Tupels en woordenboeken

ZIE OOK
❮ 110–111 Soorten data
❮ 128–129 Lijsten

Python gebruikt lijsten om data op volgorde te houden, maar heeft ook andere soorten data om informatie in op te slaan: 'tupels' en 'woordenboeken'. Dit soort datatypen, waarin veel items kunnen worden opgeslagen, heten 'containers'.

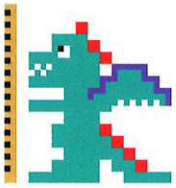
Tupels

Tupels lijken wel wat op lijsten, maar de items erin kunnen niet worden veranderd. Als je eenmaal een tupel hebt geplaatst, blijft die altijd gelijk.

Tupels staan altijd tussen ronde haken

```
>>> dragonA = ('Sam', 15, 1.70)
>>> dragonB = ('Fiona', 16, 1.68)
```

De items in een tupel worden gescheiden door komma's



◁ Wat is een tupel?

Een tupel bevat items gescheiden door komma's en staan tussen haken. Tupels dienen om meerdere bits data tezamen op te slaan, zoals de naam, leeftijd en hoogte van een draak.

▷ Een item uit een tupel pakken

Als je een item uit een tupel wilt halen, doe je dat via zijn positie (zijn index) in de tupel. Tupels tellen vanaf nul, net als lijsten en strings.

```
>>> dragonB[2]
1.68
```

Hiermee selecteer je het item van positie 2

◁ Een tupel splitsen in variabelen

Wijs drie variabelen toe aan de tupel 'dragonA' – 'name', 'age' en 'height'. Python splitst de tupel nu in drie items en plaatst elk daarvan in een variabele.

```
>>> name, age, height = dragonA
>>> print(name, age, height)
Sam 15 1.7
```

De items die de tupel 'dragonA' vormen, worden apart weergegeven

▷ Tupels in een lijst plaatsen

Tupels kunnen in een lijst worden geplaatst omdat containers binnen elkaar kunnen staan. Met deze codering kun je een lijst tupels aanmaken.

Maak een lijst met tupels aan, genoemd 'dragons'

Lijsten staan tussen vierkante haken

```
>>> dragons = [dragonA, dragonB]
>>> print(dragons)
[('Sam', 15, 1.7), ('Fiona', 16, 1.68)]
```

Elke tupel staat tussen ronde haken binnen de vierkante haken van de lijst

Python geeft alle items in de lijst weer, niet alleen de namen van de tupels

Woordenboeken

Woordenboeken zijn net lijsten, maar ze zijn gelabeld. Deze labels, 'sleutels' genoemd, herkennen items, geen indexgetallen. Elk item in een woordenboek heeft een sleutel en een waarde. Items in een woordenboek hoeven niet op volgorde te staan en de inhoud van een woordenboek kan worden aangepast.

▷ Een woordenboek maken

Met dit programma maak je een woordenboek genaamd 'age'. De sleutel voor elk item is de naam van een persoon, de waarde is zijn leeftijd.

Woordenboeken staan tussen accolades

Items in een woordenboek worden gescheiden door komma's

Zet een dubbele punt tussen een sleutel en een waarde

```
>>> age = {'Mary': 10, 'Sanjay': 8}
```

Een sleutel werkt op dezelfde manier als een indexgetal

Een opgeslagen waarde in het woordenboek (altijd na een dubbele punt)

```
>>> print(age)
{'Sanjay': 8, 'Mary': 10}
```

Naam van het woordenboek

De sleutel van dit item is 'Sanjay'

De waarde van 'Mary' is 10

◁ Het woordenboek printen

De volgorde van de items kan veranderen, want de posities ervan staan niet vast.

▷ Een nieuw item toevoegen

Je kunt een nieuw item aan het woordenboek toevoegen door het te labelen met een nieuwe sleutel.

Naam woordenboek

Nieuwe sleutel

```
>>> age['Owen'] = 11
>>> print(age)
{'Owen': 11, 'Sanjay': 8, 'Mary': 10}
```

Voegt een nieuw item toe aan het woordenboek

De nieuwe waarde staat nu in het woordenboek

De bestaande waarden zijn nog aanwezig

◁ Een waarde veranderen

Wijs een nieuwe waarde toe aan een bestaande sleutel om de waarde ervan te veranderen.

```
>>> age['Owen'] = 12
>>> print(age)
{'Owen': 12, 'Sanjay': 8, 'Mary': 10}
```

Wijs een nieuwe waarde toe aan het item met het label 'Owen'

De waarde voor 'Owen' is gewijzigd

▷ Een item verwijderen

Als je een item uit een woordenboek verwijdert, heeft dit geen invloed op andere items, omdat die worden geïdentificeerd door hun sleutel, niet door hun positie in het woordenboek.

Dit verwijdert het item met het label 'Owen'

```
>>> del age['Owen']
>>> print(age)
{'Sanjay': 8, 'Mary': 10}
```

Het gelabelde item 'Owen' verschijnt niet langer in het woordenboek

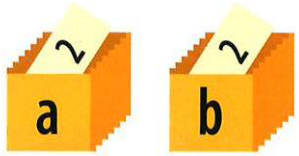
Lijsten in variabelen

De manier waarop Python lijsten opslaat in variabelen lijkt op het eerste gezicht wat vreemd. Maar als je wat beter kijkt naar hoe het achter de schermen werkt, wordt het allemaal snel duidelijk.

ZIE OOK
❮ 108–109 Variabelen in Python
❮ 128–129 Lijsten

Weet je nog hoe variabelen alleen waarden opslaan?

Variabelen zijn net dozen met waarden erin. Je kunt de waarde in een variabele kopiëren en opslaan in een andere doos. Het is net alsof je de waarde in doos 'a' fotokopieert en deze kopie opslaat in doos 'b'.



△ **Hoe variabelen werken**
Elke variabele is net een doos waarin een vel papier zit waarop een waarde is geschreven.

1 Wijs een waarde toe aan een variabele
Wijs de waarde 2 toe aan variabele 'a' en wijs dan de waarde in 'a' toe aan variabele 'b'. De waarde 2 wordt gekopieerd en opgeslagen in 'b'.

```
>>> a = 2
>>> b = a
>>> print('a =', a, 'b =', b)
a = 2 b = 2
```

Hiermee kopieer je de inhoud van 'a' naar 'b'
Nu bevatten 'a' en 'b' beide de waarde 2
Hiermee print je de variabele namen met hun waarden

2 Verander een waarde
Als je de waarde verandert die opgeslagen is in een variabele, heeft dit geen invloed op de opgeslagen waarde in een andere variabele. Op dezelfde manier zal wat geschreven is op een vel papier in doos 'a' geen invloed hebben op wat geschreven is op het papier in doos 'b'.

```
>>> a = 100
>>> print('a =', a, 'b =', b)
a = 100 b = 2
```

Verander de waarde van 'a' in 100
Nu bevat 'a' 100, maar bevat 'b' nog steeds 2

3 Verander een andere waarde
Verander de waarde in 'b' in 22. Variabele 'a' bevat nog steeds 100. Zelfs hoewel de waarde van 'b' werd gekopieerd vanuit 'a', zijn ze nu onafhankelijk van elkaar – 'b' veranderen wijzigt 'a' niet.

```
>>> b = 22
>>> print('a =', a, 'b =', b)
a = 100 b = 22
```

'b' bevat nu 22, maar 'a' is nog steeds 100

Wat gebeurt er als een lijst in een variabele wordt geplaatst?

Met het kopiëren van een waarde naar een variabele maak je twee onafhankelijke versies van de waarde. Dit werkt als de waarde een getal is, maar wat als het iets anders is? Met een lijst in een variabele werkt het iets anders.

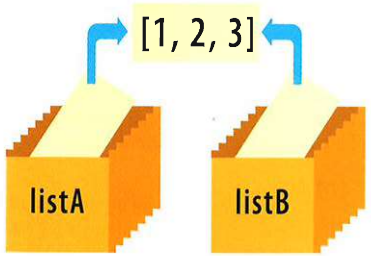
1 Kopieer een lijst
Sla de lijst [1, 2, 3] op in een variabele met de naam 'listA'. Sla dan de waarde van 'listA' op in een andere variabele met de naam 'listB'. Beide variabelen bevatten nu [1, 2, 3].

2 Verander lijst A
Verander de waarde in 'listA[1]' in 1000. 'listB[1]' bevat nu ook 1000. Het veranderen van de originele lijst heeft de kopie van de lijst eveneens gewijzigd.

3 Verander lijst B
Verander de waarde van 'listB[2]' in 75. 'listA[2]' is nu ook 75. Door de kopie van de lijst te veranderen heb je ook het origineel veranderd.

```
>>> listA = [1, 2, 3]
>>> listB = listA
>>> print('listA =', listA, 'listB =', listB)
listA = [1, 2, 3] listB = [1, 2, 3]
>>> listA[1] = 1000
>>> print('listA =', listA, 'listB =', listB)
listA = [1, 1000, 3] listB = [1, 1000, 3]
>>> listB[2] = 75
>>> print('listA =', listA, 'listB =', listB)
listA = [1, 1000, 75] listB = [1, 1000, 75]
```

Gebruik vierkante haken om een lijst te maken
Hiermee print je de variabele namen naast hun waarden om te zien wat ze bevatten
'listA' en 'listB' bevatten beide dezelfde waarde
Dit verandert het tweede item in de lijst, omdat lijsten tellen vanaf nul
Het tweede item van zowel 'listA' als 'listB' is veranderd
Dit is het derde item in de lijst
Het derde item van zowel 'listA' als 'listB' is veranderd



△ **Wat gebeurt er?**
Een variabele met een lijst erin bewaart die lijst niet zelf; hij bevat alleen een link naar de lijst. Als je de waarde in 'listA' kopieert, kopieer je dus de link. Daarom bevatten zowel 'listA' als 'listB' een link naar dezelfde lijst.

TIPS VAN EXPERTS

Lijsten kopiëren

Gebruik de functie 'copy' om een aparte kopie van de lijst te maken. 'listC' bevat nu een link naar een geheel nieuwe lijst met waarden die kopieën zijn van die in 'listA'. Als je 'listC' verandert, wijzig je 'listA' niet en andersom evenmin.

```
>>> listC = listA.copy()
```

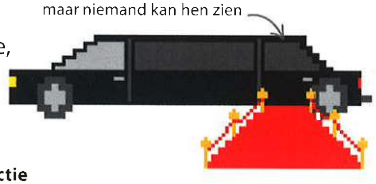
Variabelen en functies

Variabelen die binnen een functie zijn gemaakt (lokale) en variabelen in het hoofdprogramma (globale) werken op verschillende manieren.

ZIE OOK
‹ 130–131 Functies
Vormen 158–159 › maken

Lokale variabelen

Lokale variabelen bestaan alleen binnen een enkele functie, dus het hoofdprogramma en andere functies kunnen ze niet gebruiken. Probeer je een lokale variabele buiten de functie te gebruiken, dan krijg je een foutmelding.



1 Variabele binnen de functie
Maak een lokale variabele aan: 'a' binnen 'func1'. Print de waarde van 'a' door 'func1' op te roepen vanuit het hoofdprogramma.

```
>>> def func1():
        a = 10
        print(a)
>>> func1()
10
```

Oproepen van 'func1' print de waarde van 'a'

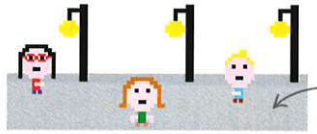
2 Variabele buiten de functie
Als je probeert 'a' rechtstreeks vanuit het hoofdprogramma te printen, krijg je een foutmelding. 'a' bestaat alleen binnen 'func1'.

```
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
```

Het hoofdprogramma kent 'a' niet, dus geeft het een foutmelding

Globale variabelen

Een globale variabele is gemaakt in het hoofdprogramma. Andere functies kunnen hem lezen, maar de waarde ervan niet wijzigen.



1 Variabele buiten de functie
Maak een globale variabele 'b' in het hoofdprogramma. De nieuwe functie ('func2') kan de waarde van 'b' lezen en printen.

```
>>> b = 1000
>>> def func2():
        print(b)
>>> func2()
1000
```

'func2' kan de waarde van 'b' zien omdat 'b' een globale variabele is

Printen van 'func2' geeft je de waarde die is opgeslagen in 'b'

2 Dezelfde globale variabele
We kunnen 'b' ook rechtstreeks vanuit het hoofdprogramma printen. Je kunt 'b' overal zien, omdat hij niet binnen een functie is gemaakt.

```
>>> print(b)
1000
```

Globale variabele 'b' kan overal in het hoofdprogramma worden toegepast

Variabelen als input in functies

Als een variabele wordt gebruikt als input voor een functie, wordt zijn waarde naar een nieuwe variabele gekopieerd. Als je zo de waarde van deze nieuwe lokale variabele binnen de functie wijzigt, verandert dat de waarde van de originele variabele niet.

1 Waarden veranderen binnen een variabele
'func3' gebruikt input 'y', een lokale variabele. Het print de waarde van 'y', verandert die waarde in 'bread' en print de nieuwe waarde.

```
>>> def func3(y):
        print(y)
        y = 'bread'
        print(y)
>>> z = 'butter'
>>> func3(z)
butter
bread
```

'y' bevat de waarde die hem werd toegewezen toen 'func3' werd opgeroepen

'y' bevat hier 'bread'

Hiermee maak je de globale variabele 'z'

De input 'y' bevat nu de waarde die 'z' werd toegewezen toen 'func3 werd opgeroepen

2 Print variabele Printen van de waarde van 'z' na het oproepen van 'func3' toont dat die niet is gewijzigd. Oproepen van 'func3' kopieert de waarde in 'z' ('butter') naar lokale variabele 'y', maar laat 'z' onveranderd.

```
>>> print(z)
butter
```

Print de waarde in globale variabele 'z' nadat 'func3' is gestopt

Lokale variabele 'y' in 'func3' bevat een kopie van de waarde in 'z'. Hoewel 'y' is gewijzigd in 'bread', is 'z' niet aangetast en heet nog 'butter'

Een globale variabele maskeren

Een globale variabele kan niet worden gewijzigd door een functie. Een functie die probeert een globale variabele te wijzigen maakt in feite een lokale variabele aan met dezelfde naam. Hij verhult of 'maskeert' de globale variabele met een lokale versie.

1 Een globale variabele veranderen
Aan globale variabele 'c' is de waarde 12345 gegeven. 'func4' geeft 'c' de waarde 555 en print deze. Het lijkt alsof onze globale variabele 'c' is veranderd.

```
>>> c = 12345
>>> def func4():
        c = 555
        print(c)
>>> func4()
555
```

Begin-waarde in globale variabele 'c'

Print de waarde van 'c' binnen 'func4'

2 Een variabele printen
Als we 'c' printen van buiten de functie, zien we dat 'c' helemaal niet is veranderd. 'func4' print alleen de waarde van de nieuwe lokale variabele – ook 'c' geheten.

```
>>> print(c)
12345
```

De waarde in globale variabele 'c' is niet gewijzigd

TIPS VAN EXPERTS

Functies oproepen

Je kunt functies op twee manieren oproepen.

functie(a)

In Python worden items van data 'objecten' genoemd. Sommige functies worden opgeroepen door er het data-object ('a') aan toe te wijzen.

a.functie()

Andere functies worden opgeroepen door hun naam toe te voegen aan het eind van het data-object ('a'), na een punt.