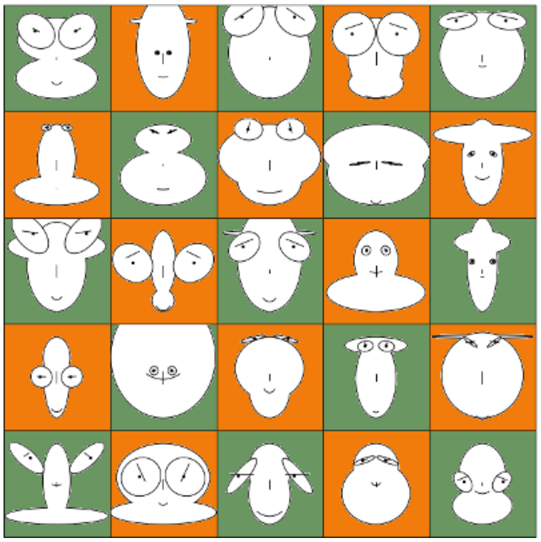
SYNTHETICAL TEST DATA WITH PYTHON

DEFINITION OF SYNTHETICAL DATA



There is hardly any engineer or scientist who doesn't understand the need for synthetical data, also called synthetic data. But some may have asked themselves what do we understand by synthetical test data? There are lots of situtations, where a scientist or an engineer needs learn or test data, but it is hard or impossible to get real data, i.e. a sample from a population obtained by measurement. The task or challenge of creating synthetical data consists in producing data which resembles or comes quite close to the intended "real life" data. Python is an ideal language for easily producing such data, because it has powerful numerical and linguistic functionalities.

Synthetic data are also necessary to satisfy specific needs or certain conditions that may not be found in the "real life" data. Another use case of synthetical data is to protect privacy of the data needed.

In our previous chapter "Python, Numpy and Probability", we have written some functions, which we will need in the following:

- find_interval
- weighted_choice
- cartesian_choice
- weighted_cartesian_choice
- weighted_sample

You should be familiar with the way of working of these functions.

We saved the functions in a module with the name bk_random.

DEFINITION OF THE SCOPE OF SYNTHETIC DATA CREATION

We want to provide solutions to the following task:

We have n finite sets containing data of various types:

D_1, D_2, ... D_n

The sets D_i are the data sets from which we want to deduce our synthetical data.

In the actual implementation, the sets will be tuples or lists for practical reasons.

The process of creating synthetic data can be defined by two functions "synthesizer" and "synthesize". Usually, the word synthesizer is used for a computerized electronic device which produces sound. Our synthesizer produces strings or alternatively tuples with data, as we will see later.

The function synthesizer creates the function synthesize:

synthesize = synthesizer((D_1, D_2, ... D_n))

The function synthesize, - which may also be a generator like in our implementation, - takes no arguments and the result of a function call sythesize() will be

- a list or a tuple t = (d_1, d_2, ... d_n) where d_i is drawn at random from D_i
- or a string which contains the elements str(d_1), str(d_2), ... str(d_n) where d_i is also drawn at random from D_i

Let us start with a simple example. We have a list of firstnames and a list of surnames. We want to hire employees for an institute or company. Of course, it will be a lot easier in our synthetical Python environment to find and hire specialsts than in real life. The function "cartesian_choice" from the bk_random module and the concatenation of the randomly drawn firstnames and surnames is all it takes.

```
import bk_random
firstnames = ["John", "Eve", "Jane", "Paul",
              "Frank", "Laura", "Robert",
              "Kathrin", "Roger", "Simone",
              "Bernard", "Sarah", "Yvonne"]
surnames = ["Singer", "Miles", "Moore",
            "Looper", "Rampman", "Chopman",
            "Smiley", "Bychan", "Smith",
            "Baker", "Miller", "Cook"]

number_of_specialists = 15

employees = set()
while len(employees) < number_of_specialists:
    employee =
bk_random.cartesian_choice(firstnames, surnames)
    employees.add(" ".join(employee))
print(employees)
```

```
{'Laura Smith', 'Yvonne Miles', 'Sarah Cook',
'Jane Smith', 'Paul Moore', 'Jane Miles', 'Jane
Looper', 'Frank Singer', 'Frank Miles', 'Jane
Cook', 'Frank Chopman', 'Laura Cook', 'Yvonne
Bychan', 'Eve Miles', 'Simone Cook'}
```

