Lesbrief 5 For loops, while loops en strings

Nog een keer loops, maar dan *interactieve while loops*

In één van de vorige lesbrieven heb je geleerd hoe loops werken. Als je wilt dat iets tien keer wordt herhaald, dan gebruik je de for-loop. Hier een voorbeeld:

```
for i in range(10):
      print ("Dit is herhaling " + str(i))
```

In dit geval wordt de regel "Dit is herhaling" tien keer herhaald. En na de zin *Dit is een herhaling* volgt de inhoud van de loopcounter `i`. De uitvoer van dit programma kun je zien op: Herhaling: for loop

Opdracht 1
Schrijf een programma dat vraagt of iemand wel of niet naar de dierentuin wil, en als het antwoord ja is, vraag dan met hoeveel mensen zij erheen willen. Print dan voor ieder persoon waarvan ze willen die mee gaat een keer “veel plezier!”.

***** Opdracht 2 (ster opdrachten moet je sowieso inleveren!)**
Breid het voorgaande uit. Als er op de eerste vraag een nee gegeven wordt, vraag dan of ze anders misschien naar de film willen. Zo ja, met hoeveel personen (en weer zo vaak printen), en zo nee print dan een gepast antwoord.

In de voorgaande herhalingen weet je precies hoe vaak je het wilt herhalen. Maar wat als je *niet* van te voren weet hoe vaak je iets wilt herhalen? Dan gebruik je een andere soort loop. De `while`-loop. Zie hier het onderstaande programma (of klik hier: Voorbeeld while loop):

‘while’ betekent ‘zolang’
Je herhaalt dus iets net zo lang als het waar is. Als het niet meer waar is stop je.

```
while True:
  woord = input("Typ een woord")
  lengte = len(woord)
  print ("De lengte van jouw woord is " + str(lengte))

print ("EINDE")
```

Als je dit programma uitvoert, dan zal hij nooit stoppen. Hij zal continu vragen om een woord in te typen en zodra je een woord intypt geeft hij de lengte van het ingetypte woord. Dit betekent dat het woordje EINDE nooit geprint zal worden. Daar kunnen we echter wel een verandering in aanbrengen. We kunnen namelijk dit zeggen: Als iemand het woordje quit intypt, dan stoppen we met de loop. Dit

stoppen doen we met het commando `break`. Dit ziet er als volgt uit: (While loop met een break):

```
while True:
  woord = input("Typ een woord")
  if woord == "quit":
    break

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

print ("EINDE")
```

Dit worden ook wel *interactieve while loops* genoemd. Het aantal keren dat de loop herhaalt is dan niet van te voren bepaald, maar hangt af van de invoer van de gebruiker!

Opdracht 3

Maak een interactieve while loop waarin een gebruiker net zo lang woorden kan intypen, totdat hij het woordje **stop** ingeeft. Zodra hij dat doet, gaat het programma uit de loop en eindigt hij. Een verloop van het programma zou er als volgt uit kunnen zien:

```
>>> Voer een woord in: hoi
>>> Voer een woord in: doei
>>> Voer een woord in: informatica
>>> Voer een woord in: boe
>>> Voer een woord in: stop
>>> Je wilt stoppen? Dat kan! Tot de volgende keer!
```

***** Opdracht 4**

Schrijf een programma dat blijft vragen om een getal, en deze steeds bij elkaar op blijft tellen tot er stop wordt gezegd. En print vervolgens het totaal van alle getallen. Dat zal er zo uit zien:

```
>>> Geef een getal! 34
>>> je zit nu op 34
>>> Geef een getal! 23425
>>> je zit nu op 23459
>>> Geef een getal! 234
>>> je zit nu op 23693
>>> Geef een getal! 6445
>>> je zit nu op 30138
>>> Geef een getal! 0695
>>> je zit nu op 30833
>>> Geef een getal! quit
>>> je totaal is 30833
>>> EINDE
```

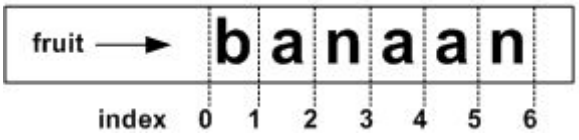
Hint: Je moet hierbij een variabele aan maken waar je steeds de input bij op telt om vervolgens te printen!

