Python 2

DEEL 3, DATASTRUCTUREN

Na onze eerste kennismaking met Python is het tijd om nader naar wat complexere datastructuren te kijken: Lists en dictionaries. We duiken ook in de structuur van strings, zodat je afzonderlijke letters kunt manipuleren. Start je Python-interpreter maar alvast op!

In het vorige deel leerde je met drie datatypes in Python werken: int, float en str.

Laten we nog even kijken naar deze datatypes die we in de vorige module hebben bekeken:

Voorbeeld	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float

De bovenstaande datatypes zijn vrij eenvoudig. Maar in veel programma's heb je types met meer structuur nodig, we noemen dit ook wel **datastructuren**. We hebben dit in het vorige deel al een beetje gezien. Na een korte herhaling leer je in dit deel met twee van die belangrijke datastructuren van Python te werken. Namelijk list (lijst) en dict (bibliotheek)

Voorbeeld	Data Type
x = ["apple", "banana", "cherry"]	list
x = {"name" : "John", "age" : 36}	dict

In de taken werken hier weer in 'interactieve' modus, waarbij je de commando's intypt in de Thonny shell zonder eerst een file aan te maken.

Taak 1 - List

In veel programma's werk je niet met één specifiek gegeven, maar met een hele reeks. Een list is daarvoor ideaal. Zo maak je bijvoorbeeld een lijst met namen aan:

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
```

Een list bestaat uit elementen gescheiden door een , (comma). En kan elementen van verschillende datatypes bevatten, zoals een float, twee strings en een int. Maar vaak heeft een lijst alleen elementen van hetzelfde type. Zoals hierboven.

Python kent ook heel wat mogelijkheden om met de elementen in een list te werken in de volgende taken gaan we hier naar kijken.

Aantal elementen in een list opvragen - len ()

De functie len, die we kennen uit de vorige les om de lengte van een string terug te geven, werkt ook op een list: dan krijg je het aantal elementen in die list.

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
>>> len(namen)
>6
>>> legelijst = []
```

```
>>> len(legelijst)
>0
```
Let bij het bovenstaande op het gebruik van ' [' en ']' (rechte haken).

Opdracht

- Maak een list aan met namen in je Python-interpreter met 6 elementen (bijvoorbeeld de namen van je mede-studenten).
- Vraag met len de lengte van je list op (heb je genoeg elementen toegevoegd?).

Taak 2 – de waarde van elementen opvragen

In de vorige taak heb je een list aangemaakt met 6 elementen. In deze taak leer je verschillende manieren hoe je de waarde van deze elementen opvraagt via de positie (index).

De waarde van een element op een specifieke positie opvragen – [#]

Zo vraag je eenvoudig een element uit de lijst op een specifieke positie (ook 'index' genoemd) op:

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
>>> namen[2]
'pieter'
```

Merk op dat de positie in een lijst vanaf 0 begint te tellen: het eerste element is namen[0], het tweede namen[1], het derde namen[2] enzovoort Het laatste element kun je opvragen met [5].
>>> namen[5]
'rob'

Maar je kan in Python ook met negatieve posities werken, waarmee je vanachter in de lijst begint te tellen. Het laatste element heeft dan positie -1:

```
>>> namen[-1]
'rob'
>>> namen[-2]
'joris'
```

Opdracht

- Gebruik de list die je hebt aangemaakt in de opdracht van taak 1
- Vraag de waarde van het element op positie 3 op
- Vraag de waarde van het element op de positie -3 op
- Vraag de waarde van het element op positie 5 op
- Vraag de waarde van het element op positie -1 op

Taak 3 – Naar een element zoeken

Het kan ook andersom dus via de waarde de positie opvragen van een element.
Nu zoek je dus eigenlijk waar dit element staat en of dit element voorkomt en daarnaast kun je ook nog opvragen hoe vaak dit element voorkomt.

De positie van een element opvragen via de waarde van het element – index('x')

Je kunt ook via de waarde de positie van een element in een lijst opvragen met de functie **index():**

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
>>> namen.index('jan')
>1
>>> namen.index('pieter')
```

```
>2
>>> namen.index('koen')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: 'koen' is not in list
```

Zoals je ziet krijg je een foutmelding (ValueError) als het gevraagde element zich niet in de lijst bevindt. Voor een element dat zich meerdere keren in de lijst bevindt, geeft de functie index alleen de eerste positie terug.

