5.2 While - Loop

While-Loop

De while loop is eigenlijk eenvoudiger uit te leggen. Hij kent één vergelijking, de lus-voorwaarde en zolang die true is blijft de loop doorlopen.

```
$i=0;
while($i<10){
	$i++;
    ...
}
```

Met deze while-loop doe je eigenlijk hetzelfde als in de for loop: `for($i=0; $i<10; $i++)` Je ziet dat de startwaarde op regel 1 staat en de lus-teller op regel 3.

Deze loop is iets 'gevaarlijker', omdat de drie fases van de loop niet per sé bij elkaar staan. Dat is 'gevaarlijk', omdat je dan eerder het overzicht verliest en misschien het ophogen met de lus-teller (in het voorbeeld regel 3) vergeet. Ook kan het minder overzichtelijk zijn.

Bij een while-loop moet je net als bij een for-loop altijd nadenken over de 3 fasen:

1. Wat is de start-waarde; hoe begint de loop (in het voorbeeld $i=0)?
2. Wat is de lus-teller?
3. Wat is de lus-voorwaarde??

Stel je hebt een functie die een row uit de database haalt, deze functie heet getRow() en deze functie returned de row uit de database. Als er geen rows meer zijn dat wordt er 0 teruggegeven. Je kunt nu een while-loop gebruiken om alle regels af te drukken:

```
<?php
$dezeRegel = getRow();

while($dezeRegel<>0){
	echo $dezeRegel;
    $dezeRegel=getRow();
}
?>
```

Het is gebruikelijk om dit op deze manier uit te voeren, maar het kan ook in een for-loop. Welke methode vind jij beter leesbaar?

```
<?php
for( $dezeRegel = getRow(); $dezeRegel<>0; $dezeRegel=getRow();){
echo $dezeRegel;
}
?>
```

In sommige gevallen wil je eindeloze loop maken. Een while loop kun je eenvoudig eindeloos maken: `while(true){...`

Do-While

De do-while loop is een variant op de while-loop. In deze loop staat de lus-voorwaarde aan het eind in plaats van in het begin zoals bij de while-loop en de for-loop.

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

Wat denk je dat deze code doet? Probeer deze code. Als je deze code uitvoert, dan zie je dat de loop 1x uitgevoerd wordt en dat terwijl $i nooit groter dan 0 is (en dus de vergelijking nooit true oplevert). Een Do-While loop wordt dus altijd minimaal 1x uitgevoerd. Dat geldt niet voor de 'gewone' while-loop of voor de for-loop.

Stroomdiagrammen

Hieronder zie je twee stroom-diagrammen. Welke hoort bij de while en welke bij de do-while? Welke zou het beste passen bij een for-loop?

Image result for do while loop php	Image result for while loop in php

Break

Met een break statement spring je uit de huidige loop. Je gaat direct naar het eind en begint met het statement dat vlak na de } staat aan het einde van de loop. De lus-code wordt dus niet verder uitgevoerd.

Continue

Met het continue statement stop je met de huidige iteratie en ga je naar het begin van de loop om de volgende iteratie te beginnen. De lus-code wordt dus weer vanaf het begin uitgevoerd. Bij een for-loop wordt dat we de lus-teller bijgewerkt en wordt gekeken of er nog aan de lus-voorwaarde wordt voldaan.

Opdracht 1

De volgende code moet alle getallen van 1 t/m 25 afdrukken, maar de programmeur heeft één regel code vergeten. Denk aan de lus-voorwaarde, lus-startwaarde en lus-teller; staan die er allemaal in? Kun jij een regel toevoegen, zodat de getallen 1 t/m 25 worden afgedrukt?

```
<?php
$count=0;
while (true) {
  echo $count . "<br>";
  if ( $count == 25 ) {
    break;
  }
}
?>
```

Opdracht 2a

Met de volgende code gooi je met twee dobbelstenen ($dice1 en $dice2). Beide dobbelstenen krijgen een willekeurige waarde van 1 t/m 6, net als een dobbelsteen.

```
$dice1=rand(1,6);
$dice2=rand(1,6);
echo $dice1. "-" .$dice2;
echo "<br>";
```

Zet de code in een loop, zodat er 10x met beide dobbelstenen wordt gegooid.

De output ziet er bijvoorbeeld als volgt uit:

```
1-2
5-6
4-2
```

```
5-3
4-3
3-1
5-6
6-1
3-2
1-3
```

Opdracht 2b

Verander de loop nu, zodat de code net zolang blijft 'gooien' totdat er twee maal 6 is gegooid. Op de laatste regel moet dus 6-6 komen te staan.

Zet een getal voor de worp, zodat je eenvoudig kan zien hoe vaak de code de dobbelstenen heeft 'gegooid'. Zet ook achter elke worp de som van de twee dobbelstenen. De output ziet er dan bijvoorbeeld als volgt uit. De laatste regel moet dus altijd twee zessen laten zien.

```
worp 1: 1-2 totaal 3
worp 2: 5-6 totaal 11
worp 3: 4-2 totaal 6
worp 4: 5-3 totaal 8
worp 5: 4-3 totaal 7
worp 6 :3-1 totaal 4
worp 7: 6-6 totaal 12
```

Opdracht 2c

Pas de code van opdracht 2b aan, zodat je met drie dobbelstenen gooit. De output iet er bijvoorbeeld als volgt uit:

```
worp 1: 3-4-3
worp 2: 1-1-6
worp 3: 2-2-5
worp 4: 3-2-3
...
```

'Gooi' nu net zolang met de drie dobbelstenen totdat je drie maal een 6 heb gegooid.

--

Revision #11
Created 2020-02-04 09:22:01 UTC by Max
Updated 2020-04-17 08:05:39 UTC by Max