# Christmas party

Every year, the members of a family gather around the fireplace to celebrate Christmas. At the climax of the evening, every family member gives a present to another member. Every family member gives one gift and receives one gift. In order to organize this, little cards with the names of the family members are put in a big hat a couple of weeks prior to the festivities. Every member may pull one name out of the hat. If a person pulls out his own name, all cards go back in the hat and the procedure is repeated until everybody pulls somebody else's name. Because everybody happened to pull out a name of a member of his or her own family, the family decided to repeat the process until everyone has pulled out the name of a member of another family.

## Preparation

In the `random` module, a `shuffle` is defined that shakes the elements of a given list in a random order. The session below illustrates the working of this function. Do note that the function `shuffle` doesn't print a new list, but changes the positions of the elements *in place*.

```
>>> from random import shuffle

>>> list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> shuffle(list)
>>> list
[2, 7, 8, 9, 3, 6, 1, 5, 4, 10]
>>> shuffle(list)
>>> list
[8, 10, 6, 7, 4, 2, 9, 3, 5, 1]
```

## Assignment

In this assignment we represent a **family** as a (frozen) collection of strings, where the strings contain the names of the family members. A composition of a **family** is represented as a list, a tuple or a collection of families. You may assume that a family consists of at least two families and that two persons in a family never have the same name. For a given family, a **drawing** is represented as a dictionary that portrays all names of a family one on one on the names of all family members. The name of every family member occurs once as a key and once as a value in the dictionary. This representation indicates who should give a present to whom (value that corresponds with the key).

- Write a function isValidDraw to which two arguments must be given: a draw and the composition of the family for which the draw has happened. The function must print a Boolean value that indicates whether the draw is valid or not. Here, the function may assume that the given draw represents a one on one representation of the names of the family member. However, the function must verify that no family member gives a present to himself (which makes the draw invalid). The function also has a third, optional parameter sameFamily (standard value: False), that indicates if the function should verify if a family member should give a gift to a member of his own family (which makes the draw invalid).
- Write a function makeDraw to which the composition of a family should be given. The function must print a dictionary, that represents a draw for the given family. Note that for this, the validity of the function isValidDraw, in other words, that everybody must give a present to another person of another family. These conditions will only be imposed to the function that is described below.
- Use the functions isValidDraw and makeDraw to write a function makeValidDraw, to which the composition of a family must be given. The function also has a second, optional parameter sameFamily (standard value: False), that has the same meaning as with the function isValidDraw. The function must print a dictionary, that gives a **valid** draw in the meaning as set for the function isValidDraw.

## Example

```
>>> family = ({'Alice', 'Bob', 'Cindy'}, {'Dora', 'Eric', 'Felix'})

>>> draw = {
...   'Alice': 'Bob',
...   'Cindy': 'Dora',
...   'Dora': 'Alice',
...   'Eric': 'Eric',
...   'Bob': 'Felix',
...   'Felix': 'Cindy'
... }
>>> isValidDraw(draw, family)
False

>>> draw = {
...   'Alice': 'Dora',
...   'Cindy': 'Felix',
...   'Dora': 'Eric',
...   'Eric': 'Cindy',
...   'Bob': 'Alice',
...   'Felix': 'Bob'
... }
>>> isValidDraw(draw, family)
True
>>> isValidDraw(draw, family, sameFamily=True)
False

>>> draw = {
...   'Alice': 'Dora',
...   'Cindy': 'Eric',
...   'Dora': 'Alice',
...   'Eric': 'Bob',
...   'Bob': 'Felix',
...   'Felix': 'Cindy'
... }
```

```
>>> isValidDraw(draw, family)
True
>>> isValidDraw(draw, family, sameFamily=True)
True

>>> makeDraw(family)
{
 'Alice': 'Cindy',
 'Cindy': 'Alice',
 'Dora': 'Bob',
 'Eric': 'Dora',
 'Bob': 'Felix',
 'Felix': 'Eric'
}

>>> makeValidDraw(family)
{
 'Alice': 'Cindy',
 'Cindy': 'Bob',
 'Dora': 'Alice',
 'Eric': 'Felix',
 'Bob': 'Eric',
 'Felix': 'Dora'
}
>>> makeValidDraw(family, sameFamily=True)
{
 'Alice': 'Felix',
 'Cindy': 'Eric',
 'Dora': 'Bob',
 'Eric': 'Alice',
 'Bob': 'Dora',
 'Felix': 'Cindy'
}
```

Naar jaarlijkse gewoonte scharen de gezinnen van een familie zich rond het haardvuur om Kerstmis te vieren. Op het hoogtepunt van de avond deelt elk familielid een geschenk uit aan een andere familielid, waarbij iedereen één geschenk uitdeelt en iedereen één geschenk ontvangt. Om dat te organiseren, worden er een paar weken voorafgaand aan de festiviteiten kaartjes met alle namen van de familieleden in een grote hoed gestopt en mag iedereen blind één kaartje uit de hoed trekken. Als iemand toevallig zijn eigen naam zou trekken, dan gaan alle kaartjes terug in de hoed, en wordt de procedure herhaald totdat iedereen iemand anders getrokken heeft. Omdat iedereen dit jaar toevallig een kaartje getrokken heeft waarop de naam van iemand van zijn of haar eigen gezin staat, heeft de familie afgesproken om de volgende trekking te blijven herhalen totdat niemand een kaartje getrokken heeft van een eigen gezinslid.

