Hoofdstuk 11

Tuples

Een tuple is een groep van één of meer waardes die als een geheel behandeld worden. Dit hoofdstuk legt uit hoe je tuples kunt herkennen en gebruiken.[15]

11.1 Gebruik van tuples

Een tuple bestaat uit een aantal waardes die van elkaar gescheiden zijn met komma's. Meestal worden tuples geschreven met haakjes eromheen, maar de haakjes zijn niet noodzakelijk (behalve in omstandigheden waar anders verwarring zou ontstaan). Bijvoorbeeld:

```
t1 = ("appel", "mango")
print( type( t1 ) )
t2 = "banaan", "kers"
print( type( t2 ) )
```

Je kunt in een tuple verschillende data types mixen.

```
t1 = ("appel", 3, 1.4)
t2 = ("appel", 3, 1.4, ("banaan", 5))
```

Je kunt de **len()** functie gebruiken om te bepalen hoeveel elementen een tuple heeft.

```
t1 = ("appel", "mango")
t2 = ("appel", 3, 1.4)
t3 = ("appel", 3, 1.4, ("banaan", 5))
print( len( t1 ) )
print( len( t2 ) )
print( len( t3 ) )
```

[15] In het Nederlands kun je "tuple" vertalen als "tupel." Je spreekt de twee woorden op dezelfde manier uit. Ik gebruik de Engelse schrijfwijze omdat tuple een data type is.

Merk op dat in dit voorbeeld de lengte van `t3` 4 is, en niet 5. Het laatste element van `t3` is de tuple `("banaan", 5)`, wat telt als één element.

Je kunt een **`for`** loop gebruiken om de elementen van een tuple te doorlopen.

```
t1 = ("appel", 3, 1.4, ("banaan", 5))
for element in t1:
    print( element )
```

Je kunt de **`max`**`()` en **`min`**`()` functies gebruiken om het maximum respectievelijk het minimum te bepalen van een tuple die bestaat uit getallen. Je kunt de elementen van een tuple met numerieke elementen bij elkaar optellen middels de **`sum`**`()` functie.

```
t1 = (327, 419, 101, 667, 925, 225)
print( max( t1 ) )
print( min( t1 ) )
print( sum( t1 ) )
```

Je kunt testen of een element onderdeel van een tuple is met behulp van de **`in`** operator.

```
t1 = ("appel", "banaan", "kers")
print( "banaan" in t1 )
print( "mango" in t1 )
```

11.1.1 Tuple assignments

Je kunt een tuple creëren door een assignment van een aantal waardes gescheiden door komma's aan een variabele. Haakjes zijn optioneel. Wat als je een tuple wilt creëren met slechts één element?

```
t1 = ("appel")
print( type( t1 ) )
```

Als je deze code uitvoert, zie je dat `t1` de class `str` heeft, dat wil zeggen, `t1` is een string. Dat er haakjes omheen staan bij de assignment maakt het niet tot een tuple. Python heeft een truukje om een tuple met slechts één element te maken, namelijk door een komma te plaatsen achter het ene element. Dit is niet erg intuïtief en ik zou het zelfs wat zwak willen noemen, maar historisch was dit de oplossing die een vroege versie van Python bevatte, en kennelijk heeft niemand iets beters weten te verzinnen.

```
t1 = ("appel",)
print( type( t1 ) )
print( len( t1 ) )
```

Python staat toe een tuple van variabelen links van de assignment operator te plaatsen. Dit is een uitzondering op de regel dat slechts één variabele links van de assignment operator staat. De waardes aan de rechterkant worden één voor één naar de linkerkant gekopieerd, van links naar rechts.

```
t1, t2 = "appel", "banaan"
print( t1 )
print( t2 )
```

Je kunt haakjes om de waardes aan de rechterkant zetten, en je kunt ook haakjes rond de variabelen aan de linkerkant zetten; dat maakt geen verschil.

