

For loop

Tijdens de eerste les zagen we de *while*-lus om dingen te herhalen. Een andere manier om code te herhalen is de *for*-lus. De *for*-lus kan je gebruiken indien je op voorhand weet hoe vaak je de lus moet uitvoeren.

Volgende code drukt 3 maal de zin "I love Python" af:

In [1]:

```
for i in range(3):  
    print("I love Python")
```

```
I love Python  
I love Python  
I love Python
```

In de 'for i in range(3):' is *i* een variabele die over "range(3)" gaat itereren; dit wil zeggen: de getallen 0, 1 en 2. *range(n)* is dus de reeks getallen van 0 tot en met *n-1*. Volgend stukje code laat zien dat *i* inderdaad de opeenvolgende waarden van de range aanneemt:

In [2]:

```
for i in range(3):  
    print("start van het stukje herhaalde code")  
    print("i =",i)  
    print("einde van het stukje herhaalde code")
```

```
start van het stukje herhaalde code  
i = 0  
einde van het stukje herhaalde code  
start van het stukje herhaalde code  
i = 1  
einde van het stukje herhaalde code  
start van het stukje herhaalde code  
i = 2  
einde van het stukje herhaalde code
```

De betekenis van bovenstaande code is dus: voer het ingesprongen stukje uit voor achtereenvolgens *i* gelijk aan 0, 1, 2. Met andere woorden, het is equivalent aan:

In [3]:

```
i=0
print("start van het stukje herhaalde code")
print("i =",i)
print("einde van het stukje herhaalde code")
i=1
print("start van het stukje herhaalde code")
print("i =",i)
print("einde van het stukje herhaalde code")
i=2
print("start van het stukje herhaalde code")
print("i =",i)
print("einde van het stukje herhaalde code")
```

```
start van het stukje herhaalde code
i = 0
einde van het stukje herhaalde code
start van het stukje herhaalde code
i = 1
einde van het stukje herhaalde code
start van het stukje herhaalde code
i = 2
einde van het stukje herhaalde code
```

Met de variabele *i* kan je nu ook gaan rekenen; we kunnen bijvoorbeeld de maaltafel van 3 afdrukken:

In [4]:

```
for i in range(11):
    print(i,"maal 3 is",3*i)
```

```
0 maal 3 is 0
1 maal 3 is 3
2 maal 3 is 6
3 maal 3 is 9
4 maal 3 is 12
5 maal 3 is 15
6 maal 3 is 18
7 maal 3 is 21
8 maal 3 is 24
9 maal 3 is 27
10 maal 3 is 30
```

Hier is de betekenis dus gelijk aan:

In [5]:

```
i=0
print(i,"maal 3 is",3*i)
i=1
print(i,"maal 3 is",3*i)
i=2
print(i,"maal 3 is",3*i)
i=3
print(i,"maal 3 is",3*i)
i=4
print(i,"maal 3 is",3*i)
i=5
print(i,"maal 3 is",3*i)
i=6
print(i,"maal 3 is",3*i)
i=7
print(i,"maal 3 is",3*i)
i=8
print(i,"maal 3 is",3*i)
i=9
print(i,"maal 3 is",3*i)
i=10
print(i,"maal 3 is",3*i)
```

```
0 maal 3 is 0
1 maal 3 is 3
2 maal 3 is 6
3 maal 3 is 9
4 maal 3 is 12
5 maal 3 is 15
6 maal 3 is 18
7 maal 3 is 21
8 maal 3 is 24
9 maal 3 is 27
10 maal 3 is 30
```

We kunnen de maaltafel ook omgekeerd afdrukken:

In [6]:

```
for i in range(11):
    j=10-i
    print(j,"maal 3 is",3*j)
```

```
10 maal 3 is 30
9 maal 3 is 27
8 maal 3 is 24
7 maal 3 is 21
6 maal 3 is 18
5 maal 3 is 15
4 maal 3 is 12
3 maal 3 is 9
2 maal 3 is 6
1 maal 3 is 3
0 maal 3 is 0
```

Naast de vorm `range(n)` die een reeks getallen van 0 tot en met $n-1$ genereert, bestaan ook de volgende vormen:

- `range(start,einde)` : generator voor de reeks getallen `start`, `start+1`, ..., `einde-1`
- `range(start,einde,stap)` : generator voor de reeks getallen `start`, `start+stap`, `start+2stap`, ... en deze reeks gaat door zolang je niet voorbij `einde` gaat. Merk op dat `stap`* ook negatief mag zijn.

Enkele voorbeelden:

In [7]:

```
# getallen 1 tot en met 5:  
for i in range(1,6):  
    print(i)
```

```
1  
2  
3  
4  
5
```

In [8]:

```
# veelvouden van 2, beginnend van 4, tot 12 (niet inbegrepen):  
for i in range(4,12,2):  
    print(i)
```

```
4  
6  
8  
10
```

In [9]:

```
# Countdown van 10 tot 1:  
for i in range(10,0,-1):  
    print(i)
```

```
10  
9  
8  
7  
6  
5  
4  
3  
2  
1
```

In [10]:

```
# Countdown van 100 tot 10, maar in stappen van 10:  
for i in range(100,0,-10):  
    print(i)
```

```
100  
90  
80  
70  
60  
50  
40  
30  
20  
10
```

Verschil while en for loop

In elke situatie waarin je de *for*-loop kan gebruiken, kan je ook de *while*-loop gebruiken; je moet dan wel zelf de index-variabele initialiseren, verhogen en de stop conditie coderen. Volgende stukken code zijn bijvoorbeeld equivalent:

In [11]:

```
for i in range(5):  
    print(i)  
  
i=0  
while i<5:  
    print(i)  
    i=i+1
```

```
0  
1  
2  
3  
4  
0  
1  
2  
3  
4
```

Bijvoorbeeld, onze maaltafel van 3 kan ook als volgt gemaakt worden:

In [12]:

```
i=0
while i<11:
    print(i,"maal 3 is",3*i)
    i=i+1
```

```
0 maal 3 is 0
1 maal 3 is 3
2 maal 3 is 6
3 maal 3 is 9
4 maal 3 is 12
5 maal 3 is 15
6 maal 3 is 18
7 maal 3 is 21
8 maal 3 is 24
9 maal 3 is 27
10 maal 3 is 30
```

Let op! Vaak voorkomende fouten:

- vergeten de variabele `i` te initialiseren (`i=0` vergeten) met als resultaat dat Python de conditie `i<11` niet kan evalueren
- vergeten de teller te verhogen (`i=i+1` vergeten) met als resultaat dat de `while`-lus oneindig voortgaat.

We kunnen een `for`-lus dus steeds vervangen door een `while`-lus. Het omgekeerde is echter niet waar; er bestaan situaties waarin we geen `for`-lus kunnen gebruiken, namelijk als we bij aanvang van de lus niet kunnen zeggen hoe vaak de lus uitgevoerd zal moeten worden.

Een voorbeeld van zulk een situatie: *Lees getallen in van de gebruiker, totdat die 0 ingeeft:*

In [13]:

```
getal=-1 # eender welke waarde behalve 0 zodat de while minstens 1 maal wordt uigevoer
d
while getal != 0: # != betekent "niet gelijk aan"
    getal=int(input("Geef een getal (0 om af te sluiten) :"))
```

We kunnen nu bijvoorbeeld de som en het aantal getallen bijhouden om het gemiddelde te berekenen:

In [14]:

```
getal=-1 # eender welke waarde behalve 0 zodat de while minstens 1 maal wordt uigevoer
d
som=0
aantal=0
while getal != 0: # != betekent "niet gelijk aan"
    getal=int(input("Geef een getal (0 om af te sluiten) :"))
    som=som+getal
    aantal=aantal+1
print("Gemiddelde:",som/(aantal-1))
```

Gemiddelde: 2.0

(weet je waarom er `aantal-1` staat?)

We kunnen de *for*-lus echter *wel* gebruiken als we eerst aan de gebruiker vragen hoeveel getallen hij of zij wil inlezen:

In [15]:

```
aantal=int(input("Hoeveel getallen wil je inlezen?"))
som=0
for i in range(aantal):
    getal=int(input("Geef een getal (0 om af te sluiten) :"))
    som=som+getal
print("Gemiddelde:",som/aantal)
```

Gemiddelde: 1.5

(nu delen we wel door aantal i.p.v. aantal-1; wat is het verschil?)

Merk het verschil tussen de twee situaties: omdat we nu aan de gebruiker vragen hoeveel getallen die wil inlezen, weten we bij de start van de *for*-lus hoe veel iteraties die moet maken. Het is dus niet nodig dat de programmeur reeds weet hoe vaak de *for*-lus doorlopen moet worden, maar op het moment dat de lus start, moeten we het kunnen berekenen.