Python les-materialen

# Lijsten

Eerder, bij het bespreken van strings, introduceerden we het concept van een *sequences* in Python. Lijsten kunnen worden gezien als de meest algemene versie van een *reeks/sequences* in Python. In tegenstelling tot strings zijn ze veranderlijk, wat betekent dat de elementen in een lijst kunnen worden gewijzigd!

In deze sectie leren we over:

1.) Lijsten maken  
 2.) Lijsten indexeren en snijden (sliceren)  
 3.) Basislijstmethoden  
 4.) Lijsten nesten  
 5.) Inleiding tot lijstbegrippen (Comprehensations)

Lijsten zijn opgebouwd met vierkantehaakjes [] en komma’s die elk element in de lijst scheiden.

Laten we eens kijken hoe we lijsten kunnen maken!

# Maak een List genoemd my\_list aan  
my\_list = [1,2,3]

We hebeen net een List van integers gecreëerd maar de lijst kan inderdaad meerdere datatypen bevatten. Bijvoorbeeld.:

my\_list = ['A string',23,100.232,'o']

Net zoals strings, de len() functie gaat teruggeven het aantal van elementen in de list zitten.

len(my\_list)

4

### Indexering en Sliceren

We gebruiken indexering en slicering functies net zoals we vroeger met String gebruiken. Laten wee een nieuwe list creëren voor het herinneren hoe ze met String gebruikt worden:

my\_list = ['one','two','three',4,5]

# Grijp een element op index 0  
my\_list[0]

'one'

# Grijp index 1 en alle volgende indexen  
my\_list[1:]

['two', 'three', 4, 5]

# Grijp alles tot index 3  
my\_list[:3]

['one', 'two', 'three']

We kunnen ook + gebruiken om lijsten te concateneren. Het is vergelijkbaar met de samenvoegingen van Strings.

my\_list + ['new item']

['one', 'two', 'three', 4, 5, 'new item']

Opmerking: De instructie boven gaat niet de echte waarden van de originele-list veranderen

my\_list

['A string', 23, 100.232, 'o']

// TE DOEN: U zou de lijst opnieuw moeten toewijzen om de wijziging permanent te maken.

# Opnieuw toewijzen  
my\_list = my\_list + ['add new item permanently']

my\_list

['A string', 23, 100.232, 'o']

We kunnen ook de \* gebruiken voor een duplicatiemethode die lijkt op strings:

# Verdubbel de lijst  
my\_list \* 2

['A string', 23, 100.232, 'o', 'A string', 23, 100.232, 'o']

# Opnieuw verdubbeling niet permanent  
my\_list

['A string', 23, 100.232, 'o']

## Basislijstmethoden

Als je bekend bent met een andere programmeertaal, zou je parallellen kunnen trekken tussen arrays in een andere taal en lijsten in Python. Lijsten in Python zijn echter over het algemeen flexibeler dan arrays in andere talen om twee goede redenen: ze hebben geen vaste grootte (wat betekent dat we niet hoeven te specificeren hoe groot een lijst zal zijn), en ze hebben geen vaste typebeperking (zoals we hierboven hebben gezien).

Laten we verder gaan en enkele meer speciale methoden voor lijsten verkennen:

# Create a new list  
list1 = [1,2,3]

Gebruik de **append**-methode om een item permanent aan het einde van een lijst toe te voegen:

# Append  
list1.append('append me!')

# Show  
list1

[1, 2, 3, 'append me!']

Gebruik **pop** om een item uit de lijst te verwijderen. Pop haalt standaard de laatste index uit, maar u kunt ook specificeren welke index moet worden verwijderd. Laten we een voorbeeld bekijken:

# Pop off the item with 0 indexed   
list1.pop(0)

1

# Tonen  
list1

[2, 3, 'append me!']

# Wijs (assign) het popped element toe, houd rekenen mee dat de standaard popped index -1 popped\_item is  
popped\_item = list1.pop()

popped\_item

'append me!'

# Toon resterende lijst  
list1

[2, 3]

Er moet ook worden opgemerkt dat indexering van lijsten een fout retourneert als er geen element in die index is. Bijvoorbeeld:

list1[100]

---------------------------------------------------------------------------  
  
IndexError Traceback (most recent call last)  
  
<ipython-input-22-af6d2015fa1f> in <module>()  
----> 1 list1[100]  
  
  
IndexError: list index out of range

We kunnen de **sort** methode en de **reverse** methode gebruiken om ook uw lijsten te beïnvloeden:

new\_list = ['a','e','x','b','c']

#Tonen  
new\_list

['a', 'e', 'x', 'b', 'c']

# Gebruik omgekeerd om de volgorde om te keren (dit is permanent!)  
new\_list.reverse()

new\_list

['c', 'b', 'x', 'e', 'a']

# Gebruik sorteren om de lijst te sorteren (in dit geval alfabetische volgorde, maar voor nummers zal het oplopend (ASC) gaan)  
new\_list.sort()

new\_list

['a', 'b', 'c', 'e', 'x']

## Nestlijsten

Een geweldige eigenschap van Python-datastructuren is dat ze *nesting* ondersteunen. Dit betekent dat we datastructuren binnen datastructuren kunnen hebben. Bijvoorbeeld: Een lijst in een lijst.

Laten we eens kijken hoe dit werkt!

# Laten we drie lijsten maken  
lst\_1=[1,2,3]  
lst\_2=[4,5,6]  
lst\_3=[7,8,9]  
  
# Maak een lijst van lijsten om een matrix te vormen  
matrix = [lst\_1,lst\_2,lst\_3]

# Tonen  
matrix

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

We kunnen indexering opnieuw gebruiken om elementen te pakken, maar nu zijn er twee niveaus voor de index. De items in het matrixobject en dan de items in die lijst!

# Pak het eerste item in het matrixobject  
matrix[0]

[1, 2, 3]

# Pak het eerste item van het eerste item in het matrixobject  
matrix[0][0]

1

# Lijstbegrippen

Python heeft een geavanceerde functie genaamd lijstbegrippen (list comprehensions). Ze zorgen voor een snelle opbouw van lijsten. Om lijstbegrippen volledig te begrijpen, moeten we for-loops begrijpen. Maak je dus geen zorgen als je dit gedeelte niet helemaal begrijpt, en sla het gerust over, want we komen later op dit onderwerp terug.

Maar voor het geval je het nu wilt weten, hier zijn een paar voorbeelden!

# Bouw een lijstbegrip (comprehension) op door een for-lus te deconstrueren binnen een []  
first\_col = [row[0] for row in matrix]

first\_col

[1, 4, 7]

We hebben hier een lijstbegrip gebruikt om het eerste element van elke rij in het matrixobject te pakken. We zullen dit later in veel meer detail behandelen!

Voor meer geavanceerde methoden en functies van lijsten in Python, bekijk de sectie Geavanceerde lijsten verderop in deze cursus of van Python documentatie!