# Nuttige operators

Er zijn een paar ingebouwde functies en "operators" in Python die niet goed in een categorie passen, dus we zullen ze in deze lezing bespreken, laten we beginnen!

## range (bereik)

Met de bereikfunctie kun je snel *genereren* een lijst met gehele getallen, dit is erg handig, dus let goed op hoe je het gebruikt! Er zijn 3 parameters die u kunt doorgeven, een start, een stop en een stapgrootte. Laten we enkele voorbeelden bekijken:

```
range(0,11)
```

```
range(0, 11)
```

Merk op dat dit een **generator**-functie is, dus om er een lijst uit te halen, moeten we deze naar een lijst casten met **list()**. Wat is een generator? Het is een speciaal type functie dat informatie genereert en niet in het geheugen hoeft op te slaan. We hebben het nog niet gehad over functies of generatoren, dus houd dit voorlopig in je notities.

```
# Notice how 11 is not included, up to but not including 11, just like slice notation!
list(range(0,11))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
list(range(0,12))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
# Derde parameter is stapgrootte!
# stapgrootte betekent gewoon hoe groot je sprong/leap/stap je bent
# neem van het start-getal om naar het volgende getal te gaan.
```

```
list(range(0,11,2))
```

```
[0, 2, 4, 6, 8, 10]
```

```
list(range(0,101,10))
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

## opsommen (enumerate)

Enumerate is een zeer nuttige functie om te gebruiken met for-lussen. Laten we ons de volgende situatie voorstellen:

```
index_count = 0

for letter in 'abcde':
    print("At index {} the letter is {}".format(index_count,letter))
    index_count += 1
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```

Het bijhouden (track) van hoeveel loops je hebt doorlopen is zo gewoon, dat enumerate is gemaakt, zodat je je geen zorgen hoeft te maken over het creëren en updaten van deze index_count of loop_count variabele.

```
# Let op het uitpakken van de tuple!

for i,letter in enumerate('abcde'):
    print("At index {} the letter is {}".format(i,letter))
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```

### zip

Merk op dat het formaat enumerate daadwerkelijk terugkeert, laten we eens kijken door het te transformeren naar een list()

```
list(enumerate('abcde'))
```

```
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
```

Het was een lijst met tuples, wat betekent dat we tuple uitpakken konden gebruiken tijdens onze for-loop. Deze datastructuur is eigenlijk heel gewoon in Python , vooral als je met externe bibliotheken werkt. U kunt de functie **zip()** gebruiken om snel een lijst met tuples aan te maken door twee lijsten aan elkaar te "zippen".

```
mylist1 = [1,2,3,4,5]
mylist2 = ['a','b','c','d','e']

# Deze is ook een generator! We zullen dit later uitleggen, maar laten we het nu omzetten (
zip(mylist1,mylist2)
```

```
<zip at 0x1d205086f08>
```

```
list(zip(mylist1,mylist2))
```

```
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

Om de generator te gebruiken, kunnen we gewoon een for-lus gebruiken

```
for item1, item2 in zip(mylist1,mylist2):
    print('For this tuple, first item was {} and second item was {}'.format(item1,item2))
```

```
For this tuple, first item was 1 and second item was a
For this tuple, first item was 2 and second item was b
For this tuple, first item was 3 and second item was c
For this tuple, first item was 4 and second item was d
For this tuple, first item was 5 and second item was e
```

## in operator

We hebben het trefwoord (keyword) **in** al gezien tijdens de for-lus, maar we kunnen het ook gebruiken om snel te controleren of een object in een lijst staat.

```
'x' in ['x','y','z']
```

```
True
```

```
'x' in [1,2,3]
```

```
False
```

## not in

We kunnen **in** combineren met een **not** operator, om te controleren of een object of variabele niet in een lijst voorkomt/staat.

```
'x' not in ['x','y','z']
```

```
False
```

```
'x' not in [1,2,3]
```

```
True
```

## min and max

Controleer snel het minimum of maximum van een lijst met deze functies.

```
mylist = [10,20,30,40,100]
```

```
min(mylist)
```

```
10
```

```
max(mylist)
```

```
100
```

## random (willekeurig)

Python wordt geleverd met een ingebouwde willekeurige (random) bibliotheek. Er zijn veel functies in deze willekeurige bibliotheek, dus we zullen je voorlopig slechts twee handige functies laten zien.

```
from random import shuffle
```

```
# Hierdoor wordt de lijst "in-place" geschud (shuffled), wat betekent dat er niets wordt ge
shuffle(mylist)
```

```
mylist
```

```
[40, 10, 100, 30, 20]
```

```
from random import randint
```

```
# Retourneer willekeurig geheel getal in bereik [a, b], inclusief beide eindpunten.
randint(0,100)
```

```
25
```

```
# Retourneer willekeurig geheel getal in bereik [a, b], inclusief beide eindpunten.
randint(0,100)
```

```
91
```

## input (invoer)

```
input('Enter Something into this box: ')
```

```
Enter Something into this box: great job!
```

```
'great job!'
```