Vanaf een specifieke positie zoeken naar een element

Standaard zoekt index () vanaf positie 0. Als je vanaf een specifieke positie wil zoeken naar een element kun je dit opgeven door de positie aan de functie toe te voegen:

```
>>> namen.index('jan', 2)
```

Dezelfde elementen in een lijst opvragen – count()

Als je goed hebt opgelet, zie je dat de string 'jan' twee keer in de lijst staat. Het aantal keren dat een element in een lijst staat kun je opvragen met de functie **count()**:

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
>>> namen.count('jan')
>2
>>> namen.count('pieter')
>1
>>> namen.count('koen')
>0
```

Opdracht

- Gebruik de list die je hebt aangemaakt in de opdracht van taak 1
- Voeg een van de namen nog een keer toe aan de lijst (je hebt nu 7 elementen)
- Vraag de positie van een van de namen uit de lijst op met index
- Vraag de positie van een van de namen op maar begin te zoeken na het eerste element
- Vraag met count() op hoe vaak de laatste naam die je hebt toegevoegd voorkomt
- Vraag met count() een naam op welke niet in de lijst staat.

Taak 4 – Waarde van elementen aanpassen of sorteren

De waarde van een element aanpassen

Als je een lijst hebt aangemaakt, kun je die nog altijd veranderen. In het eenvoudigste geval verander je bijvoorbeeld één element:

```
>>> namen = ['kees', 'jan', 'pieter', 'jan', 'joris', 'rob']
>>> namen[1] = 'koen'
>>> namen
['kees', 'koen', 'pieter', 'jan', 'joris', 'rob']
```

De elementen omdraaien – reverse()

Je kunt een lijst ook omdraaien of sorteren:

```
>>> namen.reverse()
>>> namen
['rob', 'joris', 'jan', 'pieter', 'koen', 'kees']
```

De elementen sorteren – sort():

Je kunt een list sorteren (in dit geval van a tot z)

```
>>> namen.sort()
>>> namen
['jan', 'joris', 'kees', 'koen', 'pieter', 'rob']
```

Opdracht

- Gebruik de list die je hebt aangemaakt in de opdracht van taak 3
- Pas de naam die dubbel voorkomt aan en vul daar een andere naam in
- Draai de elementen om zodat de laatste als eerste in de list voorkomt
- Sorteer de list op volgorde van a-z

Taak 5 – elementen toevoegen of verwijderen

Je kan aan de list ook elementen toevoegen of verwijderen

Elementen aan het einde toevoegen – append()

Een element aan het einde van een lijst toevoegen:

```
>>> namen
['jan', 'joris', 'kees', 'koen', 'pieter', 'rob']
>>> namen.append('aniek')
>>> namen
['jan', 'joris', 'kees', 'koen', 'pieter', 'rob', 'aniek']
```

Elementen op een specifieke positie (index) tussen de andere elementen voegen – insert()

Via de insert() functie kun je elementen toevoegen op een specifieke plaats in de list.

```
>>> namen.insert(3, 'mireille')
>>> namen
['jan', 'joris', 'kees', 'mireille', 'koen', 'pieter', 'rob', 'aniek']
```

Elementen met een bepaalde waarde verwijderen – remove()

Je kunt ook bestaande elementen verwijderen. Zo verwijder je met de functie remove(x) het eerste element waarvan de waarde gelijk is aan x:

```
>>> namen
['jan', 'joris', 'kees', 'mireille', 'koen', 'pieter', 'rob', 'aniek']
>>> namen.remove('pieter')
```

Je krijgt een foutmelding als je vraagt om een element te verwijderen dat niet in de lijst zit.

```
>>> namen
['jan', 'joris', 'kees', 'mireille', 'koen', 'rob', 'aniek']
>>> namen.remove('pieter')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Elementen op een specifieke positie verwijderen – pop()

Je kunt ook een element op een bepaalde positie (index) verwijderen. Dat doe je met de functie pop():

```
>>> namen
['lies', 'jan', 'joris', 'kees', 'mireille', 'koen', 'rob', 'aniek']
>>> namen.pop(2)
'joris'
>>> namen
['lies', 'jan', 'kees', 'mireille', 'koen', 'rob', 'aniek']
```

Als je goed hebt opgelet, zie je dat de functie pop niet alleen een element verwijdert, maar op de opdrachtregel ook als waarde het verwijderde element teruggeeft ('joris').