Als je meer variabelen aan de linkerkant zet dan waardes aan de rechterkant, krijg je een runtime error. Hetzelfde geldt voor minder (met als uitzondering dat je precies één variabele plaatst). Je kunt wel tuples aan de rechterkant plaatsen door haakjes te gebruiken.

```
t1, t2 = ("apple", "banaan"), "kers"
print( t1 )
print( t2 )
```

11.1.2 Tuple indices

Net als bij strings, kun je individuele elementen van een tuple benaderen via indices. Waar bij strings de individuele elementen tekens zijn, zijn het bij tuples waardes. Bijvoorbeeld:

```
t1 = ("appel", "banaan", "kers", "doerian")
print( t1[2] )
```

Je kunt zelfs sub-tuples maken, met dezelfde regels als je hebt voor substrings (als je die niet meer weet, lees dan nog eens hoofdstuk 10). Een sub-tuple is ook weer een tuple. Bijvoorbeeld:

```
t1 = ("appel", "banaan", "kers", "doerian", "mango")
print( t1[1:4] )
```

Omdat tuples indices hebben, kun je als alternatief voor een **for** loop om de elementen van de tuple te doorlopen, gebruik maken van de indices.

listing1101.py

```
t1 = ("appel", "banaan", "kers", "doerian", "mango")
i = 0
while i < len( t1 ):
    print( t1[i] )
    i += 1
```

Opgave Schrijf een **for** loop die alle elementen van een tuple toont, en die ook hun indices toont.

11.1.3 Tuple vergelijkingen

Je kunt twee tuples met elkaar vergelijken met behulp van de reguliere vergelijkingsoperatoren. Deze operatoren vergelijken eerst de eerste elementen van beide tuples. Als ze

verschillend zijn, dan geeft de vergelijking van die twee elementen op basis van de regels die voor hun data types geldt, de gevraagde uitkomst. Als ze gelijk zijn, dan worden de volgende elementen van beide tuples met elkaar vergeleken, etcetera.

listing1102.py

```
t1 = ( "appel", "banaan" )
t2 = ( "appel", "banaan" )
t3 = ( "appel", "kers" )
t4 = ( "appel", "banaan", "kers" )
print( t1 == t2 )
print( t1 < t3 )
print( t1 > t4 )
print( t3 > t4 )
```

11.1.4 Tuple retourwaardes

In hoofdstuk 8 legde ik uit dat functies meerdere waardes kunnen retourneren. Als je zoiets programmeert, dan komt het er in feite op neer dat de functie een tuple retourneert. Om zulke retourwaardes af te handelen, doe je wat hierboven beschreven is voor "tuple assignments.".

11.2 Tuples zijn onveranderbaar

Net als strings, zijn tuples onveranderbaar. Dat wil zeggen dat je geen nieuwe waarde kunt toekennen aan een element van een tuple. Het voorbeeld hieronder veroorzaakt een runtime error als je het uitvoert.

```
t1 = ("appel", "banaan", "kers", "doerian")
t1[0] = "mango" # Runtime error
```

11.3 Toepassingen van tuples

Tuples worden niet vaak in Python code gebruikt (behalve als retourwaardes van functies). Een logische applicatie van tuples zou zijn de afhandeling van waardes die altijd voorkomen in kleine verzamelingen. Echter, object oriëntatie (hoofdstuk 20 en verder) biedt veel technieken en hulpmiddelen om met zulke kleine verzamelingen om te gaan, wat betekent dat programmeurs meestal voor object oriëntatie kiezen als ze dat soort zaken onder handen hebben.

Hier volgt toch een voorbeeld van het gebruik van tuples in een applicatie. Stel je voor dat je een applicatie moet schrijven die geometrische figuren in een 2-dimensionale ruimte bewerkt. Een concept dat je daarbij nodig hebt is een punt: een locatie in de 2D ruimte die geïdentificeerd wordt via twee coördinaten. In plaats van functies te schrijven die steeds een aparte X en een aparte Y coördinaat meekrijgen, kun je specificeren dat coördinaten worden meegegeven als tuples.

listing1103.py

```
from math import sqrt

# Retourneert de afstand tussen twee punten in 2-dimensionale
# ruimte. The punten zijn de parameters van de functie.
# Ieder punt is een tuple met twee numerieke waardes.
def afstand( p1, p2 ):
    return sqrt( (p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 )

punt1 = (1,2)
punt2 = (5,5)
print( "Afstand tussen", punt1, "en", punt2, "is",
    afstand( punt1, punt2 ) )
```