This was easy enough, but we want to do it now in a more structured way, using the synthesizer approach we mentioned before. The code for the case in which the parameter "weights" is not None is still missing in the following implementation:

```
import bk_random
firstnames = ["John", "Eve", "Jane", "Paul",
              "Frank", "Laura", "Robert",
              "Kathrin", "Roger", "Simone",
              "Bernard", "Sarah", "Yvonne"]
surnames = ["Singer", "Miles", "Moore",
            "Looper", "Rampman", "Chopman",
            "Smiley", "Bychan", "Smith",
            "Baker", "Miller", "Cook"]
```

```
def synthesizer( data, weights=None,
format_func=None, repeats=True):
    """
    data is a tuple or list of lists or tuples
containing the
    data
    weights is a list or tuple of lists or tuples
with the
    corresponding weights of the data lists or
tuples
    format_func is a reference to a function which
defines
    how a random result of the creator function
will be formated.
    If None, "creator" will return the list "res".
    If repeats is set to True, the results of
helper will not be unique
    """
    if not repeats:
        memory = set()
    def synthesize():
        while True:
            res =
bk_random.cartesian_choice(*data)
            if not repeats:
                sres = str(res)
                while sres in memory:
                    res =
bk_random.cartesian_choice(*data)
                    sres = str(res)
                memory.add(sres)
            if format_func:
                yield format_func(res)
            else:
                yield res
    return synthesize
```

```
recruit_employee = synthesizer( (firstnames,
surnames),

format_func=lambda x: " ".join(x),
                                 repeats=False)
employee = recruit_employee()
for _ in range(15):
    print(next(employee))
```

```
John Smiley
Sarah Miller
Kathrin Miles
Yvonne Chopman
Yvonne Smiley
Yvonne Smith
Yvonne Bychan
Robert Looper
Kathrin Bychan
Bernard Miller
Laura Baker
Bernard Rampman
Laura Looper
Laura Rampman
Roger Bychan
```

Every name, i.e first name and last name, had the same likehood to be drawn in the previous example. This is not very realistic, because we will expect in countries like the US or England names like Smith and Miller to occur more often than names like Rampman or Bychan. We will extend our synthesizer function with additional code for the "weighted" case, i.e. weights is not None. If weights are given, we will have to use the function weighted_cartesian_choice from the bk_random module. If "weights" is set to None, we will have to call the function cartesian_choice. We put this decision into a different subfunction of synthesizer to keep the function synthesize clearer.

We do not want to fiddle around with probabilites between 0 and 1 in defining the weights, so we take the detour with integer, which we normalize afterwards.

```
from bk_random import cartesian_choice,
weighted_cartesian_choice
weighted_firstnames = [ ("John", 80), ("Eve", 70),
("Jane", 2),
                        ("Paul", 8), ("Frank",
20), ("Laura", 6),
                        ("Robert", 17), ("Zoe",
3), ("Roger", 8),
                        ("Simone", 9), ("Bernard",
8), ("Sarah", 7),
                        ("Yvonne", 11), ("Bill",
12), ("Bernd", 10)]
weighted_surnames = [('Singer', 2), ('Miles', 2),
('Moore', 5),
                     ('Looper', 1), ('Rampman',
1), ('Chopman', 1),
                     ('Smiley', 1), ('Bychan', 1),
('Smith', 150),
                     ('Baker', 144), ('Miller',
87), ('Cook', 5),
                     ('Joyce', 1), ('Bush', 5),
('Shorter', 6),
                     ('Klein', 1)]
firstnames, weights = zip(*weighted_firstnames)
wsum = sum(weights)
weights_firstnames = [ x / wsum for x in weights]
surnames, weights = zip(*weighted_surnames)
wsum = sum(weights)
weights_surnames = [ x / wsum for x in weights]
weights = (weights_firstnames, weights_surnames)
def synthesizer( data, weights=None,
format_func=None, repeats=True):
    """
    "data" is a tuple or list of lists or tuples
containing the
    data.

    "weights" is a list or tuple of lists or
tuples with the
    corresponding weights of the data lists or
tuples.

    "format_func" is a reference to a function
which defines
    how a random result of the creator function
will be formated.
    If None,the generator "synthesize" will yield
the list "res".

    If "repeats" is set to True, the output values
yielded by
    "synthesize" will not be unique.
    """

    if not repeats:
        memory = set()

    def choice(data, weights):
        if weights:
            return
weighted_cartesian_choice(*zip(data, weights))
        else:
            return cartesian_choice(*data)
    def synthesize():
        while True:
            res = choice(data, weights)
            if not repeats:
                sres = str(res)
                while sres in memory:
                    res = choice(data, weights)
                    sres = str(res)
                memory.add(sres)
            if format_func:
                yield format_func(res)
            else:
                yield res
    return synthesize

recruit_employee = synthesizer( (firstnames,
surnames),
                                weights = weights,
                                format_func=lambda
x: " ".join(x),
                                repeats=False)
employee = recruit_employee()
for _ in range(8):
    print(next(employee))
```

```
Bill Smith
Eve Baker
Robert Miller
Eve Smith
John Miller
Roger Baker
Robert Baker
Frank Baker
```