Opdracht

- Gebruik de list die je hebt aangemaakt in de opdracht van taak 4
- Voeg twee namen toe aan het einde van de lijst (je hebt nu 9 elementen)
- Voeg een naam toe op de tweede positie (je hebt nu 10 elementen)
- Zoek op wat de derde naam in de lijst is en verwijder deze met de functie remove(), (je hebt nu 9 elementen)
- Voeg een naam toe op de vierde positie (je hebt nu 10 elementen)
- Probeer een ValueError te krijgen
- Verwijder het vijfde element uit de lijst (je hebt nu 9 elementen)

Taak 6 – Snijden in een list

Python heeft een krachtige manier om een lijst in stukken te snijden: 'slicing'. Herinner je de notatie [n] voor het n-de element?

De syntax is als volgt: naamList[start:stop:step].
Step laten we in deze taak nog even achterwege.

Met [n:] krijg je de elementen terug vanaf index n, met [:n] de elementen tot index n (niet inbegrepen) en met [m:n] de elementen van index m tot n (die laatste niet inbegrepen). Enkele voorbeelden maken dit duidelijk:

```
>>> namen = {'lies', 'jan', 'kees', 'mireille', 'keen', 'rob'}
>>> namen[1:] //alles vanaf element 1
['jan', 'kees', 'mireille', 'koen', 'rob']
>>> namen[:4] //alles vanaf element 0 tot 3
['lies', 'jan', 'kees', 'mireille']
```

```
>>> namen[1:4] //alles vanaf element 1 tot 3
['jan', 'kees', 'mireille']
```

Omdat Python begint te tellen vanaf 0 en in de beginpositie van een slice het element zelf meerekent, maar in de eindpositie niet, is de notatie van slicing nogal verwarrend. Het helpt daarom om deze posities te beschouwen als de posities van de komma's in de lijst, te tellen vanaf 1. Alles tussen de komma's op die posities is dan de gevraagde slice.

Bekijk het volgende plaatje en vervang de + door een ,:

```
                +---+---+---+---+---+---+
                | P | y | t | h | o | n |
                +---+---+---+---+---+---+
Slice position: 0   1   2   3   4   5   6
Index position:   0   1   2   3   4   5
```

Neem bijvoorbeeld namen[1:4]

	[	'lies'	,	'jan'	,	'kees'	,	'mireille'	,	'koen'	,	'rob'	]
index		0		1		2		3		4		5	
slice	0		1		2		3		4		5		6

Omdat namen gelijk is aan ['lies', 'jan', 'kees', 'mireille', 'koen', 'rob'], nemen we alles tussen de eerste en de vierde komma, dus van 'jan' tot en met 'mireille', oftewel ['jan', 'kees', 'mireille'].

	[	'lies'	,	'jan'	,	'kees'	,	'mireille'	,	'koen'	,	'rob'	]
index		0		1		2		3		4		5	
slice	0		1		2		3		4		5		6

Slicing is ook een krachtige manier om een deel van een lijst te veranderen. Zo vervang je eenvoudig voorgaande slice in de lijst door een andere naam:

```
>>> namen
['lies', 'jan', 'kees', 'mireille', 'koen', 'rob']
>>> namen[1:4] = ['bas']
>>> namen
['lies', 'bas', 'koen', 'rob']
```

Opdracht

- Gebruik de list die je hebt aangemaakt in de opdracht van taak 6
- Gebruik slice om alles tot het tweede element op te vragen
- Gebruik slice om alle elementen op te vragen vanaf element drie
- Gebruik slice om het vierde t/m zesde element op te vragen

Taak 7 -Dictionaries

In de vorige taken hebben we het over een list gehad. Nu gaan we hier de datastructuur dictonary aan toevoegen.

In een list heeft elk element als index zijn positie, zodat je eenvoudig het element op een specifieke positie kunt opvragen. Een andere datastructuur is de 'dictionary', die als index voor zijn elementen een sleutel gebruikt, vaak een string of een getal. Elke sleutel van de dictionary moet uniek zijn, zodat je eenvoudig de waarde die bij een specifieke sleutel hoort, kunt opvragen.
Een voorbeeld maakt duidelijk hoe je met een dictionary werkt:

```
>>> scores = {'lies': 5, 'bas': 2, 'kees': 1, 'aniek': 3}
>>> scores['aniek']
>3
>>> scores['bert']
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'bert'
>>> len(scores)
>4
```

Let bij het bovenstaande op het gebruik van accolades '{' en '}'.

