1 Voorwaardelijke lus: `while`

Het gebeurt wel eens dat je een stukje code meerdere keren wil uitvoeren. Dat is waarvoor een lus gebruikt kan worden. Volgend blokje code toont de opbouw van een lus in python:

```
while voorwaarde:
    # code die wordt uitgevoerd zolang
    # de voorwaarde resulteert in de logische waarde True
    #
    # net zoals bij een if-statement moeten al de regels code
    # die tot de lus behoren ingesprongen zijn
    # de lus loopt door tot de eerste regel code
    # die niet meer ingesprongen is
```

Het `while`-sleutelwoord geeft aan dat je een lus wil starten. Natuurlijk wil je ook niet dat de lus oneindig blijft doorgaan. Daarom geef je een voorwaarde mee aan de lus. Net zoals bij het `if`-statement, is de voorwaarde een expressie tot een logische waarde. Deze expressie wordt geëvalueerd telkens de lus opnieuw start. Dit gaat als volgt:

- Stap 1:
  De voorwaarde wordt geëvalueerd, en het resultaat is:
  - `True`: ga naar stap 2.
  - `True`: ga naar stap 3.
- Stap 2:
  De code in de lus uitgevoerd.
  Ga terug naar stap 1.
- Stap 3:
  De lus wordt niet meer opnieuw uitgevoerd. In de plaats gaat het programma meteen verder vanaf de eerste regel code na de lus.

Meestal gaan we een uitdrukking gebruiken als de voorwaarde van een lus. Maar we mogen ook gewoon een van de twee logische waardes rechtstreeks gebruiken.
Wat zouden de volgende twee lussen doen? Zullen we eens proberen in de interactive shell?

```
while: False
    print("Hallo")
```

```
while: True
    print("Hallo")
```

Tijd voor een echt voorbeeldje, ik denk dat je daar klaar voor bent! Ken je de reeks van Fibonacci? Dat is een getallenrij waarbij ieder getal de som is van de twee voorgaande getallen. De rij begint met tweemaal het getal 1. Wij hebben de eerste getallen al voor jou berekend: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Hebben we ze allemaal juist volgens jou?

Nu we weten wat de reeks van Fibonacci is, kunnen we deze reeks ook berekenen met een lus. We starten met de eerste twee getallen, en gaan proberen om de volgende tien getallen te berekenen.

```
# we starten met de eerste twee getallen van de reeks van Fibonacci
# en houden deze bij in de variabelen a en b
getalM2 = 1             # getal op twee plaatsen terug
getalM1 = 1             # vorige getal

# we houden bij hoeveel getallen we al berekend hebben
# we beginnen bij 2, want de eerste twee getallen kennen we al
getallenBerekend = 2

# we maken een lus die telkens een nieuw getal van de rij berekent
# dit blijven we doen tot we aan 10 getallen zitten
while getallenBerekend < 10:
    # we berekenen het volgende getal
    # door de som te nemen van de vorige twee
    # getallen
    volgendGetal getalM2 + getalM1

    # we tonen het getal
    print(f"Fibonacci getal {getallenBerekend}: {volgendGetal}")

    # in de volgende lus wordt:
    # 1. het vorige getal nu het getal op twee plaatsen terug
    getalM2 = getalM1
    # 2. het nieuwe getal nu het vorige getal
    getalM1 = volgendGetal

    # tot slot tellen we 1 op bij het aantal getallen dat we al
    # berekend hebben, anders riskeren we dat de lus nooit stopt!
    getallenBerekend += 1
print(
    f"We hebben {getallenBerekend} Fibonacci getallen berekend!"
)
```

In het voorbeeld schuiven we de getallen telkens op. Dit doen we hier:

```
getalM2 = getalM1
getalM1 = volgendGetal
```

We hadden ook dit kunnen doen:

```
getalM1 = volgendGetal
getalM2 = getalM1
```

Denk je dat het op deze manier ook gewerkt zou hebben? Waarom denk je van wel, of waarom denk je van niet? Probeer het zeker eens uit in de interactive shell!

Ken je de reeks van de even getallen?
Dit zijn de getallen 0, 2, 4, 6, 8, 10, 12, 14, 16, ...
Denk je dat jij deze reeks kan berekenen met een lus? Ik denk van wel! Voor deze reeks heb je maar één variabele nodig. Je kan starten met het getal 0. In iedere lus tel je er 2 bij op.
Is het je gelukt?

1.1 `break`

Is de voorwaarde van de `while`-lus waar, dan wordt de inhoud van de lus van de eerste tot de laatste regel uitgevoerd. Soms moeten we echter de uitvoering van de lus abrupt kunnen onderbreken. Het `break`-statement is een eerste statement die dit kan. Als de utvoering van het programma het `break`-statement tegenkomt, dan is de lus meteen gedaan. Ook het stukje code van de lus dat na dit `break`-statement komt, wordt zelfs niet meer uitgevoerd!

We zullen een voorbeelde gebruiken om dit duidelijker te maken. We willen het gemiddelde berekenen van 10 getallen die we aan de gebruiker vragen. Als de gebruiker geen 10 getallen heeft, dan zou de lus toch eerder moeten kunnen stoppen. En moet het programma het gemiddelde tonen van slechts deze paar getallen die de gebruiker gegeven heeft. Hieronder hebben we dit vertaald naar een stukje python code:
Zonder het groene gearceerde blok heb je een `while`-lus dat het gemiddelde van exact 10 waardes berekend. Met het groene gearceerde blok kan de gebruiker de lus eerder laten stoppen als hij geen 10 waardes heeft.

```
# we hebben twee variabelen nodig
som = 0                     # de tussentijdse som
aantal = 0                  # het huidige aantal gekregen getallen

while aantal < 10:          # de lus moet doorgaan tot 10 getallen

    # we gebruiken input() om een nieuw getal te vragen
    nieuweInput = input("Geef een getal: ")

    # als de gebruiker geen getallen meer heeft
    # dan vult hij niets meer in
    # in dat geval is nieuweInput een lege tekst
    # dit moeten we eerst controleren voor we verder kunnen
    if nieuweInput == "":
        # de gebruiker heeft geen getallen meer
        # we moeten de lus vroegtijdig stoppen!
        break

    som += int(nieuweInput) # we zetten de tekst om in een getallen
                            # die we dan bij som optellen
    aantal += 1

# tot slot bepalen we het gemiddelde en tonen we deze aan de gebruiker
gemiddelde = som/aantal
print(f"Het gemiddelde van de gegeven getallen: {gemiddelde}")
```

Heb je het `break`*-statement begrepen? Snap je waarom het voorbeeldje hierboven het gemiddelde juist berekend, ook in die gevallen waarbij de gebruiker geen 10 getallen heeft?*
We hebben hetzelfde voorbeeldje hieronder nog eens staan, maar dit keer wordt het aantal aan het begin van de lus verhoogd.
Denk jij dat er nu een andere waarde voor het gemiddelde berekend gaat worden? Of blijft het resultaat hetzelfde?
Probeer het eens uit, kan jij uitleggen wat er juist gebeurt?

```
som = 0
aantal = 0
while aantal < 10:
    aantal += 1

    nieuweInput = input(
        "Geef een getal: "
    )

    if nieuweInput == "":
        break

    som += int(nieuweInput)
gemiddelde = som/aantal
print("Het gemiddelde van de gegeven getallen:")
print(gemiddelde)
```

1.2 `continue`

Standaard wordt de blok code van een lus volledig uitgevoerd tijdens iedere herhaling. Maar soms willen we een herhaling (gedeeltelijk) kunnen overslaan. En dit kunnen we! Wanneer de uitvoering van het programma het `continue`-statement tegenkomt, dan stopt de huidige herhaling onmiddellijk en begint meteen de volgende herhaling. De inhoud van de lus dat na het `continue`-statement komt, wordt overgeslagen.

We willen een programma maken om het gemiddelde van 10 getallen willen berekenen. Deze getallen moeten aan de gebruiker gevraagd worden via de functie `input()`. Het is heel belangerijk dat er exact 10 getallen gegeven worden. Dus als de gebruiker per ongeluk op enter duwt zonder een getal te hebben gegeven, dan telt deze input niet. Laten we dit eens vertalen naar code:
Zonder het groene gearceerde blok heb je een `while`-lus dat het gemiddelde van exact 10 waardes opvraagd van de gebruiker. Met het groene gearceerde blok worden ongeldige waardes overgeslagen.

```python
# we hebben twee variabelen nodig
som = 0                      # de tussentijdse som
aantal = 0                   # het huidige aantal gekregen getallen

while aantal < 10:           # de lus moet doorgaan tot 10 getallen

    # we gebruiken input() om een nieuw getal te vragen
    nieuweInput = input("Geef een getal: ")

    # als de gebruiker ongeldige input geeft
    # dan mag deze stap niet tellen
    if nieuweInput == "":
        # de gebruiker heeft per ongeluk op enter geduwd?
        # we moeten deze stap overslaan!
        continue

    som += int(nieuweInput) # we zetten de tekst om in een getallen
                            # die we dan bij som optellen
    aantal += 1

# tot slot bepalen we het gemiddelde en tonen we deze aan de gebruiker
gemiddelde = som/aantal
print(f"Het gemiddelde van de gegeven getallen: {gemiddelde}")
```

1.2.1 In het kort

- `while voorwaarde:`
  Een `while`-lus gebruik je om een blok code te blijven herhalen. De uitvoering stopt pas tot de voorwaarde evalueert tot `False`.

- `break`
  Het `break`-statement stopt een lus meteen. En dat meteen op de plaats waar het `break`-statement zich bevindt. Een nieuwe herhaling wordt niet meer uitgevoerd.

- `continue`
  De resterende uitvoering van de blok code wordt overgeslagen vanaf waar het `continue`-statement wordt uitgevoerd. In de plaats wordt meteen een nieuwe herhaling opgestart, beginnend met een volgende evaluatie van de voorwaarde.