WINE EXAMPLE

Let's imagine that you have to describe a dozen wines. Most probably a nice imagination for many, but I have to admit that it is not for me. The main reason is that I am not a wine drinker!

We can write a little Python program, which will use our synthesize function to create automatically "sophisticated criticisms" like this one:

This wine is light-bodied with a conveniently juicy bouquet leading to a lingering flamboyant finish!

Try to find some adverbs, like "seamlessly", "assertively", and some adjectives, like "fruity" and "refined", to describe the aroma.

If you have defined your lists, you can use the synthesize function.

Here is our solution, in case you don't want to do it on your own:

```
import bk_random
body = ['light-bodied', 'medium-bodied', 'full-
bodied']

adverbs = ['appropriately', 'assertively',
'authoritatively',
           'compellingly', 'completely',
'continually',
           'conveniently', 'credibly',
'distinctively',
           'dramatically', 'dynamically',
'efficiently',
           'energistically', 'enthusiastically',
'fungibly',
           'globally', 'holisticly',
'interactively',
           'intrinsically', 'monotonectally',
'objectively',
           'phosfluorescently', 'proactively',
'professionally',
           'progressively', 'quickly',
'rapidiously',
           'seamlessly', 'synergistically',
'uniquely']
noun = ['aroma', 'bouquet', 'flavour']
aromas = ['angular', 'bright', 'lingering',
'butterscotch',
          'buttery', 'chocolate', 'complex',
'earth', 'flabby',
          'flamboyant', 'fleshy', 'flowers', 'food
friendly',
          'fruits', 'grass', 'herbs', 'jammy',
'juicy', 'mocha',
          'oaked', 'refined', 'structured',
'tight', 'toast',
          'toasty', 'tobacco', 'unctuous',
'unoaked', 'vanilla',
          'velvetly']

example = """This wine is light-bodied with a
completely buttery
bouquet leading to a lingering fruity  finish!"""
def describe(data):
    body, adv, adj, noun, adj2 = data
    format_str = "This wine is %s with a %s %s
%s\nleading to"
    format_str += " a lingering %s finish!"
    return format_str % (body, adv, adj, noun,
adj2)

t = bk_random.cartesian_choice(body, adverbs,
aromas, noun, aromas)
data = (body, adverbs, aromas, noun, aromas)
synthesize = synthesizer( data, weights=None,
format_func=describe, repeats=True)
criticism = synthesize()
```

```
for i in range(1, 13):
    print("{0:d}. wine:".format(i))
    print(next(criticism))
    print()

1. wine:
This wine is full-bodied with a professionally
structured flavour
leading to a lingering unctuous finish!
2. wine:
This wine is medium-bodied with a quickly mocha
aroma
leading to a lingering unoaked finish!
3. wine:
This wine is full-bodied with a energistically
fruits aroma
leading to a lingering mocha finish!
4. wine:
This wine is light-bodied with a intrinsically
grass flavour
leading to a lingering fruits finish!
5. wine:
This wine is full-bodied with a quickly toasty
bouquet
leading to a lingering oaked finish!
6. wine:
This wine is medium-bodied with a fungibly
flamboyant aroma
leading to a lingering unctuous finish!
7. wine:
This wine is light-bodied with a completely food
friendly aroma
leading to a lingering refined finish!
8. wine:
This wine is light-bodied with a compellingly
herbs bouquet
```

```
leading to a lingering flabby finish!
9. wine:
This wine is full-bodied with a authoritatively
angular bouquet
leading to a lingering vanilla finish!
10. wine:
This wine is medium-bodied with a authoritatively
fleshy flavour
leading to a lingering toasty finish!
11. wine:
This wine is medium-bodied with a progressively
butterscotch flavour
leading to a lingering chocolate finish!
12. wine:
This wine is medium-bodied with a seamlessly herbs
aroma
leading to a lingering flamboyant finish!
```