## Voorbereiding

In de module random wordt een functie shuffle gedefinieerd die de elementen van een gegeven lijst willekeurig door elkaar schudt. Onderstaande sessie illustreert de werking van deze functie. Merk dus op dat de functie shuffle geen nieuwe lijst teruggeeft, maar de elementen *in place* van plaats verwisselt.

```
>>> from random import shuffle

>>> lijst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> shuffle(lijst)
>>> lijst
[2, 7, 8, 9, 3, 6, 1, 5, 4, 10]
>>> shuffle(lijst)
>>> lijst
[8, 10, 6, 7, 4, 2, 9, 3, 5, 1]
```

## Opgave

In deze opgave stellen we een **gezin** voor als een (bevroren) verzameling van strings, waarbij de strings de namen van de gezinsleden bevatten. Een samenstelling van een **familie** wordt voorgesteld als een lijst, een tuple of een verzameling van gezinnen. Hierbij mag verondersteld worden dat een familie uit minstens twee gezinnen bestaat, en dat er binnen een familie geen twee personen zijn die dezelfde naam hebben. Voor een gegeven familie wordt een **trekking** voorgesteld als een dictionary die de namen van alle familieleden één-op-één afbeeldt op de namen van alle familieleden. De naam van elk familielid komt dus juist één keer als sleutel en juist één keer als waarde voor in de dictionary. Deze afbeelding stelt dus voor wie (sleutel in dictionary) een geschenk moet uitdelen aan wie (waarde die correspondeert met sleutel).

- Schrijf een functie isGeldigeTrekking waaraan twee argumenten moeten doorgegeven worden: een trekking en de samenstelling van de familie waarvoor de trekking gebeurd is. De functie moet een Booleaanse waarde teruggeven die aangeeft of de trekking al of niet geldig is. Hierbij mag de functie ervan uitgaan dat de gegeven trekking een één-op-één afbeelding van de namen van de familieleden voorstelt. De functie moet echter wel nagaan of geen enkel familielid aan zichzelf een geschenk zou moeten uitdelen (wat de trekking ongeldig maakt). De functie heeft ook nog een derde optionele parameter zelfdeGezin (standaardwaarde: False), die aangeeft of de functie bijkomend moet nagaan of geen enkel familielid een geschenk zou moeten uitdelen aan iemand van zijn of haar eigen gezin (wat de trekking ongeldig maakt).

- Schrijf een functie maakTrekking waaraan de samenstelling van een familie moet doorgegeven worden. De functie moet een dictionary teruggeven, die een trekking voor de gegeven familie voorstelt. Merk op dat hiervoor dus niet moet gelden dat de trekking geldig is volgens de functie isGeldigeTrekking, met andere woorden dat iedereen aan iemand anders een geschenk moet uitdelen of dat gezinsleden niet aan elkaar een geschenk mogen uitdelen. Deze voorwaarden zullen pas opgelegd worden aan de functie die hierna wordt beschreven.
- Gebruik de functies isGeldigeTrekking en maakTrekking om een functie maakGeldigeTrekking te schrijven, waaraan de samenstelling van een familie moet doorgegeven worden. De functie heeft ook nog een tweede optionele parameter zelfdeGezin (standaardwaarde: False), die dezelfde betekenis heeft als bij de functie isGeldigeTrekking. De functie moet een dictionary teruggeven, die een **geldige** trekking teruggeeft in de betekenis zoals vastgelegd voor de functie isGeldigeTrekking.

## Voorbeeld

```
>>> familie = ({'Alice', 'Bob', 'Cindy'}, {'Dora', 'Eric', 'Felix'})

>>> trekking = {
...   'Alice': 'Bob',
...   'Cindy': 'Dora',
...   'Dora': 'Alice',
...   'Eric': 'Eric',
...   'Bob': 'Felix',
...   'Felix': 'Cindy'
... }
>>> isGeldigeTrekking(trekking, familie)
False

>>> trekking = {
...   'Alice': 'Dora',
...   'Cindy': 'Felix',
...   'Dora': 'Eric',
...   'Eric': 'Cindy',
...   'Bob': 'Alice',
...   'Felix': 'Bob'
... }
>>> isGeldigeTrekking(trekking, familie)
True
>>> isGeldigeTrekking(trekking, familie, zelfdeGezin=True)
False

>>> trekking = {
...   'Alice': 'Dora',
...   'Cindy': 'Eric',
...   'Dora': 'Alice',
...   'Eric': 'Bob',
...   'Bob': 'Felix',
...   'Felix': 'Cindy'
... }
>>> isGeldigeTrekking(trekking, familie)
True
>>> isGeldigeTrekking(trekking, familie, zelfdeGezin=True)
True

>>> maakTrekking(familie)
```

```
{
 'Alice': 'Cindy',
 'Cindy': 'Alice',
 'Dora': 'Bob',
 'Eric': 'Dora',
 'Bob': 'Felix',
 'Felix': 'Eric'
}

>>> maakGeldigeTrekking(familie)
{
 'Alice': 'Cindy',
 'Cindy': 'Bob',
 'Dora': 'Alice',
 'Eric': 'Felix',
 'Bob': 'Eric',
 'Felix': 'Dora'
}
>>> maakGeldigeTrekking(familie, zelfdeGezin=True)
{
 'Alice': 'Felix',
 'Cindy': 'Eric',
 'Dora': 'Bob',
 'Eric': 'Alice',
 'Bob': 'Dora',
 'Felix': 'Cindy'
}
```