Hoofdstuk 14

Sets

Sets zijn ongeordende data structuren die alleen unieke elementen kunnen bevatten. Slechts weinig programmeertalen ondersteunen het gebruik van sets, maar Python doet het wel. Sets worden niet vaak gebruikt, maar kunnen soms een leuke oplossing geven voor een probleem.

14.1 Basis van sets

Een set is een ongeordende collectie van elementen. Je kunt geen specifieke elementen van een set benaderen via een index of een key. De enige manier om elementen van een set te benaderen is via een **`for`** loop, of door met de **`in`** operator te testen of een element in de set bestaat.

Je moet bij sets denken aan wiskundige verzamelingen. In de wiskunde is een verzameling een collectie van elementen die alle uniek zijn, en ieder element zit ofwel in de verzameling, ofwel niet in de verzameling. Er zijn bepaalde operatoren die toestaan verzamelingen te combineren. Ik gebruik vanaf dit punt het woord "set" in plaats van "verzameling," omdat **`set`** een data type is.

Python gebruikt dictionaries om sets te implementeren; specifiek implementeert het de elementen van een set als keys van een dictionary. Dat betekent dat alleen onveranderbare data types in een set kunnen worden opgeslagen. Sets zelf zijn echter wel veranderbaar.

Omdat Python dictionaries gebruikt om sets te implementeren, denk je misschien dat je een lege set kunt creëren door `{}` toe te kennen aan een variabele. Dat creëert echter een lege dictionary, en niet een lege set. In plaats daarvan creëer je een lege set door de retourwaarde van de functie **`set()`** (zonder argumenten) toe te kennen aan een variabele.

Om een set te creëren waarin al een paar elementen zitten, kun je wel die elementen tussen accolades toekennen aan een variabele. Als alternatief kun je de **`set()`** functie aanroepen met een list met de elementen als argument.

```
fruitset = { "appel", "banaan", "kers" }
print( fruitset )
```

Als je een set wilt creëren bestaande uit de verschillende letters van een string, dan kun je **set()** aanroepen met de string als argument (dubbele letters worden automatisch genegeerd).

```
helloset = set( "hello world" )
print( helloset )
```

Je kunt een **for** loop gebruiken om een set te doorlopen. De variabele van de **for** loop krijgt toegang tot alle element van de set. Er is geen manier om te bepalen in welke volgorde je de elementen te zien krijgt. Je kunt ze niet sorteren zolang ze in de set zitten. Je kunt echter wel met een list casting de set omzetten in een list, en die list dan sorteren.

listing1401.py

```
fruitset = { "appel", "banaan", "kers", "doerian", "mango" }
for element in fruitset:
    print( element )
print()

fruitlist = list( fruitset )
fruitlist.sort()
for element in fruitlist:
    print( element )
```

Met behulp van de **len()** functie kun je het aantal elementen in de set vaststellen.

14.2 Set methodes

Om de inhoud van een set te manipuleren, zijn de volgende methodes beschikbaar. Dit is geen complete lijst van set methodes, maar het zijn de meest gebruikte.

14.2.1 `add()` en `update()`

Om een nieuw element aan een set toe te voegen gebruik je de `add()` methode, met het nieuwe element als argument. Als je meerdere nieuwe elementen wilt toevoegen, kun je de `update()` methode gebruiken, met een list die de nieuwe elementen bevat als argument. Je kunt ook een tuple als argument gebruiken, of zelfs een string. Als je een string gebruikt, wordt ieder teken van de string gezien als een element.

Omdat een set alleen unieke elementen kan bevatten, worden duplicaten genegeerd.

listing1402.py

```
fruitset = { "appel", "banaan", "kers", "doerian", "mango" }
print( fruitset )

fruitset.add( "appel" )
fruitset.add( "aalbes" )
print( fruitset )
```

```
fruitset.update( ["appel","appel","appel","aardbei",
    "aardbei","appel","mango"] )
print( fruitset )
```

14.2.2 remove(), discard(), en clear()

Om elementen uit een set te verwijderen, gebruik je de `remove()` of `discard()` methodes. Beide krijgen het te verwijderen element als argument. Het verschil is dat `remove()` een runtime error geeft als het element niet bestaat in de set, terwijl `discard()` niet-bestaande elementen gewoon negeert.

```
fruitset = { "appel", "banaan", "kers", "doerian", "mango" }
print( fruitset )

fruitset.remove( "appel" )
print( fruitset )
```

`clear()` verwijdert alle elementen van de set in één keer.