EXERCISE: INTERNATIONAL DISASTER OPERATION

It would be gorgeous, if the problem described in this exercise, would be purely synthetic, i.e. there would be no further catastophes in the world. Completely unrealistic, but a nice daydream. So, the task of this exercise is to provide synthetical test data for an international disaster operation. The countries taking part in this mission might be e.g. France, Switzerland, Germany, Canada, The Netherlands, The United States, Austria, Belgium and Luxembourg.

We want to create a file with random entries of aides. Each line should consist of:

UniqueIdentifier, FirstName, LastName, Country, Field

For example:

```
001, Jean-Paul,  Rennier, France, Medical Aid
002, Nathan, Bloomfield, Canada, Security Aid
003, Michael, Mayer, Germany, Social Worker
```

For practical reasons, we will reduce the countries to France, Italy, Switzerland and Germany in the following example implementation:

```
from bk_random import cartesian_choice,
weighted_cartesian_choice
countries = ["France", "Switzerland", "Germany"]
w_firstnames = { "France" : [ ("Marie", 10),
("Thomas", 10),
                            ("Camille", 10),
("Nicolas", 9),
                            ("LÃ©a", 10),
("Julien", 9),
                            ("Manon", 9),
("Quentin", 9),
                            ("ChloÃ©", 8),
("Maxime", 9),
                            ("Laura", 7),
("Alexandre", 6),
                            ("Clementine", 2),
("GrÃ©gory", 2),
                            ("Sandra", 1),
("Philippe", 1)],
               "Switzerland": [ ("Sarah", 10),
("Hans", 10),
                            ("Laura", 9),
("Peter", 8),
                            ("MÃ©lissa", 9),
("Walter", 7),
                            ("OcÃ©ane", 7),
("Daniel", 7),
                            ("NoÃ©mie", 6),
("Reto", 7),
                            ("Laura", 7),
("Bruno", 6),
                            ("Eva", 2), ("Urli",
4),
                            ("Sandra", 1),
("Marcel", 1)],
               "Germany": [ ("Ursula", 10),
("Peter", 10),
                            ("Monika", 9),
("Michael", 8),
                            ("Brigitte", 9),
("Thomas", 7),
                            ("Stefanie", 7),
("Andreas", 7),
                            ("Maria", 6),
("Wolfgang", 7),
                            ("Gabriele", 7),
("Manfred", 6),
                            ("Nicole", 2),
("Matthias", 4),
                            ("Christine", 1),
("Dirk", 1)],
               "Italy" : [ ("Francesco", 20),
("Alessandro", 19),
                            ("Mattia", 19),
("Lorenzo", 18),
                            ("Leonardo", 16),
("Andrea", 15),
                            ("Gabriele", 14),
("Matteo", 14),
                            ("Tommaso", 12),
("Riccardo", 11),
                             ("Sofia", 20),
("Aurora", 18),
                             ("Giulia", 16),
("Giorgia", 15),
                             ("Alice", 14),
("Martina", 13)]}

w_surnames = { "France" : [ ("Matin", 10),
("Bernard", 10),
                           ("Camille", 10),
("Nicolas", 9),
                           ("Dubois", 10),
("Petit", 9),
                             ("Durand", 8),
("Leroy", 8),
