Toetsvoorbereiding week 5

Wat je moet kunnen voor de Python toets

- De leerling weet wat operatoren zijn en kan gebruik maken van vergelijkende -, rekenkundige - en logische operatoren.
- De leerling is bekend met het begrip concatenation en kan met behulp van string concatenation woorden/zinnen aan elkaar toevoegen.
- De leerling weet wat een datatype is, kan enkele voorbeelden opnoemen en werken met (verschillende) datatypen.
- De leerling kan de ingebouwde type-functie toepassen om de datatype van bijvoorbeeld een variabele hiermee achterhalen.
- De leerling heeft kennis van variabelen en kan hiermee werken.
- De leerling weet wat een Python for-loop is, is bekend met de syntax en kan deze schrijven.
- De leerling weet wat een if-statement is, is bekend met de syntax en kan deze schrijven.
- De leerling weet wat een break - en continue statement is en kan deze toepassen in een for-loop.
- De leerling weet wat een datastructuur is, kan enkele voorbeelden noemen en kan werken met Python lists en dictionaries.
- De leerling is bekend met de volgende ingebouwde Python functies:
  - print()
  - len()
  - range()

De bovenstaande punten worden in de volgende secties toegelicht en uitgewerkt.

> **LET OP! Deze pagina bevat informatie voor de toets in week 5. Echter, van de leerling wordt ook verwacht dat hij/zij in staat is om bijvoorbeeld for-loops te schrijven die een list of dictionary loopen. Hierbij is het belangrijk om te weten hoe je 1) Loopt door een list en 2) Loopt door een dictionary.**
>
> **Van de leerling wordt verwacht dat hij zijn de key-value pairs, keys en values kan**

loopen wanneer het om een dictionary gaat.

Het loopen van lists/dictionaries is in de les gedemonstreerd. Voorbeelden zijn ook te vinden in de uitwerkingen van de toetsstof van week 4.

Operatoren (operators)

de leerling weet wat operatoren zijn en kan gebruik maken van vergelijkende -, rekenkundige - en logische operatoren.

Vergelijkende operatoren

Vergelijkende operatoren resulteren altijd in een Boolean waarde.

Syntax	Beschrijving	Voorbeeld
<	Kleiner dan	5 < 7 = True
<=	Kleiner dan of gelijk aan	5 <= 5 = True
>	Groter dan	7 > 5 = True
>=	Groter dan of gelijk aan	7 >= 5 = True
==	Gelijk aan (gebruikt voor een vergelijking. Niet hetzelfde als =)	7 == 5 = False
!=	Niet gelijk aan	7 != 5 = True

Rekenkundige operatoren

Rekenkundige operatoren worden gebruikt voor calculaties met numerieke waarden.

Syntax	Beschrijving	Voorbeeld
+	Plus	4 + 2 = 6
-	Min	4 - 2 = 2
*	Keer (vermenigvuldiging)	4 * 2 = 8
/	Delen	4 / 2 = 2

%	Modulus (rest van een deelsom)	4 % 2 = 0 (geen rest, want 4 is een even getal) 7 % 2 = 1 (rest 1, want 7 is een oneven getal)
**	Exponent (herhaaldelijk vermenigvuldigen)	4 ** 2 = 16 (4 ^2 = 4 * 4 = 16)
//	Geeft een geheel getal na deling (afgerond naar beneden). Bijvoorbeeld 4.5 = 4.	7 // 2 = 3 (7/2 is 3.5, maar een dubbele forward-slash zorgt voor een afronding naar beneden)

Logische operatoren

Logische operatoren resulteren in een Boolean waarde.

Syntax	Beschrijving	Voorbeeld
or	Of	True or False = True
and	En	True and False = False
not	Niet	not True = False

De leerling is bekend met het begrip concatenation en kan met behulp van string concatenation woorden aan elkaar toevoegen.

String concatenation

Een datatype is een bepaalde type data, bijvoorbeeld tekst. In Python - en andere programmeertalen - wordt dit String genoemd.
Text zullen we voortaan beschrijven als een string, en een string wordt aangeduid door enkele - of dubbele aanhalingstekens. Data van het type string kun je samenvoegen. Over het algemeen gebruik je hier een plus teken voor, bijvoorbeeld:

```
>>> 'hello '+'world'
'hello world'
```

Dit samenvoegen wordt in het Engels *concatenation* genoemd. Je kunt dit proces dus beschrijven als het samenvoegen van twee or meer aparte stukken data van het type string.

