

Lijsten

Tot nogtoe zagen we reeds enkele datatypes:

- integer
- float
- string
- boolean

Deze datatypes hebben allen de eigenschap dat ze 1 waarde kunnen bevatten. Wat als we meerdere waarden tegelijk moeten bewaren? We kunnen dan verschillende variabelen gebruiken, maar dat is niet optimaal. Stel bijvoorbeeld dat we namen van studenten willen bewaren:

In [2]:

```
student1=""
student2=""
student3=""
student4=""

# inlezen studentennamen
print("geef een lege naam om te stoppen")
student1=input("Geef naam student 1 : ")
if student1!="":
    student2=input("Geef naam student 2 : ")
    if student2!="":
        student3=input("Geef naam student 3 : ")
        if student3!="":
            student4=input("Geef naam student 4 : ")

print("lijst van studenten : ")
if (student1!=""):
    print(student1)
    if (student2!=""):
        print(student2)
        if (student3!=""):
            print(student3)
            if (student4!=""):
                print(student4)
```

geef een lege naam om te stoppen

lijst van studenten :

Dit is uiteraard erg onhandig. Bovendien werkt dit enkel als we op voorhand weten *hoeveel* studenten we *maximaal* kunnen verwachten. Om dit soort situaties op te vangen hebben we een zogenaamd **container type** nodig. Een container type is een type waarin we meerdere waarden kunnen opslaan en dat kan groeien als we meer en meer elementen toevoegen.

Het *list* type is een geordende lijst van willekeurige grootte waarin we alle soorten waarden kunnen opslaan zonder beperking op het aantal waardes. Een lijst aanmaken doen we als volgt:

In [2]:

```
L=["a",1,2,"bcd",True,1]
print(L)
```

```
['a', 1, 2, 'bcd', True, 1]
```

Met deze regel maken we een lijst aan met 6 elementen. Merk op dat de elementen niet noodzakelijk van hetzelfde type moeten zijn, al zal dat in jouw programma's meestal wel het geval zijn. Ook kan een lijst meerdere keren dezelfde waarde bevatten. Bovendien is de volgorde van de elementen in een lijst van belang; de lijst `[1,2,3]` is verschillend van de lijsten `[3,2,1]` en `[1,1,2,3,3]`.

Een element toevoegen aan een lijst

Het is een goed idee om de studentenlijst op te slaan als een *list*. Om dat te doen starten we van een lege lijst (`[]`) waarin we één voor één de namen van de studenten toevoegen. Een element toevoegen aan een lijst doen we via de functie `L.append(item)`, waarbij `L` de lijst is waaraan we een item willen toevoegen; de lijst van daarnet kunnen we dus ook maken als volgt:

In [3]:

```
L=[]
L.append("a")
L.append(1)
L.append(2)
L.append("bcd")
L.append(True)
L.append(1)
print(L)
```

```
['a', 1, 2, 'bcd', True, 1]
```

We kunnen de lijst van studenten dus als volgt opbouwen:

In [4]:

```
L=[]
next=True
print("geef een lege naam om te stoppen")
while next:
    naam=input("Geef naam student : ")
    if naam=="":
        next=False
    else:
        L.append(naam)

print(L)
```

```
geef een lege naam om te stoppen
Geef naam student : tom
Geef naam student : toon
Geef naam student : daphne
Geef naam student : stephen
Geef naam student :
['tom', 'toon', 'daphne', 'stephen']
```

Individuele elementen in een lijst identificeren

Als we de namen één voor één willen afdrukken, elk op een nieuwe lijn, dan moeten we de elementen in de lijst kunnen overlopen. Dat kan makkelijk met de notatie $L[i]$ waarbij we het $i+1$ -ste element van de lijst ophalen. Bij een lijst met 5 elementen zijn de items in de lijst dus: $L[0]$, $L[1]$, $L[2]$, $L[3]$, $L[4]$. Een list afdrukken kan dus zo:

In [5]:

```
L2=[1,2,3,4,5]
for i in range(5):
    print("Op positie",i,"staat",L2[i])
```

```
Op positie 0 staat 1
Op positie 1 staat 2
Op positie 2 staat 3
Op positie 3 staat 4
Op positie 4 staat 5
```

De positie i in $L[i]$ noemen we de *index* van dat element. Merk op dat $range(n)$ gaat van 0 tot en met $n-1$, zijnde de geldige *indices* van elementen in L . Dat is niet toevallig.