                             ("Fournier", 7),
("Lambert", 6),
                             ("Mercier", 5),
("Rousseau", 4),
                             ("Mathieu", 2),
("Fontaine", 2),
                             ("Muller", 1),
("Robin", 1)],
               "Switzerland": [ ("MÃ¼ller", 10),
("Meier", 10),
                             ("Schmid", 9),
("Keller", 8),
                             ("Weber", 9),
("Huber", 7),
                             ("Schneider", 7),
("Meyer", 7),
                             ("Steiner", 6),
("Fischer", 7),
                             ("Gerber", 7),
("Brunner", 6),
                             ("Baumann", 2),
("Frei", 4),
                            ("Zimmermann", 1),
("Moser", 1)],
               "Germany": [ ("MÃ¼ller", 10),
("Schmidt", 10),
                            ("Schneider", 9),
("Fischer", 8),
                            ("Weber", 9),
("Meyer", 7),
                            ("Wagner", 7),
("Becker", 7),
                            ("Schulz", 6),
("Hoffmann", 7),
                            ("SchÃ¤fer", 7),
("Koch", 6),
                            ("Bauer", 2),
("Richter", 4),
                            ("Klein", 2),
("SchrÃ¶der", 1)],
               "Italy" : [ ("Rossi", 20),
("Russo", 19),
                            ("Ferrari", 19),
("Esposito", 18),
                            ("Bianchi", 16),
("Romano", 15),
                            ("Colombo", 14),
("Ricci", 14),
                            ("Marino", 12),
("Grecco", 11),
                            ("Bruno", 10),
("Gallo", 12),
                            ("Conti", 16), ("De
Luca", 15),
                            ("Costa", 14),
("Giordano", 13),
                            ("Mancini", 14),
("Rizzo", 13),
                            ("Lombardi", 11),
("Moretto", 9)]}
# separate names and weights
synthesize = {}
identifier = 1
for country in w_firstnames:
    firstnames, weights =
zip(*w_firstnames[country])
    wsum = sum(weights)
    weights_firstnames = [ x / wsum for x in
weights]
    w_firstnames[country] = [firstnames,
weights_firstnames]
    surnames, weights = zip(*w_surnames[country])
    wsum = sum(weights)
    weights_surnames = [ x / wsum for x in
weights]
    w_surnames[country] = [surnames,
weights_firstnames]
    synthesize[country] = synthesizer(
(firstnames, surnames),

(weights_firstnames,

weights_surnames),

format_func=lambda x: " ".join(x),
                                 repeats=False)
nation_prob = [("Germany", 0.3),
               ("France", 0.4),
               ("Switzerland", 0.2),
               ("Italy", 0.1)]
profession_prob = [("Medical Aid", 0.3),
                   ("Social Worker", 0.6),
                   ("Security Aid", 0.1)]
```

```
helpers = []
for _ in range(200):
    country =
weighted_cartesian_choice(zip(*nation_prob))
    profession =
weighted_cartesian_choice(zip(*profession_prob))
    country, profession = country[0],
profession[0]
    s = synthesize[country]()
    uid = "{id:05d}".format(id=identifier)
    helpers.append((uid, country, next(s),
profession ))
    identifier += 1

print(helpers)
```

```
[('00001', 'France', 'Thomas Durand', 'Medical
Aid'), ('00002', 'France', 'Maxime Petit', 'Social