Datatype

Zoals het woord *datatype* aangeeft, gaat het bij datatypes om het type gegeven. Voorbeelden van datatypen zijn:

Datatype	Python (syntax)
Teskt type	str
Numeriek type	int, float
Reekstype	list, range
Map type	dict
Boolean type	bool

Uiteraard zijn er meer datatypen. De bovenstaande datatypen hebben we in de les behandeld.

Let op! List en dict (dictionary) zijn datastructuren, maar behoren in Python ook tot datatypen omdat je o.a. met de type-functie kan achterhalen tot welke datatype class ze behoren.

De type-functie is een ingebouwde Python functie en wordt gebruikt om een datatype van een bepaald gegeven te achterhalen. Hieronder zie je enkele voorbeelden:

```
>>> type([])
<class 'list'>
>>> type('hello world')
<class 'str'>
>>> type(1)
<class 'int'>
>>> type(True)
<class 'bool'>
>>> type({})
<class 'dict'>
>>> type(.5)
<class 'float'>
```

Variabelen

Met behulp van variabelen kun je data uit het computergeheugen ophalen en wijzigen. Python is, in vergelijking tot bijvoorbeeld Java, er makkelijk met het aanmaken van variabelen. In Python hoef je bijvoorbeeld geen datatype aan te geven wanneer je een variabele declareert. Bijvoorbeeld:

```
x = "ik ben een variebele


y = 5
```

```
variebel_in_python = "In Python wordt ieder woord gescheiden met een underscore(_)"


mijn_mooie_list = []
```

For-loop

Een for-loop kun je het beste gebruiken wanneer je een bepaalde handeling herhaaldelijk wilt uitvoeren. De logica werkt als volgt:

- Voer de volgende handeling(en) uit voor ieder getal binnen een bepaald interval.

of:

- Voer de volgende handeling(en) uit voor ieder element in een list.

range()

Het volgende voorbeeld demonstreert de for-loop aan de hand van het interval 0 tot en met 9. Dit interval zetten we vast met behulp van de range functie, dit is een ingebouwde Python functie. De syntax voor de range functie is: `range()`. Omdat we over een functie spreken, is het noodzakelijk dat we de haakjes gebruiken om deze aan te roepen. Tussen de haakjes zetten we het getal, bijvoorbeeld 6: `range(6)`. Dit zal ervoor zorgen dat we straks loopen, startend bij 0 tot en met 6 - 1 = 5.

Syntax

De for-loop heeft altijd dezelfde opbouw. Het begrip met het woordje *for* gevolgd door één of meerdere variabelen. Vervolgens komt het woordje *in*, gevolgd door hetgeen waar we doorheen willen itereren. Denk hierbij een een interval, een list, een dictionary enzovoort.

Voor dit voorbeeld werken we, zoals eerder vermeldt, met een range functie met de waarde 6 en loopen we door ieder getal binnen het interval 0 tot en met 5. Tijdens iedere iteratie printen we het getal:

```
>>> for i in range(6):
...     print(i)
...
0
1
2
3
4
```

```
5
```

Let op! De i na het woordje *for* is de naam van mijn variabele. Het had evengoed een andere naam kunnen hebben.

If-statement

Wanneer een if-statement binnen, bijvoorbeeld een for-loop staat, is het mogelijk de break - of continue statement te gebruiken. Met behulp van deze statements kun je een loop laten stoppen of ervoor zorgen dat deze blijft itereren zonder een opdracht uit te voeren. Beide statements worden aan de hand van een voorbeeld gedemonstreerd.

Break statement

Stel dat we een for-loop hebben die 10 keer looped. Deze for-loop stoppen we zodra onze variabele *i* de waarde 7 heeft. Dit doen we met behulp van de break statement:

```
>>> for i in range(10):
...     if i == 7:
...             break
...     else:
...             print('De waarde van i is ', i)
...
De waarde van i is  0
De waarde van i is  1
De waarde van i is  2
De waarde van i is  3
De waarde van i is  4
De waarde van i is  5
De waarde van i is  6
```

Merk op dat *De waarde van i is 7* niet is geprint. Dit komt omdat de *break* statement ervoor zorgt dat we uit de for-loop stappen.

Continue statement

De continue statement wordt gebruikt om delen van de code binnen een iteratie over te slaan. We zouden deze statement bijvoorbeeld kunnen gebruiken in een for-loop die alle even getallen print en geen code uitvoert als het getal oneven is. Bijvoorbeeld:

```
>>> for i in range(10):
...     if i % 2 == 0:
```

```
...             print(i, ' is een even getal')
...     else:
...             continue
...
0  is een even getal
2  is een even getal
4  is een even getal
6  is een even getal
8  is een even getal
```

Merk op dat de print is overgeslagen voor alle oneven getallen.