Tekstwaarden zijn rijen met letters ...

Een stukje tekst (string) in Python is eigenlijk een rij met letters. En zo kun je een variabele met tekst ook benaderen. Zie het volgende voorbeeld:

```
fruit = "banaan"
print (fruit[3])
```

Door achter de variabele `fruit` blokhaken (daarmee bedoel ik de tekens '[' en ']') te plaatsen en daartussen een getal te zetten, geef je eigenlijk aan dat je de letter op positie drie wilt uitprinten. Dus in dit geval zeg ik: geef mij de letter op positie drie in het woord 'banaan'. Maar denk erom dat python begint te tellen bij 0! Kortom, je krijgt niet de *n* te zien, maar de *a*! Zie ook het onderstaande plaatje:



Opdracht 5
Maak een programma dat de gebruiker steeds vraagt(in een while loop) om een woord in te voeren. Als het woord eindigt op een n, dan zegt hij "Je woord eindigt op een n" en stopt hij met de loop.

TIP: Omdat je niet weet hoe lang het woord is, dat de gebruiker intypt, kun je ook niet zomaar naar de zoveelste letter kijken. Maar wellicht kun je wel dankbaar gebruik maken van de `len()` functie!

Loopen door een woord

Het volgende programma telt het aantal keren dat de letter ‘a’ voorkomt in een woord ("a" teller):

```
woord = 'banaan'
teller = 0
for letter in woord:
      if letter == 'a':
            teller = teller+ 1
            print (teller)
print (teller)
```

Hierboven staat een variant van een loop die je als het goed is nog niet eerder hebt gezien. `for letter in woord` is een loop die in dit geval *zes* keer wordt herhaald. Waarom? Omdat het woordje banaan uit zes letters bestaat! En elke keer bevat de variabele `letter` de volgende letter in het woord. De eerste keer bevat het de letter `b`, de tweede keer de letter `a`, de derde keer `n` enzovoorts.

Dit programma demonstreert ook het gebruik van een *teller*. De variabele teller is aan het begin ingesteld op 0 en vervolgens wordt hij iedere keer opgehoogd zodra een ‘a’ is gevonden. Als de loop afloopt, bevat teller het resultaat: namelijk het totaal aantal ‘a’-s.

Probeer te doorgronden hoe de loop werkt en de werking van de teller. Doe dat door gebruik te maken van de PythonTutor. Hier een link: hier

Opdracht 6
Pas het bovenstaande programma aan, zodat de gebruiker een woord mag ingeven. Vervolgens moet het programma het aantal klinkers (‘a’, ‘e’, ‘i’, ‘o’ , ‘u’) tellen in dat woord.

For-loops in for-loops

Net als dat if-opdrachting in if-opdrachten gezet kunnen worden, kan dit ook met for-loops. Dit is heel handig als je bijvoorbeeld tien keer door een woord wilt loopen, kan je dat bijvoorbeeld doen als volgt (for-loop in een for-loop):

```
woord = 'banaan'
for i in range(5):
  for letter in woord:
      print (letter)
```

Zoals je kan zien wordt er hier 5 keer door het woord 'banaan' heen geloopt. En elke keer word er een print actie uitgevoerd per letter in dat woord. Daarom staat elke letter op een nieuwe regel. Als je bijvoorbeeld een ander woord zou nemen in plaats van "for i in range(5):" zou er net zo vaak banaan geschreven worden, als er letters in dat woord zou zitten (geneste for-loop met twee woorden).

```
woord = 'banaan'
for i in 'koe':
  for letter in woord:
      print (letter)
```

Hier zie je dus dat het woord banaan drie keer geprint wordt, omdat het woord 'koe' bestaat uit drie letters. Bij het volgende voorbeeld wordt dat nog iets duidelijker hoe het werkt (per woord, per letter in de for-loop):

```
woord = 'banaan'
for i in 'koe':
  for letter in woord:
    print (i)
    print (letter)
```

Hier zie je dat er eerst een 'k' geprint wordt, vervolgens een 'b', daarna weer een 'k', dan een 'a' etc. Zo zie je dat er per letter in koe, steeds door het woord banaan gegaan wordt.

