Python 2

DEEL 5, FORMATTING

In Python kun je de in- en uitvoer heel precies formatteren zoals je die hebben wilt. Het zal duidelijk zijn dat een simpel print() statement niet altijd voldoende is. In dit deel gaan we die uitvoer aanpassen èn we duiken dieper in lijsten, dictionaries en tuples.

Taak 1 - De uitvoer van print aanpassen

Tot dusver gebruikten we deze opdracht om een getal uit een dictionary op het scherm te tonen: Ga nog eens naar de vorige opdracht waarin je de dictionary "getallen" gebruikte en als index het cijfer (de hieraan gekoppelde waarde was dan dit cijfer uitgeschreven in tekst).

```
getallen = {1: 'een', 2: 'twee', 3: 'drie', 4: 'vier', 5:
'vijf', 6: 'zes', 7: 'zeven', 8: 'acht', 9: 'negen', 10: 'tien'}
getal = int (input ("Voer een geheel getal van 1 tot en met 10
in:"))
if getal < 1:
    print("Het getal is kleiner dan 1.")
elif getal > 10:
    print("Het getal is groter dan 10.")
else:
    print(getallen[getal])
```

Het print statement zag er zó uit:

```
    print(getallen[getal])
```

Maar wat als je meerdere objecten op dezelfde regel wilt tonen? Dat kan eenvoudig door meerdere objecten aan de functie print door te geven, gescheiden door een komma. Verander het in:

```
    print (getal, "is gelijk aan", getallen [getal])
```

Opdracht: Maak de verandering en voer deze code uit.

Zoals jullie inmiddels gezien hebben (ondermeer bij de eindopdracht van Python Basic), worden alle objecten die je aan print doorgeeft, gescheiden door een spatie getoond. Waar een komma staat, voegt Python een spatie toe. Maar je kunt dat print statement beïnvloeden.

Na de objecten die je aan de functie print doorgeeft, kun je nog argumenten toevoegen die de uitvoer van print veranderen, namelijk `sep` en `end`. Met `sep` (afkorting van separator) verander je de spatie tussen objecten door een andere afscheiding, en met `end` vervang je het einde van de uitvoer (standaard '\n' waarmee je een newline of een nieuwe regel aanduidt) door iets anders. Bijvoorbeeld:

```
>>> print(1, getallen[1], sep = ':', end = '\n\n')
1:één


```

Let op: bij `sep` (hierboven) staan twee single quotes met de dubbele punt ertussen. Maar het werkt ook met double quotes: " uiteraard ook met die dubbele punt. Dan wordt het:

```
>>> print(1, getallen[1], sep = ":", end = "\n\n")
1:één


```

Het gaat mis wanneer je " en ' door elkaar gebruikt: `sep = '"` Dit gaat niet goed.
Door aan het argument end twee newlines toe te kennen, komt er een lege regel na de uitvoer.

Opdracht 1:

Je kunt bij `sep` en `end` elke string neerzetten die je wilt. Dus je kunt iets maken dat een uitvoer geeft als :

```
3 is voor diegenen die niet weten hoe ze dat moeten spellen:
drie. Is dat niet gemakkelijk?
```

Probeer dat maar eens door de string tussen quotes bij `sep` en `end` aan te passen.

Taak 2 – de .format() method.

LET OP: Dit is één van de nieuwere Python string format technieken. Hij wordt dus niet ondersteund in oudere Python code. Met jullie versie 3.8.x zitten jullie goed, maar op het internet vind je nog veel syntax die ouder is.

Als je meerdere objecten in één uitvoer met print wilt opnemen en/of complexere zaken aan de uitvoer wilt veranderen, komt de methode `format` van pas. In de eenvoudigste vorm pas je

die functie toe op een string waarin je met {} plaats vrijhoudt. Het argument dat je aan de functie doorgeeft, komt op de plaats van die {}. Typ dit eens in in de shell van Thonny:

```
>>> 'Je hebt {} gekozen'.format(1)
>'Je hebt 1 gekozen.'
```

Valt het je op dat we hier geen print() statement intypen? Het resultaat is dan ook iets anders dan je gewend bent van print(). Kijk maar eens goed: de quotes staan er nog in.
Vraag: Kun je uitleggen wat hier gebeurt?