Datastructuur

Een datastructuur is een methode/manier om data op te slaan. Deze sectie behandeld de volgende datatstructuren:

- Lists
- Dictionaries

List

Een list is een datastructuur die wordt aangeduid door rechte haken (`[]`). Lists kunnen items bevatten. Deze worden door komma's van elkaar gescheiden. List items kunnen bestaan uit String(s), Integer(s) of andere list(s). De items binnen een list zijn gealloceerd aan een index, deze start bij 0.

Dictionary

Een dictionary is een datastructuur die word aangeduid door accolades (`{}`). Dictionary items bestaan uit paren, namelijk een key-value pair hebt. Deze worden van elkaar gescheiden middels een dubbele punt (`:`). Hieronder is een voorbeeld van een lege dictionary en een dictionary met key-value pairs:

```
>>> {} # lege dictionary
{}
>>> {1: 'hello', 2: 'Welcome', 3: 'to', 4: 'Python'} # dictionary met key-value pair
{1: 'hello', 2: 'Welcome', 3: 'to', 4: 'Python'}
```

Ingebouwde Python functies

Zoals alle programmeertalen, heeft ook Python ingebouwde functies. Enkele functies die je moet kennen voor de toets zijn:

- print()
- range()
- len()

Deze functies worden in de volgende secties uitgelegd en gedemonstreerd.

print()

De print-functie is een krachtige functie die ingezet kan worden door beginnende programmeurs, zodat het duidelijk wordt wat de waarde van een gegeven is. Het is echter ook een tool om je logs bij te houden aan de serverside.

Print() is een functie en functies moeten aangeroepen worden met haakjes. Tussen de haakjes komt hetgeen te staan wat je wil printen, bijvoorbeeld;

```
>>> print("Hello world")
Hello world
```

range()

De range-functie wordt gebruikt voor loops en is een functie die je op drie manieren kunt gebruiken. Het is mogelijk om de functie minimaal 1 - en maximaal 3 waarden te geven.

range() met 1 waarde

Wanneer je 1 waarde meegeeft aan de range functie begin je te itereren bij 0 tot het ingevoerde getal-1. Stel als we `range(10)` gebruiken, dan zal de loop beginnen te tellen bij 0 tot en met 9:

```
>>> for i in range(10):
...     print('i is ',i)
...
i is  0
i is  1
i is  2
i is  3
i is  4
i is  5
i is  6
i is  7
```

```
i is  8
i is  9
```

range() met 2 waarden

De range functie accepteert ook twee waarden, waarbij te ingevoerde getallen dienen als een interval. Stel als we `range(2, 12)` gebruiken, dan zal de loop beginnen te tellen bij 2 en tellen tot en met 12-1=11:

```
>>> for i in range(2, 12):
...     print('i is ',i)
...
i is  2
i is  3
i is  4
i is  5
i is  6
i is  7
i is  8
i is  9
i is  10
i is  11
```

range() met 3 waarden

Door een derde waarde mee te geven aan de range functie, kun je de toenamefactor aangeven, `range(0, 20, 2)` zal dus beginnen met het tellen vanaf 0 tot en met 20-1=19 in stappen van 2:

```
>>> for i in range(0, 20, 2):
...     print('i is ', i)
...
i is  0
i is  2
i is  4
i is  6
i is  8
i is  10
i is  12
i is  14
i is  16
i is  18
```

len()

De length functie is een ingebouwde Python functie die je kunt gebruiken om de inhoud van een list en/of een dictionary te achterhalen. Deze functie geeft altijd de hoeveelheid items in een list/dictionary.

len() met lists

Wanneer je de length functie gebruikt om de items uit een list te tellen, resulteert dit altijd in het aantal elementen dat zich in een list bevindt. Als een list leeg is, krijg je een 0 terug. Als een list waarden bevat, dan krijg je de hoeveelheid elementen als cijfer terug. Bijvoorbeeld:

```
>>>len([])
0


>>> a = [1, 2, 3, 4]
>>> len(a)
4
```

len() met dictionary

In de vorige sectie hebben we gezien hoe de length functie toegepast kan worden om het aantal elementen in een list te achterhalen. Deze functie werkt exact hetzelfde bij dictionaries. Bij een lege dictionary krijg je een length 0 terug. Anders het aantal key-value pairs. Bijvoorbeeld:

```
>>> len({})
0


>>> d = {1: 'hello', 2: 'there', 3: 'you'}
>>> len(d)
3
```

Revision #8
Created 2020-03-07 06:59:13 UTC
Updated 2020-03-07 19:53:31 UTC