14.2.3 pop()

De `pop()` methode verwijdert een element uit de set en retourneert het. Je kunt niet zelf aangeven welk element je wilt verwijderen, en je kunt ook niet voorspellen welk element het is, omdat sets ongeordend zijn.

```
fruitset = { "appel", "banaan", "kers", "doerian", "mango" }
while len( fruitset ) > 0:
    print( fruitset.pop() )
```

14.2.4 copy()

Net als bij lists en dictionaries geldt voor sets dat als je een bestaande set toekent aan een andere variabele, je een alias creëert. Als je een kopie wilt creëren van de set, kun je de `copy()` methode gebruiken.

14.2.5 union()

De vereniging (Engels: "union") van twee sets is een set die alle elementen van beide verenigde sets bevat. Je kunt de `union()` methode voor de ene set, met als argument de andere set, gebruiken om de vereniging van beide geretourneerd te krijgen. Dit verandert niets aan de twee gebruikte sets zelf. Je kunt als alternatief de speciale operator | ("pipeline") gebruiken om de vereniging te creëren. Merk op: je zou misschien denken dat je de + operator kunt gebruiken om twee sets te verenigen, maar + is niet voor sets gedefinieerd, en $*$ is evenmin gedefinieerd.

listing1403.py

```
fruit1 = { "appel", "banaan", "kers" }
fruit2 = { "banaan", "kers", "doerian" }
fruitunion = fruit1.union( fruit2 )
print( fruitunion )

fruitunion = fruit1 | fruit2
print( fruitunion )
```

14.2.6 intersection()

De doorsnede (Engels: "intersection") van twee sets is een set die alleen de elementen bevat die beide samenstellende sets gemeen hebben. Je kunt de `intersection()` methode gebruiken voor de ene set, met als argument de andere set, om de doorsnede van beide geretourneerd te krijgen. Dit verandert de samenstellende sets niet. Als alternatief kun je de speciale operator & ("ampersand") gebruiken om de doorsnede te creëren.

listing1404.py

```
fruit1 = { "appel", "banaan", "kers" }
fruit2 = { "banaan", "kers", "doerian" }
fruitintersection = fruit1.intersection( fruit2 )
print( fruitintersection )

fruitintersection = fruit1 & fruit2
print( fruitintersection )
```

14.2.7 difference()

Het verschil (Engels: "difference") van twee sets is een set die de elementen van de eerste set bevat, met daaruit verwijderd de elementen die de eerste set gemeen heeft met de tweede set. Je kunt het verschil creëren door de `difference()` methode aan te roepen voor de ene set, met als argument de set waarvan je de elementen uit de eerste set wilt verwijderen. Dit verandert de samenstellende sets niet. Als alternatief kun je de speciale operator - ("minus") gebruiken om het verschil te creëren.

listing1405.py

```
fruit1 = { "appel", "banaan", "kers" }
fruit2 = { "banaan", "kers", "doerian" }
fruitdifference = fruit1.difference( fruit2 )
print( fruitdifference )

fruitdifference = fruit1 - fruit2
print( fruitdifference )

fruitdifference = fruit2 - fruit1
print( fruitdifference )
```

14.2.8 `isdisjoint()`, `issubset()`, en `issuperset()`

The methodes `isdisjoint()`, `issubset()`, en `issuperset()` worden alle aangeroepen als methodes voor een set, met een tweede set als argument. Ze retourneren alle **`True`** of **`False`**. `isdisjoint()` retourneert **`True`** als de twee sets geen elementen gemeen hebben. `issubset()` retourneert **`True`** als alle elementen van de eerste set ook voorkomen in de set die als argument dient. `issuperset()` retourneert **`True`** als alle elementen van de set die als argument dient ook voorkomen in de eerste set.

listing1406.py

```
fruit1 = { "appel", "banaan", "kers" }
fruit2 = { "banaan", "kers" }

print( fruit1.isdisjoint( fruit2 ) )
print( fruit1.issubset( fruit2 ) )
print( fruit2.issubset( fruit1 ) )
print( fruit1.issubset( fruit1 ) )
print( fruit1.issuperset( fruit2 ) )
print( fruit2.issuperset( fruit1 ) )
print( fruit1.issuperset( fruit1 ) )
```

Een set is zowel een subset als een superset van zichzelf.

