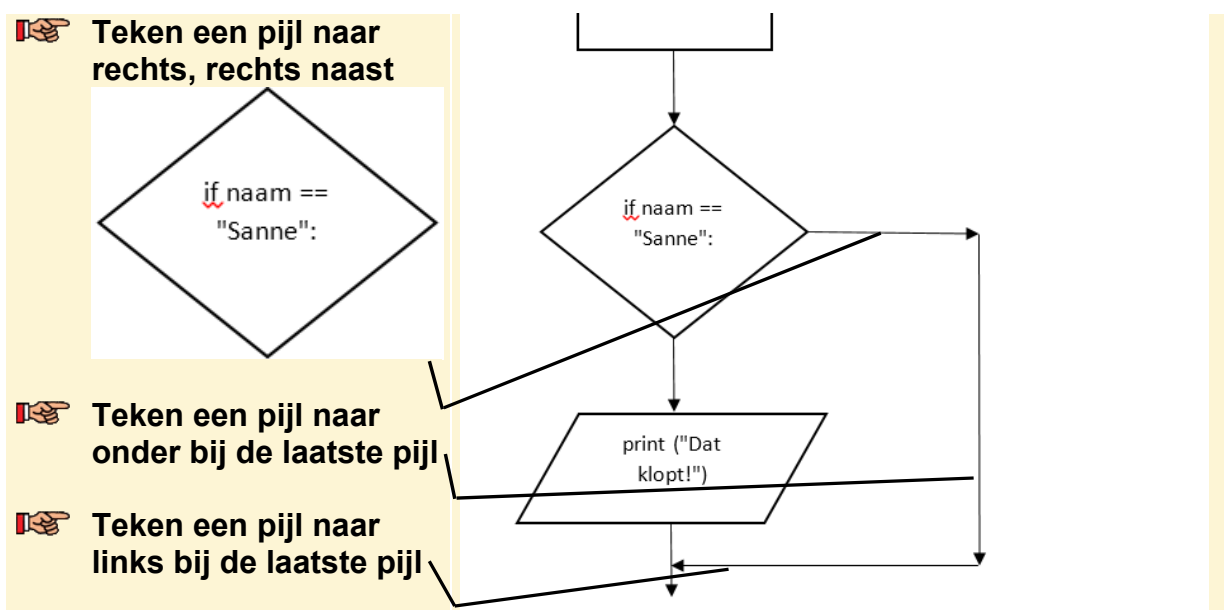
Tot nu toe voegde je bij een vorm een pijl toe. Bij een beslissing-vorm voeg je twee pijlen toe. Een voor als de vergelijking klopt. Die loopt recht naar beneden:

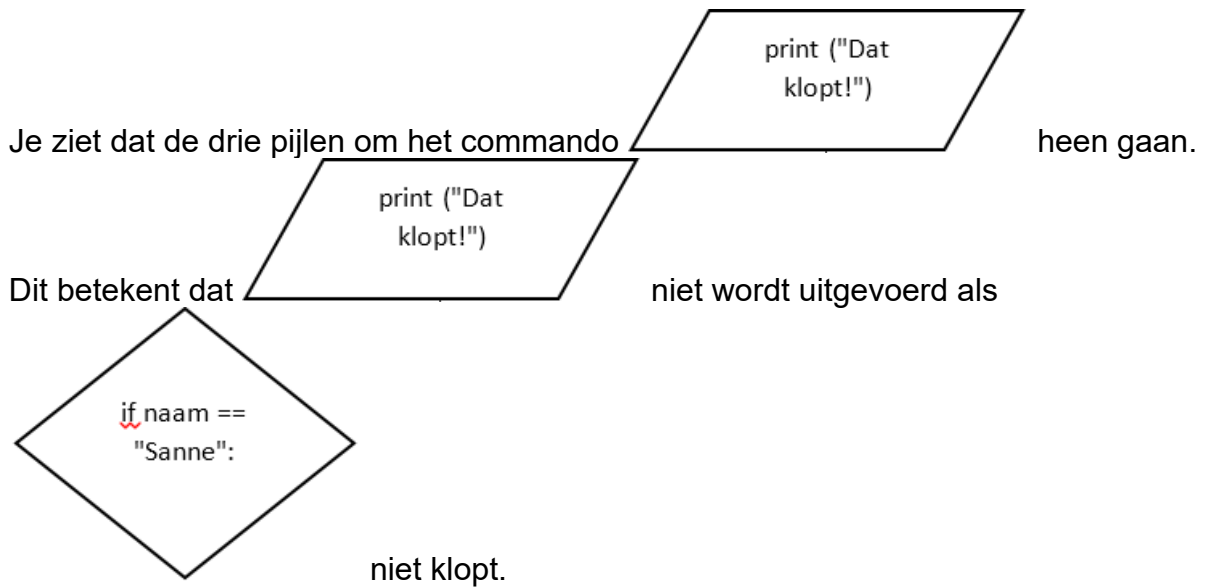


Hieronder zet je de commando's die worden uitgevoerd als de vergelijking klopt. In dit voorbeeld is dat het uitvoer-commando *print ("Dat klopt!")*:

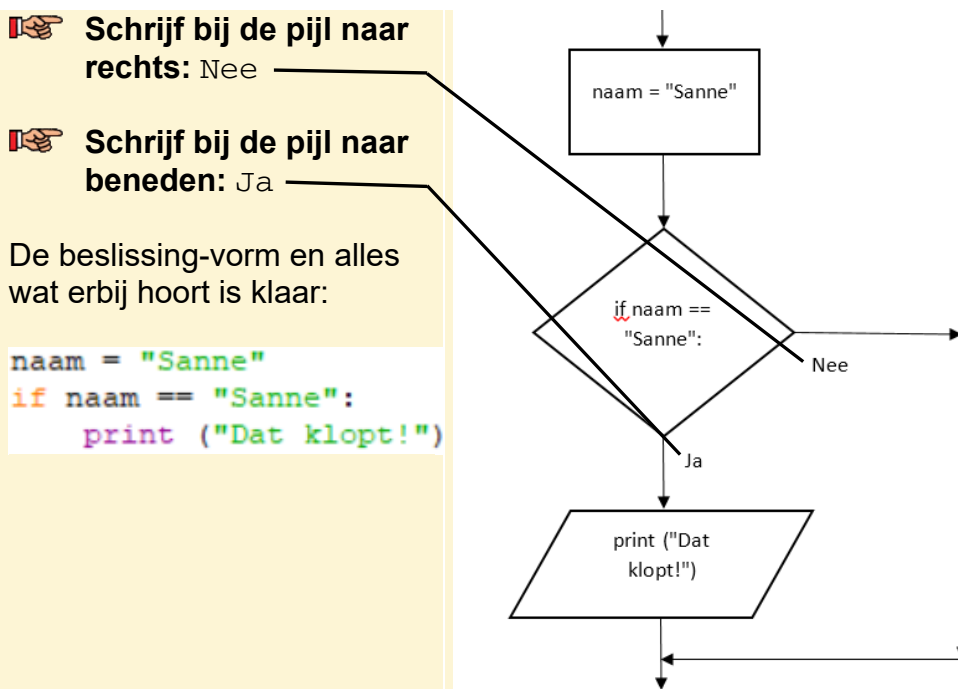


De andere pijl zet je aan de rechterkant van de beslissing-vorm. Deze geeft de route aan als de vergelijking niet klopt. In dit voorbeeld slaat de pijl het commando *print* over. Hiervoor gebruik je een aantal pijlen of een lange pijl met hoeken erin:





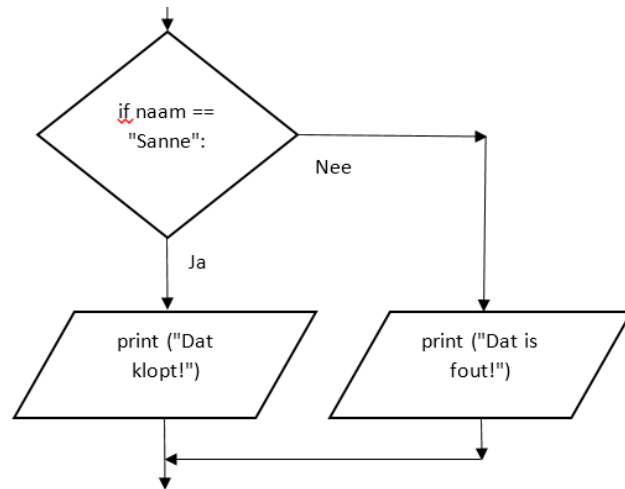
Je geeft nu nog bij de beslissing-vorm aan wanneer de pijl omlaag en wanneer de pijl naar rechts wordt genomen. Daarom zet je bij de route omlaag *Ja* en de route naar rechts *Nee*.



Als er nog meer commando's in een programma zijn, zet je die gewoon onder de laatste pijl.

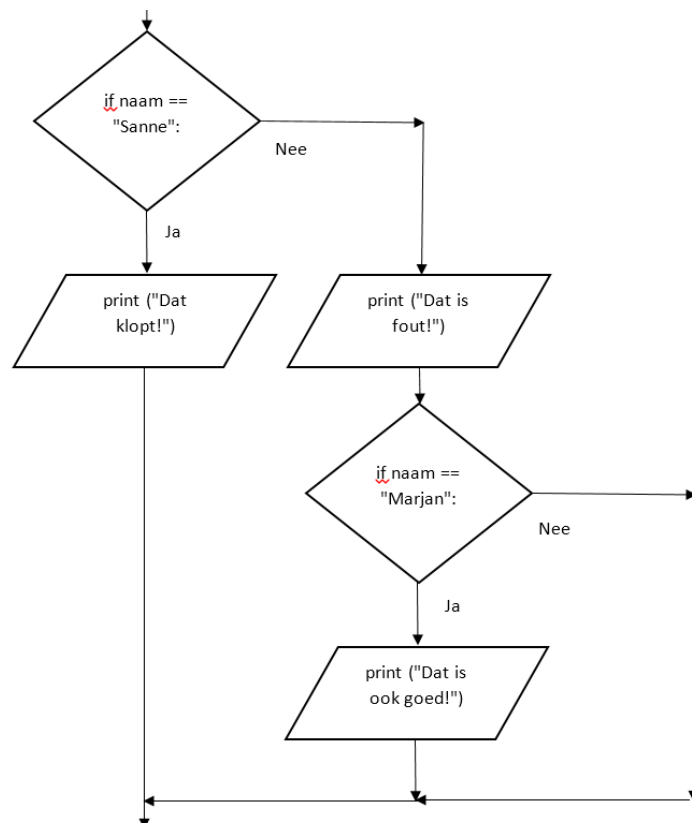
Een stroomschema maken voor een *if/else* commando werkt bijna hetzelfde als voor een *if* commando. Het enige verschil is dat je ook bij de route voor *Nee*-commando's neerzet. Dat ziet er dan bijvoorbeeld zo uit:

```
if naam == "Sanne":
    print ("Dat klopt!")
else:
    print ("Dat is fout!")
```



Je kunt in *Python* ook *elif* gebruiken bij beslissingen:


Dat ziet er dan bijvoorbeeld zo uit:

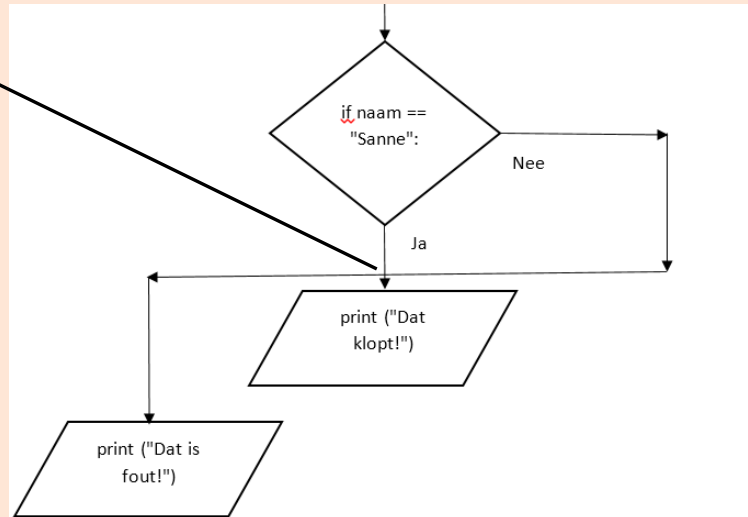


Let op!

Als je goed en netjes wilt programmeren, zorg je er voor dat de pijlen in je stroomschema elkaar niet kruisen:

Als de pijlen elkaar wel kruisen:


 **Verander je stroomschema, zodat de pijlen wel goed lopen**




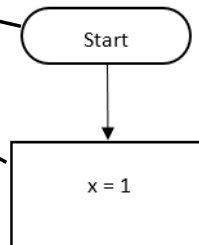
Invoer toevoegen

Je maakt een nieuw stroomschema voor het programma in *paragraaf 4.2 Een while lus maken*:


 **Maak een startvorm**

 **Teken de vorm eronder**

 **Schrijf in de nieuwe vorm: $x = 1$**




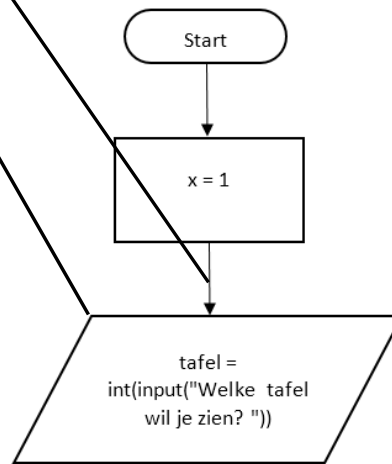
Voor het invoeren van gegevens gebruik je dezelfde vorm als voor uitvoer:  :

 **Teken een pijl omlaag**

 **Teken de vorm onder de pijl**

Je zet een tekst in de invoer-vorm:

 **Schrijf:** `tafel = int(input("Welke tafel wil je zien? "))`




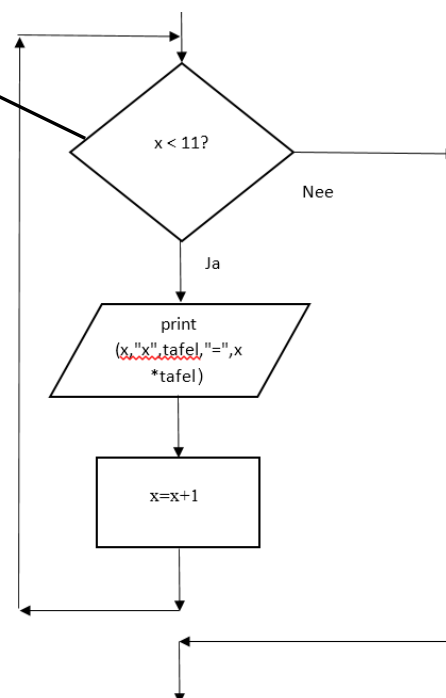
Een lus toevoegen

Een lus is eigenlijk een beslissing-vorm die ieder keer bij zichzelf terugkomt. Dat gaat net zo lang door totdat de vergelijking klopt. Daarna ga je in het stroomschema verder met het volgende commando:

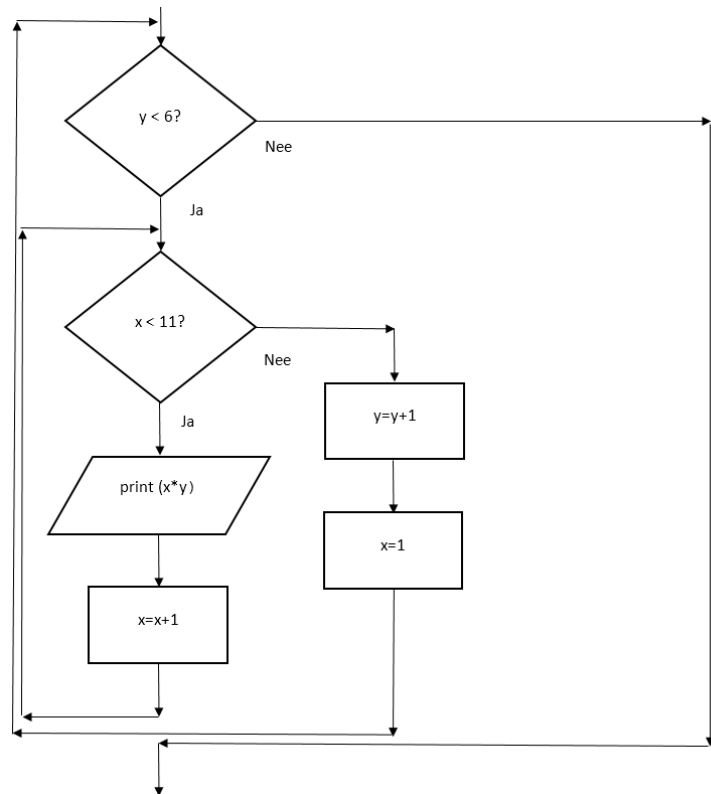
Als je bijvoorbeeld voor dit programmadeel een stroomschema wilt maken:

```
while x < 11:
    print (x, "x", tafel, "=", x*tafel)
    x = x + 1
```


 **Teken dit stroomschema**



Als je een geneste lus gebruikt, ziet dat er bijvoorbeeld zo uit:


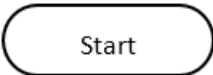


Een functie toevoegen

Je kunt delen van een programma in een functie zetten. In *Python* start je de functie met de naam, bijvoorbeeld `rechthoek()`. In een stroomschema gebruik je daarvoor de *procedure*-vorm .


 **Kijk of je nog ruimte hebt op je papier of pak een nieuw vel papier**

 **Teken een  vorm**

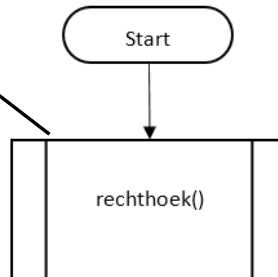
 **Teken een pijl omlaag onder **

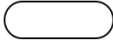
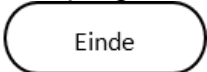
Je noemt de functie `rechthoek()` zoals in *paragraaf 7.2 Maak het jezelf makkelijk met je eigen functies*:

 **Teken de vorm** 

 **Schrijf:** `rechthoek()`

Tussen de haakjes zet je de parameters die je meestuurt, zoals
`rechthoek(lengte,breedte)`



Voor de functie zelf gebruik je de start-vorm . Alleen zet je daar geen *Start* in maar de naam van de functie. In dit voorbeeld is dat `def rechthoek()`. De rest van de functie is een deel van het programma en werkt dan ook met dezelfde vormen. Het einde geef je aan met .

De functie zet je op een plek op je papier waar nog lege ruimte is. Dat kan naast het programma zijn of op een ander vel. De functie in *paragraaf 7.2 Maak het jezelf makkelijk met je eigen functies* ziet er in een stroomschema zo uit:

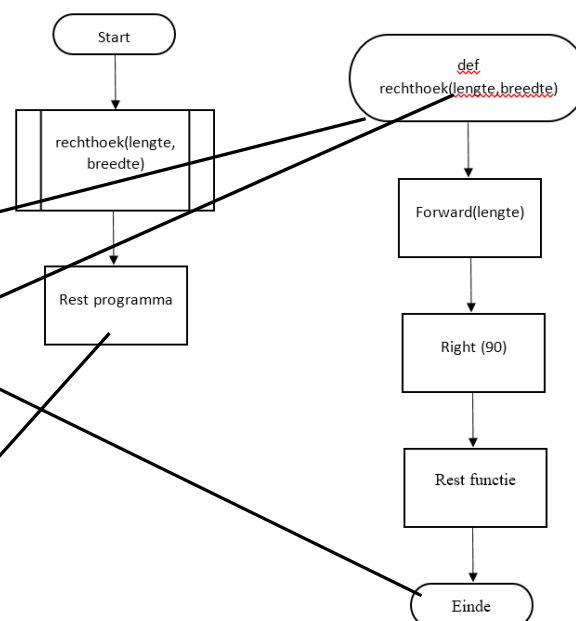
```

def rechthoek(lengte,breedte):
    forward(lengte)
    right(90)
    forward(breedte)
    right(90)
    forward(lengte)
    right(90)
    forward(breedte)
  
```

Parameters:

Einde van procedure:

Het programma gaat aan het einde van de procedure verder:








Tip

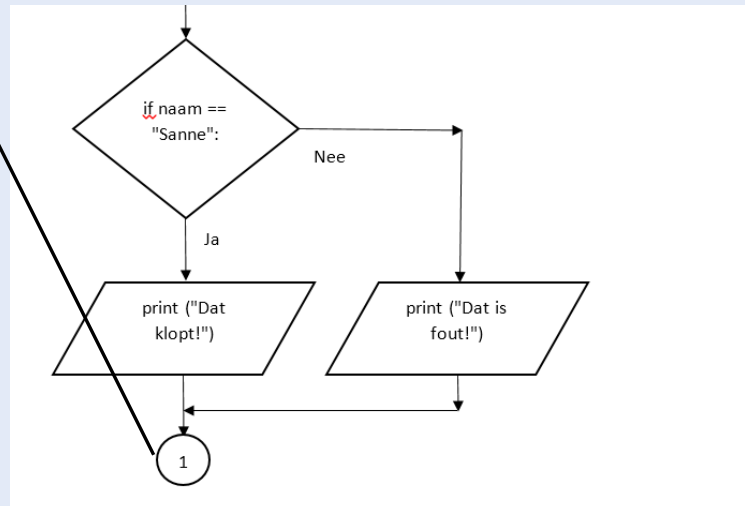
Het stroomschema is langer dan een pagina



Als je stroomschema niet helemaal past op 1 pagina, is dat niet erg. Soms kun je dat oplossen door de pijltjes wat korter te maken. Als dat niet werkt, gebruik je de




connector-vorm .

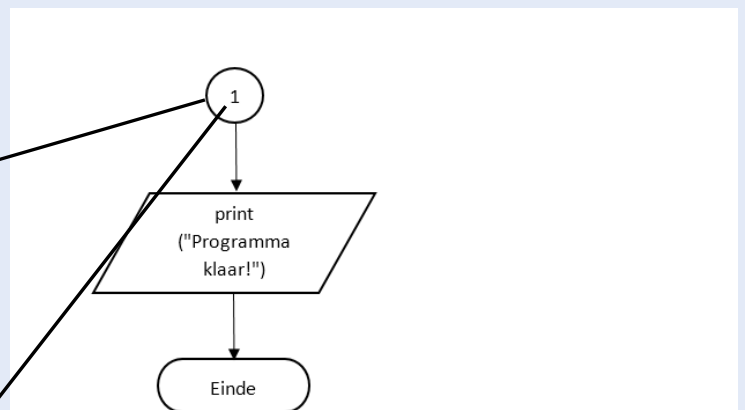
 Voeg  toe als laatste vorm in het schema op de pagina

 Schrijf een nummer in , bijvoorbeeld 1



 Voeg  toe als eerste vorm in het schema op de volgende pagina

 Schrijf hetzelfde nummer in  als in de  vorm op de vorige pagina



Het programma loopt hier nu door:

Je hebt in deze handleiding geleerd hoe je stroomschema's maakt. Je kunt nu voortaan eerst een stroomschema maken voordat je echt gaat programmeren in *Python*. Veel succes!