Diffy

Algemene info

Itereren over posities en waarden van sequentietypes

De ingebouwde functie `enumerate` kan gebruikt worden om een een iterator op te vragen van een sequentietype (*sequence type*: strings, lijsten, tuples, bestanden, ...), die zowel de positie als de waarde van het volgende element van het type teruggeeft. Onderstaand voorbeeld geeft bijvoorbeeld aan hoe je tegelijkertijd de posities en de karakters op die posities kunt overlopen voor een string.

```
>>> for index, karakter in enumerate('abc'):
...     print("index: {}".format(index))
...     print("karakter: {}".format(karakter))
...
index: 0
karakter: a
index: 1
karakter: b
index: 2
karakter: c
```

Je kan dit bijvoorbeeld gebruiken als je synchroon de karakters van twee strings wil overlopen.

```
>>> eerste = 'abc'
>>> tweede = 'def'
>>> for index, karakter in enumerate(eerste):
...    print("{}-{}".format(karakter, tweede[index]))
...
a-d
b-e
c-f
```

In dat geval is het echter aangewezen om gebruik te maken van de ingebouwde functie `zip`, die net haar bestaansrecht haal uit het feit dat ze een iterator teruggeeft voor twee of meer iterateerbare objecten.

```
>>> eerste = 'abc'
>>> tweede = 'def'
>>> for karakter1, karakter2 in zip(eerste, tweede):
...    print("{}-{}".format(karakter1, karakter2))
...
a-d
b-e
c-f
```

Opmerkingen

Tuples met één enkel element

Een tuple met één enkel element ziet er als volgt uit:

```
1,
```

Het essentiële onderdeel van deze notatie is de komma op het einde. Zoals bij elke expressie kan je optioneel ronde haakjes toevoegen, waardoor een tuple met één enkel element ook als volgt kan genoteerd worden:

```
(1, )
```

Maar het is de komma, niet de ronde haakjes, die zorgt voor de definitie van een tuple. Bekijk bijvoorbeeld het onderstaande voorbeeld:

```
>>> reeks = (3)             # een integer
>>> reeks
3
>>> isinstance(reeks, tuple)
False
>>> isinstance(reeks, int)
True
>>> reeks = (3, )           # een tuple
>>> isinstance(reeks, tuple)
True
```

De komma is noodzakelijk, omdat Python de notatie `(object)` anders interpreteert als het object zelf.

Elementen van een tuple herhalen

Net zoals strings kun je ook tuples vermenigvuldigen met een natuurlijk getal. Hierdoor wordt een nieuw tuple aangemaakt, dat een herhaling bevat van de elementen van het oorspronkelijke tuple.

```
>>> tuple = (2, 3)
>>> tuple * 4
(2, 3, 2, 3, 2, 3, 2, 3)
>>> element = (0, )
>>> 6 * element
(0, 0, 0, 0, 0, 0)
```

Een vierkante driehoek

Specifieke info

De driehoek van Pascal wordt in de Python code voorgesteld als een lijst van lijsten, maar de visualisatie uit de opgave zet niet alle element in dezelfde kolom onder elkaar. Het is makkelijker om na te denken over deze opgave als je een voorstelling van de driehoek beschouwt waar dit wel het geval is. Bekijk aandachtig de volgende figuur, waar de visuele voorstelling uit de opgave wordt omgezet naar een voorstelling als een lijst van lijsten.