Worker'), ('00003', 'France', 'Alexandre Petit',
'Medical Aid'), ('00004', 'Switzerland', 'MÃ©lissa
Meier', 'Social Worker'), ('00005', 'Switzerland',
'Daniel Schneider', 'Medical Aid'), ('00006',
'Switzerland', 'OcÃ©ane Meier', 'Social Worker'),
('00007', 'Switzerland', 'Walter Frei', 'Social
Worker'), ('00008', 'France', 'Nicolas Dubois',
'Security Aid'), ('00009', 'Germany', 'Ursula
Koch', 'Social Worker'), ('00010', 'France',
'GrÃ©gory Petit', 'Medical Aid'), ('00011',
'Switzerland', 'Walter Zimmermann', 'Medical
Aid'), ('00012', 'Switzerland', 'Urli Weber',
'Social Worker'), ('00013', 'France', 'Marie
Matin', 'Social Worker'), ('00014', 'France',
'Julien Petit', 'Social Worker'), ('00015',
'Germany', 'Wolfgang Wagner', 'Social Worker'),
('00016', 'Germany', 'Manfred Becker', 'Security
Aid'), ('00017', 'France', 'ChloÃ© Lambert',
'Security Aid'), ('00018', 'Italy', 'Matteo
Ricci', 'Medical Aid'), ('00019', 'France',
'Thomas Dubois', 'Medical Aid'), ('00020',
'France', 'Quentin Dubois', 'Social Worker'),
('00021', 'Switzerland', 'Hans Frei', 'Security
Aid'), ('00022', 'Switzerland', 'OcÃ©ane Huber',
'Medical Aid'), ('00023', 'Germany', 'Thomas
Richter', 'Social Worker'), ('00024', 'France',
'Manon Camille', 'Social Worker'), ('00025',
'Germany', 'Wolfgang Hoffmann', 'Social Worker'),
('00026', 'Germany', 'Monika Becker', 'Social
Worker'), ('00027', 'France', 'ChloÃ© Rousseau',
'Social Worker'), ('00028', 'France', 'Laura
Bernard', 'Social Worker'), ('00029', 'France',
'Julien Lambert', 'Social Worker'), ('00030',
'Switzerland', 'Hans Steiner', 'Social Worker'),
('00031', 'France', 'LÃ©a Matin', 'Social
Worker'), ('00032', 'Switzerland', 'Peter
Steiner', 'Security Aid'), ('00033',
'Switzerland', 'Eva Weber', 'Social Worker'),
('00034', 'Switzerland', 'Sarah Schmid', 'Social
Worker'), ('00035', 'France', 'Camille Camille',
'Medical Aid'), ('00036', 'Germany', 'Thomas
Meyer', 'Social Worker'), ('00037', 'France',
'Manon Dubois', 'Security Aid'), ('00038',
'Switzerland', 'Laura Weber', 'Medical Aid'),
('00039', 'France', 'Thomas Camille', 'Medical
Aid'), ('00040', 'France', 'Camille Dubois',
'Social Worker'), ('00041', 'Italy', 'Francesco
Costa', 'Security Aid'), ('00042', 'France',
'Julien Camille', 'Social Worker'), ('00043',
'France', 'Thomas Petit', 'Medical Aid'),
('00044', 'Germany', 'Matthias Becker', 'Social
Worker'), ('00045', 'France', 'Manon Nicolas',
'Medical Aid'), ('00046', 'Switzerland', 'Peter
Keller', 'Medical Aid'), ('00047', 'Germany',
'Brigitte Hoffmann', 'Security Aid'), ('00048',
'Italy', 'Francesco Lombardi', 'Social Worker'),
('00049', 'Germany', 'Brigitte Fischer', 'Social
Worker'), ('00050', 'Switzerland', 'Sarah
Fischer', 'Medical Aid'), ('00051', 'Germany',
'Monika Schneider', 'Medical Aid'), ('00052',
'Germany', 'Peter Schmidt', 'Medical Aid'),
('00053', 'Switzerland', 'NoÃ©mie MÃ¼ller',
'Medical Aid'), ('00054', 'Switzerland', 'Laura
Schneider', 'Medical Aid'), ('00055', 'France',
'Nicolas Durand', 'Social Worker'), ('00056',
'Switzerland', 'Hans Weber', 'Social Worker'),
('00057', 'Germany', 'Manfred MÃ¼ller', 'Security
Aid'), ('00058', 'Germany', 'Maria Schmidt',
'Social Worker'), ('00059', 'Switzerland', 'Reto
Meyer', 'Social Worker'), ('00060', 'France',