Een voordeel van het gebruik van tuples op deze manier is dat het relatief eenvoudig is om een functie te schrijven die kan omgaan met coördinaten in willekeurige ruimtes.

listing1104.py

```
from math import sqrt

# afstand tussen twee punten in N-dimensionale ruimte.
# De punten hebben dezelfde dimensie, ze zijn allebei tuples
# met numerieke waardes, met dezelfde lengte.
def afstand( p1, p2 ):
    totaal = 0
    for i in range( len( p1 ) ):
        totaal += (p1[i] - p2[i])**2
    return sqrt( totaal )

# 1-dimensionale ruimte
punt1 = (1,)
punt2 = (5,)
print( "1D: afstand tussen", punt1, "en", punt2, "is",
    afstand( punt1, punt2 ) )

# 2-dimensionale ruimte
punt1 = (1,2)
punt2 = (5,5)
print( "2D: afstand tussen", punt1, "en", punt2, "is",
    afstand( punt1, punt2 ) )

# 3-dimensionale ruimte
punt1 = (1,2,4)
punt2 = (5,5,8)
print( "3D: afstand tussen", punt1, "en", punt2, "is",
    afstand( punt1, punt2 ) )
```

Wat je geleerd hebt

In dit hoofdstuk is het volgende besproken:

- Tuples
- Tuple assignments
- Tuple indices
- Onveranderbaarheid van tuples
- Toepassingen van tuples

Opgaves

Opgave 11.1 Een complex getal is een getal van de vorm `a + b`i, waarbij `a` en `b` constanten zijn, en i een speciale waarde, die gedefinieerd is als de wortel uit -1. Natuurlijk kun je niet echt de wortel uit -1 berekenen, dat zou een runtime error geven; in complexe berekeningen laat je altijd de i staan. Bijvoorbeeld, het complexe getal `3 + 2`i kan niet verder gesimplificeerd worden. Het optellen van complexe getallen `a + b`i en `c + d`i is gedefinieerd als `(a + c) + (b + d)`i. Representeer een complex getal als een tuple met twee numerieke waardes, en creëer een functie die de optelling van twee complexe getallen implementeert.[16]

Opgave 11.2 Vermenigvuldiging van twee complexe getallen `a + b`i en `c + d`i is gedefinieerd als `(a*c - b*d) + (a*d + b*c)`i. Schrijf een functie die de vermenigvuldiging van twee complexe getallen implementeert.

Opgave 11.3 Stel je een nieuw data type voor. Dit nieuwe data type noem ik de `inttuple`. Een `inttuple` is gedefinieerd als zijnde ofwel een integer, ofwel een tuple die `inttuple`s als waardes bevat. Je ziet een voorbeeld van een `inttuple` hieronder. Schrijf een functie die alle integer waardes die in een `inttuple` zijn opgeslagen afdrukt. Hint: Omdat de `inttuple` recursief is gedefinieerd, is een recursieve functie waarschijnlijk de beste aanpak. Als je hoofdstuk 9 hebt overgeslagen, kun je waarschijnlijk deze opdracht beter ook overslaan. Als je hem wel maakt, dan kun je de **`isinstance()`** functie (uitgelegd in hoofdstuk 8) gebruiken om te bepalen of een element een integer of een tuple is. Als je alles correct doet, zal de functie als hij werkt op de `inttuple` die hieronder gegeven is, de getallen 1 tot en met 20 in volgorde afdrukken.

exercise1103.py

```
inttuple = ( 1, 2, ( 3, 4 ), 5, ( ( 6, 7, 8, ( 9, 10 ), 11 ), 12,
    13 ), ( ( 14, 15, 16 ), ( 17, 18, 19, 20 ) ) )
```

[16] Python heeft een apart data type `complex` dat complexe getallen representeert, dus er is eigenlijk geen noodzaak om complexe getallen als tuples te beschrijven, maar met als doel te oefenen met tuples werkt de opgave best.