Dat werkt ook met meerdere placeholders:

```
>>> 'Je hebt {} gekozen tussen de getallen {} en
{}'.format(2,1,10)
'Je hebt 2 gekozen tussen de getallen 1 en 10.'
```

Je kunt de plaatshouders ook een volgnummer geven, zodat ze in een andere volgorde in de uitvoer komen dan de argumenten die je aan format doorgeeft.
Let op: Ook hier begint de telling weer bij 0. Het eerste element is dus {0} in het . format. Dit werkt dus nèt zo als de indexering bij lists en dictionaries.

```
>>> '{1} <= {0} <= {2}'.format(2,1,10)
>'1<=2<=10'
```

Maar vaak is het duidelijker als je de argumenten een naam geeft en die namen ook in de plaatshouders gebruikt:

```
>>>> '{min} <= {getal} <= {max}'.format(min=1, max=10, getal=2)
>'1<=2<=10'
```

Om de leesbaarheid van jouw code te bevorderen, heeft deze formulering de voorkeur. Als je bijvoorbeeld in een team werkt waarbij ook anderen jouw code lezen, gebruik je dit. Trouwens, het helpt ook jezelf; wanneer je na enkele maanden (of zelfs weken) weer eens naar je oude code kijkt, want tegen die tijd ben je alweer vergeten wat je precies bedoelde.

Taak 3 - Door een lijst lopen met for

Vaak wil je niet één regel uitvoer tonen met print, maar meerdere, bijvoorbeeld voor alle elementen in een lijst of dictionary. We hebben dus een manier nodig om al die elementen af te gaan. Dat kan met een for-lus, zoals jullie al eerder hebben gedaan:

```
namen = ['lies', 'jan', 'kees1, 'mireille', 'koen', 'rob']
for naam in namen:
    print(naam)
```

Python gaat elk element uit de lijst met namen af en toont dit element met print. Als je zowel de index als de inhoud van elk element wilt tonen, maak je gebruik van de handige functie enumerate:

```
namen = ['lies', 'jan', 'kees', 'mireille', 'koen', 'rob']
```

```
for index, naam in enumerate(namen):
    print('{}: {}'.format(index, naam))
```

Opdracht 2:

Druk de hele lijst af met gebruik van .format, zoals hierboven.
Elke regel in de lijst moet er zó uit zien: (**let op**, in de print begint de lijst met 1. Daar moet je dus iets voor bedenken.)

Nummer 1 in de lijst is: lies
Nummer 2 in de lijst is: jan
… etc …

Taak 4 - Door een dictionary lopen met for

We kunnen met een for-lus ook door alle elementen van een dictionary lopen. Afhankelijk van waarin je geïnteresseerd bent, kan dat op drie manieren. We tonen in het volgende programma achtereenvolgens hoe je door alleen de sleutels gaat, door alleen de waardes en door de sleutels met hun bijbehorende waardes. **Neem de code over en voer 'm uit.** Dan zie je wat er gebeurt.

```
scores = {'lies': 6231, 'bas': 4322, 'kees': 6218, 'mireille':
4821, 'aniek': 6435, 'bert': 184, 'genius':10231}
#print('Spelers:')
# Methode 1: Alleen de sleutels worden afgedrukt
for speler in scores:
    print(speler)
    print()
print('Scores:')
# Methode 2: Alleen de waardes worden afgedrukt
for score in scores.values():
    print('Spelers met score')
    print(score)
    print()
# Methode 3: Zowel de sleutels als hun waardes worden afgedrukt
for speler, score in scores.items():
    print('{}: {}'.format(speler, score))
```

Deze code bevat twee keer dezelfde fout, waardoor de uitvoer er niet zo uit ziet als hieronder.