14.2.9 Oefening

Opgave Er is ook een methode `symmetric_difference()` die een set retourneert die alle elementen bevat van de vereniging van twee sets, behalve de elementen die zich bevinden in de doorsnede van de twee sets. Bijvoorbeeld, als set 1 de elementen A, B, en C bevat, en set 2 de elementen B, C, en D, dan bevat de "symmetric difference" van sets 1 en 2 alleen de elementen A en D. Kun je de `symmetric_difference()` methode implementeren via de combinatie van een aantal van bovenstaande methodes?

Opgave In hoofdstuk 7 was een oefening die je vroeg om alle letters te vinden die twee woorden gemeen hebben, waarbij iedere letter slechts eenmalig gerapporteerd wordt. Dit kun je zeer efficiënt doen met sets. Schrijf de code hiervoor.

14.3 Frozensets

Python kent als variant op het set type de **`frozenset`**. Je creëert een **`frozenset`** via de **`frozenset`**`()` functie. De elementen van een **`frozenset`** kunnen niet veranderd worden. Je creëert dus een **`frozenset`** onmiddellijk als je de **`frozenset`**`()` functie aanroept, want zodra de **`frozenset`** bestaat kun je geen elementen meer toevoegen of weghalen. Met andere woorden, **`frozenset`**s zijn onveranderbaar.

Alle reguliere set methodes werken ook op **`frozenset`**s, behalve de methodes die proberen de set te veranderen. Als je een dergelijke methode probeert aan te roepen voor een **`frozenset`** krijg je een syntax error.

```
fruit1 = frozenset( ["appel", "banaan", "kers"] )
fruit2 = frozenset( ["banaan", "kers", "doerian"] )

print( fruit1.union( fruit2 ) )
```

Wat je geleerd hebt

In dit hoofdstuk is het volgende besproken:

- Sets
- Methodes `add()`, `update()`, `remove()`, `discard()`, `clear()`, `pop()`, `copy()`, `union()`, `intersection()`, `difference()`, `isdisjoint()`, `issubset()`, en `issuperset()`
- Frozensets

Opgaves

Opgave 14.1 Een beroemd syllogisme zegt: *Alle mensen zijn sterfelijk. Socrates is een mens. Daarom is Socrates sterfelijk.* In de code hieronder zie je een aantal sets. De eerste is een set van alle dingen (ik weet dat er een paar ontbreken, maar beschouw hem maar als compleet voor deze opgave). De tweede is een set van alle mensen (aannemende dat de eerste set inderdaad compleet is). De derde bevat alles wat sterfelijk is (wederom, aannemende dat...). Maak gebruik van set operatoren en methodes om te laten zien dat inderdaad geldt dat (a) alle mensen sterfelijk zijn, (b) Socrates een mens is, en (c) Socrates sterfelijk is. Laat ook zien dat (d) er sterfelijke dingen zijn die geen mens zijn, en (e) er dingen zijn die niet sterfelijk zijn.

exercise1401.py

```
alles = { "Socrates", "Plato", "Eratosthenes", "Zeus", "Hera",
    "Athene", "Acropolis", "Kat", "Hond" }
mensen = { "Socrates", "Plato", "Eratosthenes" }
sterfelijken = { "Socrates", "Plato", "Eratosthenes", "Kat",
    "Hond" }
```

Opgave 14.2 Schrijf een programma dat drie sets van getallen tussen de 1 en 1000 produceert; de eerste set bestaat uit alle getallen die deelbaar zijn door 3, de tweede uit alle getallen die deelbaar zijn door 7, en de derde uit alle getallen die deelbaar zijn door 11. Dit is het gemakkelijkste te doen met list comprehensions, maar dat is niet noodzakelijk. Produceer vervolgens sets van alle getallen tussen die 1 en 1000 die (a) zowel door 3, 7, en 11 deelbaar zijn, (b) die deelbaar zijn door 3 en 7, maar niet door 11, en (c) die noch deelbaar zijn door 3, noch door 7, en noch door 11. De kortste oplossing heeft slechts één regel code nodig voor ieder van deze zes sets.