Opdracht 7
Schrijf een programma dat vier keer, elke letter van het woord “lolly” print.

Opdracht 8
Schrijf een programma dat afwisselend elke letter van twee verschillende woorden print, dus:

koe + man wordt:

```
>>> k
>>> m
>>> o
>>> a
>>> e
>>> n
```

De in-operator: checken of een letter in een woord zit

Je kunt in Python ook met de `in` operator kijken of een letter in een woord voorkomt:

```
if "i" in woord:
        print ("i zit in het woord: " + woord)
if "ik ben" in zin:
        print ("ik ben zit in de zin: " + zin)
if letter in paragraaf:
        print ("letter zit in de paragraaf")
```

In alle drie de gevallen is de `in`-constructie een *vergelijking* die waar of onwaar is. *Daarom* mag je hem achter een if zetten.

Nog een paar stringfuncties

Er zijn een aantal handige functies voor tekstwaarden (strings). De eerste waar je kennis mee hebt gemaakt is `len()`. Maar er zijn er meer! Wat te denken van `replace()`?

```
woord = "doei doei!"
woord = woord.replace ("d", "f")
print (woord)
```

De functie `replace` zet je achter de variabele. Vervolgens geef je het twee parameters mee: de letter(s) die vervangen moeten worden en de letter(s) waarmee ze vervangen moeten worden. Hierboven worden dus de d's vervangen door de f'jes. Denk er wel om dat je het resultaat weer in een (andere) variabele moet zetten. Dus de `woord =` aan het begin is noodzakelijk!

```
woord = "banaan"
nieuw_woord = woord.upper()
print (nieuw_woord)
```

De `upper()` functie zet het hele woord in hoofdletters. Je hebt ook de functie `lower()`. Raad eens wat dat voor effect heeft ;)

***** Opdracht 9**
Als jongeren iets op Whatsapp schrijven, dan korten ze vaak woorden af. Zo staat ajb voor alsjeblieft, zkr voor zeker en drm voor daarom. Dit noem je ook wel chattaal.

Maak een programma dat de gebruiker steeds vraagt om een zin in te voeren. Vervolgens worden de afkortingen zkr, drm en ajb omgezet naar de bijbehorende woorden.

Voorbeeld:
Als je deze zin invoert: Heb je dat zkr gedaan? Drm zeg ik het toch tegen je! Vertel het mij ajb.
Dan komt dit eruit: Heb je dat zeker gedaan? Daarom zeg ik het toch tegen je! Vertel het mij alsjeblieft!

Opdracht 10
Henk schrijft het volgende programma:

```
antwoord = input("Wil je een boterham met kaas?")
if antwoord == "Ja":
       print ("Hier heb je het!")
```

Het programma werkt echter niet als de gebruiker het woordje *ja* met kleine letters intypt, en ook niet als de gebruiker JA met hoofdletters intypt. Hoe kun je nou dit probleem oplossen zonder dat je drie if-opdrachten maakt? Maak slim gebruik van de upper() functie!

[Uitdaging] Opdracht 11

Ooit wel eens van een palindroom gehoord? Het is een symmetrisch woord. Dat wil zeggen, het is een woord dat als je het omkeert weer hetzelfde woord oplevert. Je gaat een programma schrijven waarmee je kunt testen of een woord een palindroom is of niet. Voorbeelden zijn parterretrap, lepel en 012210.

Schrijf een programma dat vraagt naar een woord (eventueel ook cijfers) en dat daarna het woord in 'omgekeerde vorm' toont en vermeldt of het woord wel of geen palindroom is.