De lengte van een lijst L kunnen we opvragen met $len(L)$. Alle namen afdrukken kunnen we met volgende code:

In [6]:

```
for i in range(len(L)):
    print(L[i])
```

```
tom
toon
daphne
stephen
```

Over alle elementen in een lijst *itereren* (ze één voor één aflopen) is zo een veel voorkomende operatie dat er in Python ook een directere, intuïtieve manier voor bestaat:

In [7]:

```
for naam in L:
    print(naam)
```

```
tom
toon
daphne
stephen
```

Verder zijn er nog enkele handige afkortingen in Python; zo kunnen we negatieve indices gebruiken om de lijst van achteren naar voren te indexeren. Het laatste element van een lijst L met n elementen krijgen we via $L[-1]$, het voorlaatste met $L[-2]$ enzovoort tot het eerste met $L[-n]$.

Met volgende code maken we een lijst K die de elementen van lijst L in omgekeerde volgorde bevat:

In [8]:

```
L=[1,2,3,4]
K=[]
for i in range(-1,-len(L)-1,-1):
    # print(i,L[i])  ## Haal commentaar weg voor deze regel om stap per stap te zien wa
    t wordt toegevoegd
    K.append(L[i])

print(L,K)
```

[1, 2, 3, 4] [4, 3, 2, 1]

Let wel op dat je steeds een geldige index gebruikt, anders krijg je een foutmelding:

In [9]:

```
L=[0,1,2,3]
print(L[4])  # 4 geeft het 5-de element wat niet bestaat
```

```
-----
-
IndexError                                Traceback (most recent call las
t)
<ipython-input-9-3b4d3a268acd> in <module>
      1 L=[0,1,2,3]
----> 2 print(L[4])  # 4 geeft het 5-de element wat niet bestaat

IndexError: list index out of range
```

Elementen weghalen uit een lijst

We kunnen ook elementen terug weghalen uit een lijst. Als we een element in het midden weghalen wordt de lege plaats gewoon uit de lijst gehaald. Er zijn twee manieren om een element te verwijderen uit een lijst L :

- op basis van de index i met `del(L[i])`
- op basis van het item ; `L.remove(x)` verwijdert het eerste voorkomen van x uit de lijst

In [10]:

```
L=[1,2,3,4,5,6]
del(L[3])
print(L)
```

[1, 2, 3, 5, 6]

In [11]:

```
L=[1,2,3,2,4]
L.remove(2)
print(L)
```

[1, 3, 2, 4]

Zorg ervoor dat je met *remove* enkel elementen uit de lijst weghaalt die ook werkelijk in de lijst voorkomen. Met "*x in L*" kan je testen of een element voorkomt. Als je een element probeert weg te halen dat niet in de lijst zit, dan krijg je een foutmelding:

In [12]:

```
L=[1,2,3,4]
x=int(input("Element om weg te halen : "))
if x in L:
    print("OK!")
else:
    print("o o ...")

L.remove(x)
print(L)
```

Element om weg te halen : 42
o o ...

```
-----
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-12-11e861d4679b> in <module>
      6     print("o o ...")
      7
----> 8 L.remove(x)
      9 print(L)
```

ValueError: list.remove(x): x not in list

Lijsten samenvoegen

Met behulp van de operatie *+* kunnen we twee lijsten samenvoegen. Dit noemen we *concatenatie* van twee lijsten. Concatenatie van twee lijsten creëert een nieuwe lijst die bestaat uit de samenvoeging van beide lijsten:

In [13]:

```
K=[1,2,3]
L=[4,5,6]
KL=K+L
print(KL)
```

[1, 2, 3, 4, 5, 6]

Merk op dat we op deze manier ook een nieuw element kunnen toevoegen aan een lijst, door de lijst te concateneren met de lijst die bestaat uit het ene element dat we willen toevoegen.