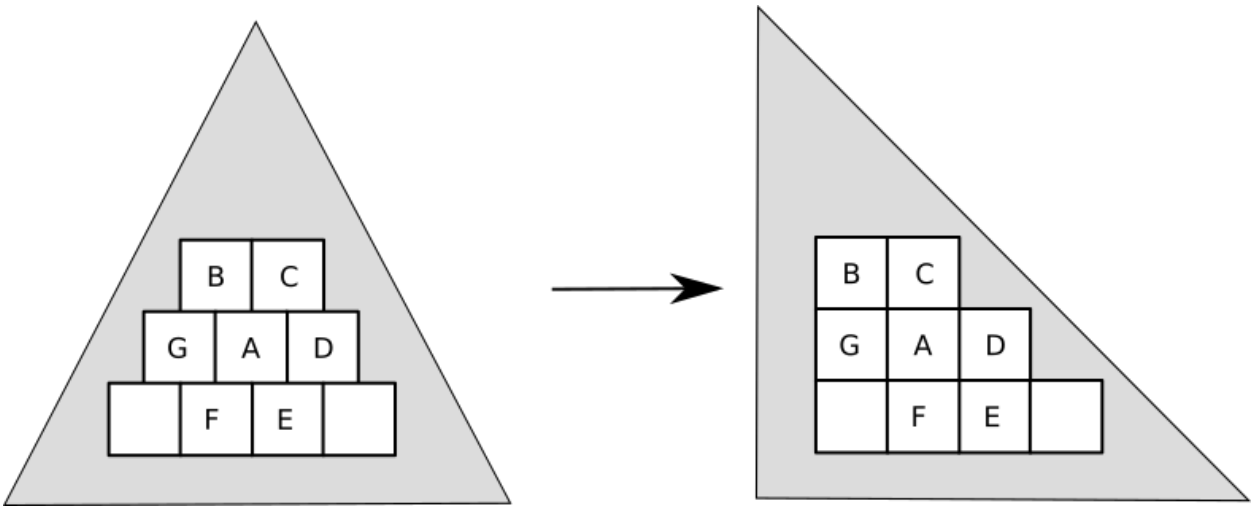

Figure 1: Driehoek van Pascal

Pozo Azul

Algemene info

Geneste lijsten initialiseren

Een rooster met m rijen en n kolommen kan voorgesteld worden door een lijst `l` van m lijsten, waarbij iedere lijst uit `l` n elementen bevat. Stel bijvoorbeeld dat je een rooster van 3 rijen en 2 kolommen wil maken, waarbij elk element uit het rooster een lege string is. Dit kan je doen door gebruik te maken van *list comprehensions*.

```
>>> m = 3
>>> n = 2
>>> rooster = [['' for _ in range(n)] for _ in range(m)]
>>> rooster
[['', ''], ['', ''], ['', '']]
>>> rooster[0][0] = 'A'
>>> rooster
[['A', ''], ['', ''], ['', '']]
```

In veel gevallen kan je ook de vermenigvuldigingsoperator `*` gebruiken om een lijst van een bepaalde grootte aan te maken, waarbij elk element van de lijst hetzelfde is. Maar als dit element mutable is, dan kan dit vreemde resultaten opleveren.

```
>>> m = 3
>>> n = 2
>>> [''] * m
['', '', '']
>>> rooster = [[''] * n] * m
>>> rooster
[['', ''], ['', ''], ['', '']]
>>> rooster[0][0] = 'A'
>>> rooster
[['A', ''], ['A', ''], ['A', '']]
```

Zoals je kan zien, wordt er bij het plaatsen van een `A` op positie `[0][0]` in het rooster ook een `A` geplaatst op posities `[1][0]` en `[2][0]`. De reden hiervoor is dat de vermenigvuldigingsoperator `*` aliassen maakt van de lijst, waardoor zowel `rooster[0]`, `rooster[1]` als `rooster[2]` verwijzen naar hetzelfde lijstobject. Als je dus één lijst aanpast, pas je ze allemaal aan. Dit kan je goed zien als je gebruik maakt van de Python Tutor.

Specifieke info

Bij deze opgave zal het zeker helpen als je twee extra functies schrijft die elk een windrichting als argument meekrijgen. De eerste functie geeft de tegenovergestelde windrichting terug van de gegeven windrichting. De tweede functie geeft twee gehele getallen terug die respectievelijk de verandering in rij en kolom voorstellen als je naar het naburige vierkant van de grot beweegt in de aangegeven windrichting.

Voorts is het zo dat de gang op twee twee mogelijke manier kan vastlopen in de grot:

- de gang in het volgende vierkant is niet verbonden met de windrichting waar je het vierkant binnenkomt; als je een vierkant langs het westen betreedt, dan moet de gang in dat vierkant verbonden zijn met de oostkant van het vierkant
- de gang het huidige vierkant botst tegen de rand van de grot, waardoor je buiten de grot zou belanden als je verder zou stappen