'LÃ©a Nicolas', 'Security Aid'), ('00061',
'France', 'Manon Durand', 'Social Worker'),
('00062', 'Switzerland', 'Peter Gerber', 'Social
Worker'), ('00063', 'France', 'LÃ©a Bernard',
'Social Worker'), ('00064', 'Germany', 'Monika
MÃ¼ller', 'Medical Aid'), ('00065', 'Germany',
'Monika Hoffmann', 'Social Worker'), ('00066',
'Italy', 'Leonardo Esposito', 'Social Worker'),
('00067', 'France', 'Alexandre Matin', 'Social
Worker'), ('00068', 'Switzerland', 'Sarah Weber',
'Social Worker'), ('00069', 'France', 'Maxime
Leroy', 'Medical Aid'), ('00070', 'Italy',
'Francesco Ferrari', 'Medical Aid'), ('00071',
'Germany', 'Monika Klein', 'Medical Aid'),
('00072', 'France', 'Camille Durand', 'Social
Worker'), ('00073', 'France', 'Quentin Mercier',
'Social Worker'), ('00074', 'Germany', 'Gabriele
Becker', 'Medical Aid'), ('00075', 'Germany',
'Andreas Schulz', 'Social Worker'), ('00076',
'Germany', 'Thomas Schneider', 'Social Worker'),
('00077', 'Switzerland', 'Sarah MÃ¼ller', 'Social
Worker'), ('00078', 'Switzerland', 'MÃ©lissa
MÃ¼ller', 'Social Worker'), ('00079', 'France',
'Nicolas Rousseau', 'Social Worker'), ('00080',
'Germany', 'Maria Hoffmann', 'Medical Aid'),
('00081', 'Switzerland', 'MÃ©lissa Meyer', 'Social
Worker'), ('00082', 'Germany', 'Thomas Koch',
'Medical Aid'), ('00083', 'Switzerland', 'Laura
Zimmermann', 'Security Aid'), ('00084', 'France',
'Marie Camille', 'Social Worker'), ('00085',
'Germany', 'Gabriele Hoffmann', 'Social Worker'),
('00086', 'Switzerland', 'Daniel Zimmermann',
'Social Worker'), ('00087', 'Switzerland', 'Laura
Gerber', 'Social Worker'), ('00088',
'Switzerland', 'Peter Schmid', 'Security Aid'),
('00089', 'France', 'Camille Lambert', 'Social
Worker'), ('00090', 'France', 'Maxime Durand',
'Medical Aid'), ('00091', 'Switzerland', 'MÃ©lissa
Keller', 'Social Worker'), ('00092',
'Switzerland', 'Laura Steiner', 'Medical Aid'),
('00093', 'France', 'Camille Nicolas', 'Social
Worker'), ('00094', 'Germany', 'Ursula Weber',
'Security Aid'), ('00095', 'Germany', 'Manfred
Wagner', 'Medical Aid'), ('00096', 'France',
'Philippe Leroy', 'Medical Aid'), ('00097',
'Switzerland', 'Sarah Gerber', 'Social Worker'),
('00098', 'France', 'Philippe Nicolas', 'Social
Worker'), ('00099', 'France', 'Clementine Durand',
'Security Aid'), ('00100', 'France', 'Laura
Nicolas', 'Social Worker'), ('00101', 'France',
'LÃ©a Petit', 'Medical Aid'), ('00102', 'France',
'Manon Fontaine', 'Medical Aid'), ('00103',
'Switzerland', 'Laura Meier', 'Social Worker'),
('00104', 'France', 'LÃ©a Leroy', 'Medical Aid'),
('00105', 'Germany', 'Thomas Weber', 'Security
Aid'), ('00106', 'France', 'Laura Petit',
'Security Aid'), ('00107', 'France', 'Marie
Nicolas', 'Social Worker'), ('00108',
'Switzerland', 'Laura Meyer', 'Medical Aid'),
('00109', 'Switzerland', 'Hans Baumann', 'Social
Worker'), ('00110', 'Germany', 'Maria Wagner',
'Security Aid'), ('00111', 'Switzerland', 'Daniel
Huber', 'Social Worker'), ('00112', 'France',
'LÃ©a Mathieu', 'Social Worker'), ('00113',
'Italy', 'Mattia De Luca', 'Social Worker'),
('00114', 'France', 'Thomas Bernard', 'Social
Worker'), ('00115', 'Switzerland', 'Walter
Steiner', 'Medical Aid'), ('00116', 'Switzerland',
'Eva Schmid', 'Security Aid'), ('00117',
