Python: Werken met data

D. Leeuw

25 juni 2025
0.0.0

1 Variabelen

Een computer slaat data op in het geheugen. Dit gebeurt door een bepaald adres in het geheugen te selecteren en daar de 1-en en 0-en die onze data vertegenwoordigen weg te schrijven. Wij als programmeurs hebben echter geen idee op welke plek (welk adres) onze data weggeschreven wordt. Dat wordt bepaald door het besturingssysteem, dus hebben we een andere manier nodig om te verwijzen naar onze data. We gebruiken variabelen (variables) om onze data in op te slaan. De naam "variabelen" zegt al dat we de data ook kunnen wijzigen, dit in tegenstelling tot constanten (constants) die een vaste niet wijzigbare data hebben.

Het geven van een waarde aan een variabele is in Python heel simpel:

```
x = 5
```

We gebruiken het = teken om aan Python te vertellen dat de variabele `x` de waarde **5** krijgt. Hierna kunnen we in onze code `x` gebruiken en zal daarvoor in de plaats **5** worden weergegeven:

```
x = 5
print(x)
```

Zo kunnen we ook onze `print` opdracht van de "Hello World!" aanpassen:

```
msg = "Hello World!"
print(msg)
```

Een variabele is hoofdlettergevoelig (case sensitive), een variabele met de naam a en met de naam A zijn twee verschillende variabelen. Voor het maken van namen van variabelen zijn er in Python een aantal regels:

- Een variabele naam moet beginnen met een letter of een underscore, hij mag dus niet beginnen met een cijfer
- Een variabele naam mag alleen letters, cijfers en underscores bevatten
- Variabele namen zijn case sensitief (Fruit, fruit en FRUIT zijn drie verschillende variabelen)
- Een variabele mag geen Python keyword zijn. Zie W3Schools voor een lijst met Python keywords (https://www.w3schools.com/python/python_ref_keywords.asp

Een goed gekozen variabele naam kan je in de toekomst helpen bij het herlezen van je code. Een variabele naam als `x` zegt niets, de naam `msg` (een afkorting voor message) is al een stuk duidelijker. Soms is het handig om een

variabele een naam te geven die bestaat uit meer dan één woord. Volgens de bovenstaande regels mag een spatie niet voorkomen in een variabele naam en `paswoordvandegebruiker` is niet lekker leesbaar. Omdat variabelen hoofdlettergevoelig zijn zou een mogelijke oplossing voor de leesbaarheid kunnen zijn `PaswoordVanDeGebruiker`, of wat ook mag `paswoord_van_de_gebruiker` of een combinatie. Wat je ook tegenkomt is de variant van de eerste optie maar waarbij het eerste woord niet met een hoofdletter geschreven is `paswoordVanDeGebruiker`. Kortom er zijn verschillende mogelijkheden te verzinnen om de leesbaarheid te vergroten. Als je binnen een script een variant kiest, gebruikt deze variant door over al in je script en ga geen varianten door elkaar gebruiken. Dat komt de leesbaarheid niet te goede.

Python geeft je de mogelijkheid om aan meerdere variabelen dezelfde waarde te geven in één enkel commando:

```
x = y = z = "test"
print(x)
print(y)
print(z)
```

Zowel `x`, als `y`, als `z` hebben nu de inhoud “test”.

Je kan ook meerdere waardes toekennen aan meerdere variabelen in één regel:

```
x, y, z = "test", 1, "regel 1"
print(x)
print(y)
print(z)
```

De variabelen `x`, `y` en `z` hebben nu elk een andere waarde.

Je kan ook meerdere variabelen tegelijk printen:

```
x = "aap"
y = "noot"
z = "mies"
print(x, y, z)
```

Het wordt één lange string. In de string van `x` en `y` zou je een spatie kunnen zetten na de waarde en voor de eind quote om spaties in de string te krijgen. Je mag inplaats van de komma ook de plus gebruiken, dat betekent voor strings hetzelfde. Let op! voor getallen (integers) gebeurt er wat anders, de getallen worden met een plus teken bij elkaar opgeteld.

2 Data Types

Data moet opgeslagen worden in het geheugen van de computer om ermee te kunnen werken. We weten dat het computergeheugen werkt met 1-en en 0-en. Wij werken meestal met tekst of getallen, onze data moet dus vertaald worden naar 1-en en 0-en.om opgeslagen te kunnen worden. Om een 2 op te slaan in het geheugen zouden we direct een binair getal kunnen nemen: 0000 0010, maar hoe zit dat dan met -2 of met 2,0? En ook hoe gaan we om met tekst en leestekens? De kunst is om zo effcient mogelijk met het geheugen om te gaan zodat er zoveel mogelijk data in zo min mogelijk geheugen past. Om deze efficientie te bereiken is het handig als een programmeertaal van te voren weet wat voor "waarde" het moet opslaan in het geheugen. Bij veel programmeertalen moet je van te voren opgeven wat voor soort waarde je wilt opslaan. In Python hoef je dat niet meteen aan te geven. Python raad wat je bedoelt. Python kent o.a. de volgende data typen:

int integer - Een positief of negatief heel getal

float Een getal met een komma, niet gehele getallen

str string - Een reeks van karakters (chars). Een karakter kan een letter, leesteken of cijfer zijn

complex Imaginaire getallen (5j)

Als je in Python de volgende variabelen maakt:

```
x = 5
y = "Hello World!"
z = 6.3
```

Dan zal Python van `x` een **int** maken, van `y` een **str** en van `z` een **float**.

Een string wordt bepaald door de quotes. Het mogen in Python enkele of dubbele quotes zijn:

```
x = "waarde 1"
y = 'waarde 1'
```

beide toekenningen (`x` en `y`) hebben dezelfde betekenis, het zijn beide strings.

2.1 Casting

Soms wil je dat data van een specifiek type is. Je zou bijvoorbeeld een getal willen kunnen behandelen als een string of een string als een getal. Om het ene data type om te zetten in het andere moeten we de data "casten". Casting is het omzetten van het ene type data in het andere.

Een string omzetten in een getal doen we zo:

```
a = "56"
x = int(a)
print(type(a))
print(type(x))
```

Een getal omzetten naar een string met:

```
a = 56
x = str(a)
print(type(a))
print(type(x))
```

We zien dat we functies/ commando's hebben die de omzetting doen. `int` zet een data type om naar een integer, als dat mogelijk is. Als we proberen om een karakter om te zetten naar een `int` dan krijgen we een error melding:

```
>>> a = "a"
>>> print(int(a))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

De `ValueError` geeft aan dat een letter a niet omgezet kan worden naar een integer.

3 Arithmetic operators

Operators worden gebruikt om acties (operaties) uit te voeren op variabelen en waarden. Een simpel voorbeeld:

```
print(42+3)
```

In dit voorbeeld is (+) de operator. Het vertelt de computer dat er iets moet gebeuren, namelijk optellen.

De Arithmetic operators zijn de operators voor rekenkundige bewerkingen. Python kent devolgende arithmetic operators:

4 Ordenen van data

Met variabelen kan je een waarde aan een variabele geven. Maar stel nu dat je 300 computertypes hebt en deze wil je opslaan. Als je voor elk computertype een variabele gaat gebruiken dan heb je er 300 nodig. Het zou makkelijker

Operator	Betekenis	Hoe te gebruiken
+	optellen van waarden; samenvoegen van strings	6+5 "Hello "+ "World"
-	aftrekken van waarden	6-5
*	vermenigvuldigen van waarden	6*5
/	delen van waarden	6/5
**	machtensverheffen	6**5
%	modulo rekenen	14%12
//	geheeltallige deling (floor devision)	14//12

Tabel 1: Comparison operators

zijn als we dan bijvoorbeeld een lijst zouden kunnen maken met één naam waarin alle types zitten. Dat is waar het in dit stuk om gaat hoe kan ik data op een makkelijke manier ordenen.

4.1 list (array)

Een list of zoals het in andere programmeertalen wel wordt genoemd een array is een lijst van waarden die aan één variable is toegekend. Voorbeeld:

```
fruit = ["appel", "peer", "banaan", "sinaasappel"]
print(fruit)
```

Een list wordt gedefinieerd door de blockhaken met daartussen de waarden die er in de list zitten gescheiden door komma's.

We kunnen in Python verschillende variabelen vullen door er een lijst met waarden aan te geven. Dan kunnen we ook doen met een list:

```
fruit = ["appel", "peer", "banaan", "sinaasappel"]
n, m, o, p = fruit
print(n)
print(m)
print(o)
print(p)
```

Een list is een geïndexeerde lijst. Het eerste item staat op plek 0, het tweede op plek 1, etc. Met print kan je dat ook zien:

```
fruit = ["appel", "peer", "banaan", "sinaasappel"]
print(fruit[0])
print(fruit[2])
```

Een index mag ook negatief zijn, dan is -1 het laatste element van de lijst en -2 de voorlaatste.

```
fruit = ["appel", "peer", "banaan", "sinaasappel"]
print(fruit[-1])
print(fruit[-3])
```

Omdat een list een geïndexeerde lijst is kunnen er ook dubbele waarden voor komen:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "banaan"]
print(fruit)
```

We kunnen op basis van de index ook een waarde wijzigen:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "banaan"]
print(fruit)
fruit[2] = "kiwi"
print(fruit)
```

De eerste banaan in de lijst is nu een kiwi geworden.

Samengevat is een list dus geordend, staat hij duplicaten toe en kunnen de elementen gewijzigd worden.

4.2 list manipulatie

Op basis van de index kan je ook een deel (range) van een lijst selecteren.

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
print(fruit[1:3])
```

Let op dat de range 1 tot 3 is, dus niet 1 tot en met 3. Je krijgt de elementen terug op posities 1 en 2.

Je kan ook een stuk vanaf het begin selecteren

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
print(fruit[:3])
```

Dit is dus van index 0 tot index 3.

Vanaf het einde mag ook, of beter van een positie tot het einde:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
print(fruit[1:])
```

Met deze selecties kan je ook waarden wijzigen:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
fruit[1:4] = [ "mango", "ananas", "papaya" ]
print(fruit)
```

De elementen op index 1 tot en met 3 worden overschreven met hun nieuwe waarde.

Als je meer elementen opgeeft als vervanging dan de opgegeven range groot is dan worden de extra elementen ingevoegd:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
fruit[1:1] = [ "mango", "ananas", "papaya" ]
print(fruit)
```

We selecteren een range van geen elementen (van 1 tot 1), we kiezen dus alleen de positie van het eerste element en voegen dan 3 nieuwe elementen in.

Je kan een list ook korter maken:

```
fruit = ["appel", "peer", "banaan", "sinaasappel", "kiwi"]
fruit[1:4] = [ "mango" ]
print(fruit)
```

Alle elementen op index 1 tot en met 3 worden vervangen door 1 nieuw element.

4.3 tuple

Een tuple is een list waarbij de elementen niet gewijzigd kunnen worden.

Een tuple maak je door een lijst tussen haakjes te zetten:

```
myTuple = ("appel", "peer", "banaan")
print(myTuple)
```

Voor het werken met tuples gelden dezelde regels als bij lists, op één uitzondering na, namelijk als je een tuple wilt maken met maar één element. Bij een list is dat geen enkel probleem, bij een tuple moet er een komma staan na het enige element. Doe je dat niet dan beschouwt Python de assignment als een normale variabele assignment:

```
myList  = ["apple"]
print(type(myList))
print(len(myList))

myTuple = ("apple",)
print(type(myTuple))
print(len(myTuple))

myNotTuple = ("apple")
print(type(myNotTuple))
```

4.4 Dictionaries (Key/Value pairs)

Een dictionary is een lijst met key/value pairs. In andere programmeertalen komen je ook weleens de term named list tegen. Dictionaries worden gemaakt met accolades. De key/value paren worden gescheiden van elkaar met komma's, en de key wordt van de value gescheiden door een dubbele punt. Een dictionary zit er dan zo uit:

```
myDictionary = {
        "Voornaam": "Jan",
        "Achternaam":  "Jansen",
        "Adres": "Dorpsstraat",
        "Huisnummer": 6,
        "Postcode": "9999 AA",
        "Plaats": "Nergenshuizen"
}
```

Dubbele elementen mogen niet voorkomen, een element dat later in de dictionary staat overschrijft zal een eerdere overschrijven. Hieruit volgt meteen dat data in een dictionary overschreven kan worden, en dus zijn dictionaries niet "unmutable".

Elementen in een dictionary worden aangesproken bij hun naam, bijvoorbeeld:

```
myDictionary = {
        "Voornaam": "Jan",
        "Achternaam":  "Jansen",
        "Adres": "Dorpsstraat",
        "Huisnummer": 6,
        "Postcode": "9999 AA",
        "Plaats": "Nergenshuizen"
}
print(myDictionary["Huisnummer"])
```

4.5 set, frozenset

Index