In [14]:

```
L=[1,2,3]
L.append(4)
L=L+[5]
print(L)
```

```
[1, 2, 3, 4, 5]
```

De *append* functie is echter efficiënter dan concatenatie, omdat dan de lijst wordt uitgebreid in plaats van een volledig nieuwe lijst te maken. Later zullen we dieper ingaan op het verschil tussen het gebruik van *append* en concatenatie, wanneer we over *mutable* en *immutable* data types zullen spreken.

Deellijsten nemen

Met de vierkante haken kunnen we niet enkel 1 specifiek element uit de lijst selecteren, maar kunnen we ook een deellijst nemen. $L[i:j]$ maakt een nieuwe lijst aan die bestaat uit de elementen vanaf index i tot index j ; het element op index j zelf is niet inbegrepen. i en j kunnen weggelaten worden uit $L[i:j]$, en dan betekent dat respectievelijk $i=0$ en $j=len(L)$.

In [15]:

```
L=[0,1,2,3,4,5,6,7,8]
K1=L[2:5]
K2=L[:3]
K3=L[4:]
print(K1,K2,K3)
```

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

We zouden deellijsten als volgt kunnen gebruiken om het element op positie i uit de lijst te verwijderen:

In [16]:

```
L=[0,1,2,3,4,5,6]
i=2

# Verwijder element op index i
L=L[:i] + L[i+1:]

print(L)
```

```
[0, 1, 3, 4, 5, 6]
```

Lijst sorteren

Een handige operatie is $L.sort()$ dat een lijst L sorteert:

In [17]:

```
L=[6,4,8,1,9,2,3]
L.sort()
print(L)
```

```
[1, 2, 3, 4, 6, 8, 9]
```

Voorbeelden

We bekijken nu een aantal voorbeelden met lijsten. In het eerste voorbeeld lezen we een lijst getallen in. Afsluiten doen we door een lege invoer te geven:

In [18]:

```
getallen=[]
next=True

while next:
    s=input("Volgend getal ? ")
    if s=="":
        next=False
    else:
        getallen.append(float(s)) # s is een string; met float(s) converteren we dat n
aar een komma-getal

print(getallen)
```

```
Volgend getal ? 1
Volgend getal ? 3.141592
Volgend getal ? 42
Volgend getal ? 
[1.0, 3.141592, 42.0]
```

We bereken het gemiddelde van een lijst getallen $x_1, x_2, \dots, x_n$ met de formule

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

In [19]:

```
som=0.0
for x in getallen:
    som=som+x
gemiddelde=som/len(getallen)

print("Het gemiddelde is: ",gemiddelde)
```

Het gemiddelde is: 15.380530666666667

Vervolgens kunnen we de standaardafwijking berekenen met volgende formule:

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x - x_i)^2}{n - 1}}$$

In [20]:

```
import math

ssom=0.0
for x in getallen:
    ssom=ssom+(x-gemiddelde)**2 # ** is machtsverheffing
sigma=math.sqrt(ssom/(len(getallen)-1))

print("Standaardafwijking is:", sigma )
```

Standaardafwijking is: 23.077992000276396

Strings zijn constante lijsten

Merk op dat we strings als constante lijsten kunnen beschouwen; alle operaties die op lijsten uitgevoerd kunnen worden zonder de lijst te veranderen, werken ook op strings:

- i-de element van een string:

In [21]:

```
s="abcd"
print(s[2])
```

c

- twee strings "concateneren":

In [22]:

```
s="ab"+"ba"
print(s)
```

abba

- Testen of een karakter voorkomt in een string:

In [23]:

```
print("a" in "bdfdfs")
```

False

- deelstring nemen:

In [24]:

```
s="ABCDEFGH"
print(s[4:-1])
```

ef

We kunnen strings echter niet aanpassen:

In [25]:

```
s="bdcscd"
# s.sort() # werkt niet
# s[2]="f" # werkt niet
# s.append("c") # werkt niet
```

Als we bijvoorbeeld het i-de karakter van een string willen wissen, dan moeten we een nieuwe string aanmaken zonder dat karakter. Die nieuwe string kunnen we dan opnieuw toekennen aan dezelfde variabele:

In [26]:

```
s="abcd"

# verwijder element op index 2 ('c')
s=s[:2]+s[3:]
print(s)
```

abd

Een handige functie die we in een aantal voorbeelden gaan gebruiken, is de functie *split* die we op een string kunnen toepassen. Als *s* een string is, dan is *s.split()* de string *s* gesplitst op witruimte (spaties, tabs, ...). Het resultaat wordt gegeven als een lijst van strings die samen *s* vormen:

In [27]:

```
s="Wat is de zin van deze zin? Het is onzin."
L=s.split()
print(L)
print("3de woord: ",L[2])
```

```
['Wat', 'is', 'de', 'zin', 'van', 'deze', 'zin?', 'Het', 'is', 'onzin.']
3de woord:  de
```

Als je een ander karakter als splitsing wil gebruiken, kan je dat als argument meegeven:

In [28]:

```
s="CV-screening-20/2/20-final"
print(s.split("-"))
```

```
['CV', 'screening', '20/2/20', 'final']
```

Wil je een hoop strings terug aan elkaar plakken, dan kan dat met *separator.join(lijt)*:

In [29]:

```
s="CV-screening-20/2/20-final"
L=s.split("-")
s2=" ".join(L)
print(s2)
```

CV screening 20/2/20 final

Speciale "lijsten"

Merk op dat we in de *for*-loop soms gebruik maken van de functie *range*. Je kan aan deze functie beschouwen alsof ze een lijst teruggeeft (in werkelijkheid maakt ze niet de hele lijst aan in het geheugen, maar is het een object dat een voor een de elementen geeft, maar dat is momenteel niet belangrijk).

Inderdaad; volgende stukjes code doen exact hetzelfde:

In [30]:

```
for i in range(5):
    print(i, end=" ") # end = " " zorgt dat na de output een spatie komt i.p.v. een ne
wline
print()
for i in [0,1,2,3,4]:
    print(i)

print("range(5)=",list(range(5))) # je moet list toepassen om range ineens alles te la
ten berekenen
```

0 1 2 3 4

0

1

2

3

4

range(5)= [0, 1, 2, 3, 4]

range heeft nog wat extra parameters en kan in een van de volgende vormen gebruikt worden:

- *range(n)* betekent $[0, 1, \dots, n-1]$
- *range(s,e)* betekent $[s, s+1, s+2, \dots, e-1]$
- *range(s,e,d)* betekent $[s, s+d, s+2d, \dots, s+kd]$ waarbij $s+kd < e \leq s+(k+1)*d$

Oefeningen

Oefening 1:

Schrijf een functie die van een lijst de cumulatieve som berekent. Dat is, gegeven een lijst L , bereken een lijst C die even lang is als L en zodat $C[i]=L[0]+L[1]+\dots+L[i]$. Als $L=[2,2,5]$, dan moet C gelijk zijn aan: $[2,4,9]$.

In [31]:

```
# Jouw antwoord hier
```

Oefening 2:

Gegeven een string s , geef de omgekeerde string. Dus als $s='bcd'$, dan bereken je $'dcb'$

In [32]:

```
# Jouw antwoord hier
```

Oefening 3:

Ga na of een gegeven string een *palindroom* is. Een palindroom is een string die gelijk is aan zijn omgekeerde. Enkele voorbeelden: abba, legovogel, levensnevel, maandnaam, meetsysteem, ...

In [33]:

```
# Jouw antwoord hier
```

In []: