Lesbrief 7 While loops met een teller en debuggen

In de vorige les hebben we het over while loops gehad, maar wat nou als je na een bepaalde tijd wilt stoppen met de loop? Bijvoorbeeld na tien keer dezelfde vraag gesteld te hebben dat het programma stopt. Dit kan door booleaanse vergelijkingen met de while mee te geven. Als de vergelijking dan False is, stopt het programma. Het gaat als volgt:

We maken een eerst een counter aan met een bepaalde begin waarde. Vervolgens updaten we deze steeds en als deze tien wordt stopt het
programma.

```
counter = 0
while counter < 10:
  woord = input("Type een woord")
  if woord == "quit":
    break

  lengte = len(woord)
  print ("De lengte van jouw woord is " + str(lengte))
  counter += 1
  print counter

print ("EINDE")
```

While met een counter

Je kan hier zien dat er hier twee verschillende manieren gebruikt worden om uit de while loop te komen. De eerste is als het woordje `"quit"` getypt wordt, want dan wordt er vervolgens een `break` gebruikt die ervoor zorgt dat we uit de loop springen. De tweede manier is dus de booleaanse vergelijking na de `while`. Zolang deze vergelijking `True` geeft, blijft de loop in werking. Maar op het moment dat deze booleaanse vergelijking een waarde van `False` heeft, wordt de loop gestopt en gaan we verder naar de rest van het programma (in dit geval dus: `print ("EINDE")`).

Opdracht 1

Schrijf een programma waar de gebruiker steeds om een getal wordt gevraagd. Tel dit getal dan op bij het voorgaande getal. Als deze som van alle voorgaande getallen uiteindelijk groter is dan 200, stopt het programma en wordt uit uiteindelijke getal geprint.

```
>>> Hallo, wil je mij een getal geven?
>>> 27
>>> Hallo, wil je mij een getal geven?
>>> 150
>>> Hallo, wil je mij een getal geven?
>>> 63
>>> Het totaal is: 240
```

Opdracht 2

Schrijf een programma waarbij de gebruiker vijf pogingen krijgt om een getal onder de tien moet raden. Als het getal geraden is stopt het programma en krijgt de gebruiker een vrolijk berichtje dat hij/zij heeft gewonnen! Als het niet geraden wordt stopt het programma dus na vijf keer.

```
>>> Geef mij een getal onder de tien a.u.b.
>>> 5
>>> Helaas!
>>> Geef mij een getal onder de tien a.u.b.
>>> 6
>>> Helaas!
>>> Geef mij een getal onder de tien a.u.b.
>>> 7
>>> Helaas!
>>> Geef mij een getal onder de tien a.u.b.
>>> 8
>>> Helaas!
>>> Geef mij een getal onder de tien a.u.b.
>>> 9
>>> Helaas! Je hebt het niet geraden binnen vijf pogingen :(
```

Opdracht 3

Vraag de gebruiker om het langste woord dat ze kunnen bedenken, en print vervolgens elke letter daarvan apart uit. (denk aan slicing)

Opdracht 4
Schrijf een programma waarin je om de gebruiker zijn email adres vraagt, en vervolgens alles voor het “@ ” teken print.

Opdracht 5
Schrijf een programma waarin je de gebruiker om een zin vraagt, en print vervolgens elk woord apart uit.

Logische operatoren and, or, not

Zoals je nu inmiddels heel goed weet kan je booleaanse expressies gebruiken bij verscheidene dingen! Maar wat nou als je meerdere voorwaarden in combinatie met elkaar wilt gebruiken?!? Nou dit kan dus heel makkelijk! Dit kan namelijk door middel van het gebruik van and, or en not operatoren. Het gaat als volgt:

Hier schrijven we een programma waarin we checken of een gegeven getal buiten twee waardes past. De "and" operator kijkt of de twee booleaanse expressies allebei true geven en dan pas is het geheel ook true.

```
keuze1a = input("wilt u appels, peren of manderijnen?")
keuze1b = input("Nogmaals, wilt u appels, peren of manderijnen?")

if keuze1a == "appels" and keuze1b == "peren":
  print ("U heeft eerst appels en toen peren gekozen!")
else:
   print ("U heeft een andere combinatie gekozen dan eerst appels en
toen peren.")
```

We kunnen ook een "or" operator gebruiken, hier wordt er gechecked of een van de twee booleaanse expressies waar zijn. Als een van de twee true geeft, is het geheel ook true. Zo kan je hier goed het verschil zien tussen "and" en "or".

```
keuze2 = input("wilt u appels, peren of manderijnen?")

if keuze2 == "appels" or keuze2 == "peren":
  print ("U heeft appels of peren gekozen")
else:
  print ("U heeft een manderijn gekozen!")
```

Dan heb je als laatst de "not" operator. Hier worden de verwachtingen van de uitkomst van de booleaanse expressies omgekeerd. Waar true wordt verwacht, wordt nu false verwacht.

```
keuze3 = input("wilt u appels, peren of manderijnen?")

if not(keuze3 == "appels" or keuze3 == "peren"):
   print ("U heeft geen appels of peren gekozen, dus veel plezier met uw manderijnen!")
else:
  print ("U heeft appels of peren gekozen")
```

and, or, not voorbeelden

Opdracht 6
Schrijf een functie die kijkt of een waarde tussen de 3 en de 8 zit. Als de waarde hier tussen zit, meld dit dan door middel van een print.