Opdracht: Spoor de fout op en verbeter die zodat de uitvoer eruit ziet als hieronder

```
Spelers:
lies
```

```
bas
kees
mireille
aniek
bert
genius

Scores:
Spelers met score
6231
Spelers met score
4322
Spelers met score
6218
Spelers met score
4821
Spelers met score
6435
Spelers met score
184
Spelers met score
10231

lies: 6231
bas: 4322
kees: 6218
mireille: 4821
aniek: 6435
bert: 184
genius: 10231
```

De sleutels van een dictionary kun je ook opvragen met de functie keys, maar dat is hier niet nodig. Bijvoorbeeld: `scores.keys()`

NB: Let op methode 3 (hierboven) waarin meer dan één variabele in de for loop genoemd is. In de for loop krijgt elk van die variabelen de waarde die in die dictionary staat.

Vraag: Hoe zou je dat doen wanneer je 4 gegevens uit een dictionary wil lezen?

Taak 5 - **List comprehension**

Facultatief: voor enthousiastelingen

For loops zijn handig, maar voor complexere bewerkingen wat omslachtig. Stel dat we de lengte van de langste naam uit een lijst willen berekenen. Met een for loop kunnen we dat als volgt doen:

```
namen = ['lies', 'jan', 'kees', 'mireille', 'koen', 'rob']
max_lengte = 0
for naam in namen:
    lengte = len(naam)
    if lengte > max_lengte:
        max_lengte = lengte
    print(max_lengte)
```

Dit werkt en is vrij duidelijk, maar het kan beter. Voor dit soort bewerkingen heeft Python een handig concept: list comprehension. Daarmee kunnen we de hele voorgaande berekening vervangen door één regel:

```
namen = ['lies', 'jan', 'kees', 'mireille', 'koen', 'rob']
print(max([len(naam) for naam in namen]))
```

Deze regel zegt: voor elke naam in de list met namen bereken je de lengte van de naam, en daarvan bereken je het maximum. Het stuk [len(naam) for naam in namen] is een beknopte manier waarmee we een lijst aanmaken van de lengtes van alle elementen in de lijst namen.

video: https://youtu.be/1HlyKKiGg-4

Taak 5 - Tekst uitlijnen en opvullen met de methode .format()

Dan gaan we nu weer terug naar de methode format. Daarmee kunnen we tekst uitlijnen, zowel links als rechts en gecentreerd. We kunnen ook lege ruimte opvullen met willekeurige tekens. Een voorbeeld:

```
'{:=^15}'.format('Titel')
'=====Titel====='
```

En met iets als:

```
print('{naam:<6}: {score:0>6}'.format(naam='kees',score=3))
```

... krijgen we als uitvoer:

```
kees : 000003
```

Wat staat er nu tussen de accolades? Dit heet een format template en het heeft een vaste indeling.

- Als we argumenten van format een naam geven, komt die naam vóór de dubbele punt.
- Dan een dubbele punt, die geeft het begin van het format template aan.
- Dan het teken dat we willen toevoegen om de string aan te vullen tot de juiste lengte. Met dit teken vullen we de lege ruimte op.
- Dat kan een "=" zijn, maar ook een "0" zoals in de voorbeelden hierboven.
- Daarna een teken waarmee we de tekst uitlijnen; Met **^** centreren we tekst, met **<** lijnen we tekst links uit en met **>** lijnen we tekst rechts uit.
- Na het uitlijningsteken komt het totaal aantal tekens dat de tekst maximaal mag innemen. Vóór het uitlijningsteken kan eventueel nog een teken komen waarmee we de lege ruimte opvullen.

Opdracht 1:

Gebruik deze dictionary scores:

```
scores = {'lies': 6231, 'bas': 4322, 'kees': 6218, 'mireille': 4821, 'aniek': 6435, 'bert': 184, 'genius':10231}
```

Maak een print van deze lijst die er voor elk element zò uit ziet: ++++lies++++ (in totaal precies 12 tekens en de naam staat in het midden), dan de tekst: "score:" gevolgd door de waarde van de score van die persoon in precies 6 cijfers: in dit geval, 006231.

```
++++lies++++ score: 006231
```

Taak 6 - Tuples.

Wat is een tuple?

Een tuple is een verzameling gegevens, gedefinieerd met ronde haken, waarin de elementen gescheiden zijn door komma's.
Een tuple is onveranderbaar: dus read only. Een tuple werkt sneller dan een list en neemt minder geheugen in beslag. Ga je er iets van merken? In deze voorbeelden niet. Maar met grote datastructuren, zoals een tuple met duizenden elementen, wèl.