Op deze manier kun je eenvoudig de score van een persoon opvragen op basis van zijn of haar naam. Je ziet hier ook dat je een foutmelding krijgt als je een element opvraagt met een index die niet in de dictionary bestaat.

Een dictionary aanpassen

Een dictionary kun je net zoals een list veranderen. Je kunt de waarde bij een specifieke sleutel veranderen, maar je kunt even eenvoudig een nieuw element toevoegen: ken gewoon een waarde toe aan een nieuwe sleutel. Bijvoorbeeld:

```
>>> scores
>{'lies': 5, 'bas': 2, 'kees': 1, 'aniek': 3}
>>> scores['lies'] += 1
>>> scores
> {'lies': 6, 'bas': 2, 'kees': 1, 'aniek': 3}
>>> scores['bert'] = 1
>>> scores
>{'lies': 6, 'bas': 2, 'kees': 1, 'aniek': 3, 'bert': 1}
```

Een sleutel en de bijbehorende waarde uit de dictionary verwijderen

Dit doe je met het speciale keyword del:

```
>>> scores
>>> del scores['kees']
>>> scores
{'lies': 6, 'bas': 2, 'aniek': 3, 'bert': 1}
```

Opdracht

- Maak een dictionary aan met 5 studenten aan en voeg voor elke student een score toe {'naam':score}
- Verwijder de eerste student met del

Taak 8 - Samenvatting

In dit deel zijn we lang stil blijven staan bij een van de meest gebruikte datastructuren in Python: de list. De kennis die je hebt opgedaan over lijsten, kun je voor heel wat andere datatypes in Python hergebruiken. Zo toonden we hoe de notatie voor een index en voor 'slicing' hetzelfde is bij een string. Een ander belangrijk datatype dat je in dit deel zag, is de dictionary, waarin je geen positie, maar een sleutel als index gebruikt.

Bekijk het volgende:

Datastructuur: een datatype dat uit elementen bestaat die met elkaar samenhangen.
List: een datastructuur waarin je elementen opvraagt aan de hand van hun positie.
Index (in een list): positie van een element in een lijst, te beginnen vanaf 0.
Dictionary: een datastructuur waarin je elementen opvraagt aan de hand van een unieke sleutel.
Index (in een dictionary): sleutel van een element waarmee het uit een dictionary op te vragen is.

Opdracht

In het volgende deel verlaten we de interactieve Python-sessies en gaan we weer programma's schrijven. De opdrachten kun je dan ook het beste als een programma maken.

1. Maak een lijst met de namen: Erik, Omar, Wim, Ayoub, Mila, Alisha, Milan
- Splits deze lijst in twee nieuwe lijsten(a en b): **a**: alleen het laatste element en **b**: de rest van de lijst.
- Geef deze twee nieuwe lijsten een eigen nieuwe naam. Let daarbij op de naamgeving van deze lijst.
- De naamgeving moet duidelijk maken hoe deze lijst tot stand kwam.

De eerste vraag die je moet stellen is: Welke functie kan ik hier het best voor gebruiken?
Hint: Het is een functie die niet alleen de waarde verwijderd, maar die de verwijderde waarde ook teruggeeft. En, welke (slimme) indexering kan ik hier het best gebruiken?

2. Maak een nieuwe string: bijvoorbeeld: 'nieuwjaarsfeest'.

Extract uit een string alle tekens, behalve het eerste en het laatste teken en stop die tekens in een nieuwe string.

Ook hier stel je jezelf een vraag: met welke indexering adresseer ik die tekens het makkelijkst? Je weet dat er indexeringen zijn die vanaf een bepaalde positie werken en andere die tot een bepaalde positie werken, onafhankelijk van de lengte van de string.

3. Maak gebruik van de namen uit opdracht 1 (hierboven).
- Maak een datastructuur waarbij elke naam gekoppeld is aan een geboortejaar.
- Vervolgens maak je iedereen een jaartje ouder in deze datastructuur met een programmaatje.
- Geef die nieuwe structuur weer op het scherm.

De vragen die je jezelf moet stellen: Welke datastructuur ga ik hiervoor gebruiken? Hoe kan ik die geboortejaren veranderen zodat iedereen een jaartje ouder lijkt?