

Dictionaries (Map)

Het laatste container type in deze reeks is een zogenaamde *dictionary* of *map*. Een dictionary kan je zien als een speciale lijst; een lijst waarin je de elementen niet indexeert met hun positie *0* tot en met *len(lijt)-1*, maar met een willekeurige sleutel. Volgende code geeft weer hoe dit kan:

```
In [1]: d=dict() # maak een lege dictionary aan
        d["a"]=1
        d["b"]=2
```

Met de vierkante haken kunnen we waarden terug opvragen:

```
In [2]: print(d["a"]+d["b"])
```

3

Let wel op: dit kan enkel indien de *sleutel* ("a" en "b") in dit geval ook voorkomen in de dictionary; anders krijg je een foutmelding:

```
In [3]: print(d["c"])
```

```
-----
KeyError                                Traceback (most recent call last)
<ipython-input-3-e4e45004c53e> in <module>
----> 1 print(d["c"])

KeyError: 'c'
```

Je kan testen of een key al toegekend is door gebruik te maken van *in*:

```
In [4]: key=input("Key? ")
        if key in d:
            print(d[key])
        else:
            print("Key bestaat niet.")
```

Key bestaat niet.

Over alle keys in een dictionary lopen doen we opnieuw met een *for*-loop. Let wel: er is geen enkele garantie over de volgorde waarin de keys worden doorlopen. Dit hoeft niet de volgorde van toevoeging te zijn en ook niet gesorteerd; de volgorde is willekeuring.

```
In [5]: print(d)
        for key in d:
            print(key,"|->",d[key])
```

```
{'a': 1, 'b': 2}
a |-> 1
b |-> 2
```

Soms is het handig om een dictionary manueel al in te vullen. Dit kan met de notatie { key:value, key:value, ... }. Neem bijvoorbeeld volgende dictionary die namen van kleuren mapt op getallen:

```
In [6]: code={"geel":1,"rood":2,"groen":3,"blauw":4}
        print(code["geel"])
```

```
1
```

Merk op dat je een lege dictionary kan aanmaken als volgt: `d={}.` Dit is ook de reden voor de inconsistentie bij het aanmaken van lege containers:

```
In [7]: L=[] # lege lijst
        d={} # lege dictionary
        t=() # lege tuple
        s=set() # lege set ; we kunnen {} niet gebruiken want dat is al in gebruik voor dictionaries

        # alternatief om lege containers aan te maken:
        L=list()
        d=dict()
        t=tuple()
        s=set()

        # geïnitieerde containers:
        L1=[1]
        s1="a"
        d1={"a":1}
        t1=("a")
        s1={"a"} # nu kunnen we wel de notatie met { en } gebruiken omdat er geen ambiguïteit is
```

Dictionaries zijn erg handig, bijvoorbeeld om te tellen hoe vaak bepaalde getallen, letters, woorden, ... voorkomen:

```
In [8]: # tel hoe vaak elke letter voorkomt in volgende zin:
zin="Wat is de zin van het leven? Wat is de zin van deze zin? Is het onzin? Ge
lukkig hebben we zin in Python. (de programmeertaal, niet de slang)"
teller={} # lege dictionary

for letter in zin:
    if letter in teller:
        teller[letter]=teller[letter]+1 # bestaat al, dus een bij optellen
    else:
        teller[letter]=1 # is de eerste keer dat we deze letter zien, dus co
unt is 1

print("Frequentie van de letters:")
for key in teller:
    if key in "abcdefghijklmnopqrstuvwxyz":
        print(key,"|->",teller[key])
```

Frequentie van de letters:

```
a |-> 8
t |-> 7
i |-> 10
s |-> 4
d |-> 5
e |-> 17
z |-> 6
n |-> 14
v |-> 3
h |-> 4
l |-> 4
o |-> 3
u |-> 1
k |-> 2
g |-> 3
b |-> 2
w |-> 1
y |-> 1
p |-> 1
r |-> 3
m |-> 2
```

Enkele uitbreidingen: we kunnen een lijst opvragen van alle keys in een dictionary met de functie `dict.keys()`. Dit kan handig zijn als we bijvoorbeeld de keys gesorteerd willen overlopen:

```
In [9]: print("Frequentie van de letters:")
letters=list(teller.keys()) # We moeten er expliciet een lijst van maken; z
oals bij range
letters.sort()
for key in letters:
    if key in "abcdefghijklmnopqrstuvwxyz":
        print(key,"|->",teller[key])
```

Frequentie van de letters:

```
a |-> 8
b |-> 2
d |-> 5
e |-> 17
g |-> 3
h |-> 4
i |-> 10
k |-> 2
l |-> 4
m |-> 2
n |-> 14
o |-> 3
p |-> 1
r |-> 3
s |-> 4
t |-> 7
u |-> 1
v |-> 3
w |-> 1
y |-> 1
z |-> 6
```