Waarom gaan we dan hier aan de gang met tuples? Omdat je ze in de praktijk tegenkomt! En dan iets moet weten van de eigenaardigheden.

Dus:

```
tuple_1 = (1,2,3,5)
```
is een tuple

```
tuple_2 = ("aap","noot", "mies")
```
is een tuple

Je ziet dat een tuple gedefinieerd wordt met ronde haken ().

Herhaling:

Een list definieer je met vierkante haken []:

```
list_1 = ["Trevor", "George", "Bob"]
```

En een dictionary met accolades {}:

```
dict_1 = {"Trevor": 18, "George" : 24, "Bob" : 17}
```

Een tuple is **onveranderbaar**. Wil je er iets aan veranderen, dan moet je een nieuwe maken.
Als je een tuple wilt sorteren dan werkt dat niet met `tuple_1.sort`.

Maar je kunt wel een nieuwe gesorteerde versie maken:

```
s_tuple_1 = sorted(tuple_1)
```

Een tuple hoeft overigens niet hetzelfde datatype te bevatten. Een voorbeeld:

```
Student_1 = ("Trevor", 18)
Student_2 = ("George", 24)
Student_3 = ("Bob", 17)
```

En je kunt meerdere gegevens opnemen:

```
Student_1 = ("Trevor", 18, "Amstelveen")
Student_2 = ("George", 24, "Amsterdam")
Student_3 = ("Bob", 17, "Aalsmeer")
Student_4 = ("Alice",18, "Amsterdam")
```

Stel je voor dat je een aantal tuples wilt maken waarin:
(Voornaam, man of vrouw, leeftijd, woonplaats) staan.

Opdracht 2:

- Maak 6 van dit soort tuples aan. *(gaap, simpel ….)*

Maar stel je nu voor dat je een hele klas hebt met dit soort tuples. Kan dat niet handiger?
Ja, dat kan! (maar het is nog steeds veel typwerk). Je kunt tuples in een tuple stoppen …

```
Klas3C = (("Trevor", 18, "Amstelveen"),("George", 24,\
"Amsterdam"),("Bob", 17, "Aalsmeer"),("Alice",18, "Amsterdam"))
```

NB: De backslash "\" geeft aan dat het statement op de volgende regel doorloopt. Typ je het hele statement op één regel dan is die backslash overbodig.

- Maak 6 van dergelijke tuples met dit soort gegevens en groepeer ze in een klas. (verzin namen, of gebruik namen die je kent). Print vervolgens alle studenten uit die klas uit, waarbij je gebruik maakt van .format. De uitvoer moet er zo uitzien:
- Voor een jongen: Zijn naam is Trevor, hij is 18 jaar en woont in Amstelveen.
- Voor een meisje: Haar naam is Alice, zij is 19 jaar en woont in Amsterdam
  Je begrijpt dat je de waarden voor de naam, de leeftijd en woonplaats uit de tuple leest
  En, je moet onderscheid maken tussen jongens en meisjes.

Vragen die je jezelf moet stellen: Hoe lees ik alle gegevens van één tuple binnen de lijst van tuples uit? Hoe maak ik onderscheid tussen jongens en meisjes? Hoe print ik die gegevens dan naar het scherm op de voorgeschreven manier?

Opdracht 3:

Maak nu een variant waarin je sorteert op leeftijd. ***Dit sorteren moet je uitzoeken***.

De aanvullende vraag die je aan jezelf kunt stellen: Een tuple is onveranderbaar. Hoe sorteer ik deze tuple dan toch?

Samenvatting:

In dit deel heb je gezien hoe je de uitvoer van print kunt aanpassen. We hebben de tekst uitgelijnd en opvultekens gebruikt. Ook heb je gezien hoe je meerdere gegevens uit elementen van een list, dictionary of tuple leest in een for- loop. Met list comprehension kun je met heel compacte code een berekening uitvoeren op een list met meerdere elementen en met de uitkomst van die berekening maak je een nieuwe list aan. In het volgende hoofdstuk gaan we verder met in- en uitvoer, maar dan met bestanden.