Opdracht 7
Schrijf een functie die kijkt of een waarde lager is dan 20 of hoger is dan 100. Meld weer je bevinding van de waarde d.m.v. een print.

***** Opdracht 8 (ster opdrachten moeten sowieso ingeleverd worden!)**
Stel je krijgt de keuze om naar Walibi of naar slagharen te gaan. Als je naar Walibi wilt, dan krijg je korting als je met 10 of meer mensen gaat. En als je naar slagharen gaat, dan krijg je al korting als tussen de 5 of de 10 mensen mee neemt.

Schrijf een programma die vraag of de gebruiker naar Walibi of slagharen wilt. En vervolgens moet er gevraagd worden met hoeveel mensen de gebruiker erheen wilt. Geef dan aan of de gebruiker korting krijgt of niet bij een bepaalde hoeveelheid mensen.

Opdracht 9
Schrijf een programma dat een zin vraagt aan de gebruiker, en vervolgens kijkt of het woord “de” niet in de zin voorkomt. Meld dit dan vervolgens.

Debugging

Python tutor

Om te kijken hoe een computer werkt, kunnen we ook per regel kijken in python tutor. Dit is een site waarin je code per regel gevisualiseerd wordt. Hier kan je dus zien wat er in het “hoofd” van de computer om gaat. Voortaan als je even niet snapt wat er gebeurt, kun je dus ook altijd de code plakken in python tutor. De link is:

http://www.pythontutor.com/live.html#mode=edit

Errors

Tijdens het schrijven van je programma’s ga je altijd wel wat fouten maken waardoor je programma niet meer loopt. Naarmate je grotere en moeilijkere programma’s gaat schrijven, zal je ook veel meer tijd kwijt zijn aan het oplossen van deze fouten.

Als de computer je programma doorloopt en een fout hierin vind dan geeft hij een foutmelding, oftewel een error. Het is aan jou dan om de bron van deze error te vinden en het probleem op te lossen. Dit heet debugging. Debuggen kan erg lastig zijn en lang duren, maar als je hier eenmaal wat meer ervaring mee hebt zal het vanzelf makkelijker gaan en je zal het zelfs fijn vinden dat er gezegd wordt waar je moet kijken!

In een vorige les hadden jullie al een kleine introductie gehad over variabelen en de errors die je kan krijgen als je deze verkeerd had toegewezen of gebruikt. Zo een error zag er ongeveer zo uit:

```
Traceback (most recent call last):
  File "voorbeeld_debugging.py", line 2, in <module>
    print (x+y)
NameError: name 'y' is not defined
```

Deze error komt voort uit het volgende simpele programmaatje:

```
x = 3
print (x+y)
```

Hier zie je dus dat de variabele x word verwezen naar drie, maar y is hier vergeten om te initiëren voordat deze aan geroepen wordt in de print functie. De error verwijst hier naar, 'y' is not defined, dus 'y' is niet gedefinieerd of verwezen naar iets. Vervolgens kijk je naar welke regel het programma vast loopt, regel 2 dus, en dan kijk je wat daar vóór (dus voor regel 2) in het programma mis gaat. In dit geval moet de code veranderd worden naar het volgende:

```
x = 3
y = 8
print (x+y)
```

En zo zal het wel werken! Maar om te leren debuggen moet je het voornamelijk heel veel zelf doen. Dus daarom gaan we nu door een paar foutieve codes heen werken via trinket links die je zelf moet debuggen.

```
a = 10
b = 15
c = a + d
d = a - b
a = d - c
a = a - 3
print (a)
```

Opdracht 10
Maak het komende programma weer werkende!

Debugging 1

Of deze:

```
import math

getal_1 = input("hallo, geef mij a.u.b. een getal!")
getal_2 = int(input("Heyhoooooi, nog een getal graag! :D"))

getal_1 = getal_1 + getal_2
getal_2 + 3 = getal_2
```

```
print (getal_1, getal_2)

getal_3 = getal_1 / getal_2

print (floor(getal_3), ceil(getal_3), getal_3)
```

Opdracht 11
Maak het komende programma weer werkende!

Debugging 2

De vorige codes bevatten nog maar een klein programma, maar hoe groter en ingewikkelder de programma's worden hoe moeilijker het wordt om te vinden waar het probleem zit. Het volgende probleem is een best lastig programma, dat ook wat groter is dan de vorige. Neem je tijd en kijk goed naar de error messages!

```
def fibonacci(startgetal, x_aantal_cijfers):
  nieuw_getal = int(startgetal)
  oud_getal = int(startgetal)
  print (oudgetal)
  print (nieuw_getal)
  for i in range(int(x_aantal_cijfers) - 1):
    oud_getal, nieuw_getal = nieuw_getal, nieuw_getal + oud_getal
    print (nieuw_getal)

start = input("Hey, mag ik een startgetal?)
x_aantal_cijfers = input("Hey! Mag ik nog een getal voor het aantal
cijfers dat we gaan berekenen?!?")

fibonacci(start, x)
```

***** Opdracht 12**
Maak het komende programma weer werkende!

Fibonacci debugging