'Switzerland', 'Bruno Fischer', 'Social Worker'),
('00118', 'France', 'Julien Bernard', 'Medical
Aid'), ('00119', 'France', 'Manon Mercier',
'Social Worker'), ('00120', 'Germany', 'Stefanie
Schmidt', 'Social Worker'), ('00121', 'France',
'Thomas Fournier', 'Medical Aid'), ('00122',
'France', 'Nicolas Nicolas', 'Social Worker'),
('00123', 'France', 'Manon Matin', 'Social
Worker'), ('00124', 'Switzerland', 'Laura
MÃ¼ller', 'Social Worker'), ('00125', 'Germany',
'Manfred Schmidt', 'Social Worker'), ('00126',
'Germany', 'Brigitte MÃ¼ller', 'Medical Aid'),
('00127', 'France', 'ChloÃ© Mercier', 'Medical
Aid'), ('00128', 'France', 'Nicolas Petit',
'Social Worker'), ('00129', 'France', 'Alexandre
Bernard', 'Security Aid'), ('00130',
'Switzerland', 'Reto Keller', 'Social Worker'),
('00131', 'France', 'Camille Mercier', 'Social
Worker'), ('00132', 'Germany', 'Matthias MÃ¼ller',
'Medical Aid'), ('00133', 'Switzerland', 'Bruno
Huber', 'Social Worker'), ('00134', 'Switzerland',
'Daniel Brunner', 'Social Worker'), ('00135',
'France', 'Camille Fournier', 'Social Worker'),
('00136', 'France', 'Camille Fontaine', 'Social
Worker'), ('00137', 'France', 'Marie Fournier',
'Social Worker'), ('00138', 'France', 'LÃ©a
Fournier', 'Security Aid'), ('00139', 'France',
'Thomas Leroy', 'Social Worker'), ('00140',
'Switzerland', 'Laura Huber', 'Social Worker'),
('00141', 'Germany', 'Ursula Schulz', 'Medical
Aid'), ('00142', 'Germany', 'Matthias Schmidt',
'Social Worker'), ('00143', 'France', 'Julien
Mathieu', 'Medical Aid'), ('00144', 'Switzerland',
'NoÃ©mie Schmid', 'Medical Aid'), ('00145',
'France', 'ChloÃ© Durand', 'Social Worker'),
('00146', 'Germany', 'Michael Meyer', 'Medical
Aid'), ('00147', 'Germany', 'Peter Koch', 'Social
Worker'), ('00148', 'Germany', 'Peter Schulz',
'Medical Aid'), ('00149', 'France', 'Quentin
Nicolas', 'Social Worker'), ('00150', 'France',
'Laura Durand', 'Medical Aid'), ('00151',
'France', 'Nicolas Leroy', 'Social Worker'),
('00152', 'Italy', 'Lorenzo Romano', 'Social
Worker'), ('00153', 'Germany', 'Thomas Schmidt',
'Medical Aid'), ('00154', 'France', 'Camille
Robin', 'Medical Aid'), ('00155', 'France', 'LÃ©a
Durand', 'Security Aid'), ('00156', 'Germany',
'Brigitte Richter', 'Social Worker'), ('00157',
'Switzerland', 'Peter Baumann', 'Social Worker'),
('00158', 'Germany', 'Thomas Wagner', 'Social
Worker'), ('00159', 'France', 'ChloÃ© Nicolas',
'Social Worker'), ('00160', 'Germany', 'Thomas
Hoffmann', 'Medical Aid'), ('00161', 'Germany',
'Monika Koch', 'Medical Aid'), ('00162', 'France',
'Maxime Dubois', 'Social Worker'), ('00163',
'Italy', 'Gabriele Giordano', 'Social Worker'),
('00164', 'France', 'Maxime Nicolas', 'Social
Worker'), ('00165', 'France', 'Nicolas Bernard',
'Social Worker'), ('00166', 'France', 'ChloÃ©
```

```
Petit', 'Medical Aid'), ('00167', 'France',
'GrÃ©gory Mercier', 'Security Aid'), ('00168',
'Italy', 'Francesco Conti', 'Medical Aid'),
('00169', 'France', 'Julien Durand', 'Social
Worker'), ('00170', 'Switzerland', 'Laura
Fischer', 'Social Worker'), ('00171', 'France',
'Marie Fontaine', 'Social Worker'), ('00172',
'Germany', 'Gabriele Koch', 'Social Worker'),
('00173', 'France', 'Quentin Matin', 'Medical
Aid'), ('00174', 'France', 'ChloÃ© Dubois',
'Social Worker'), ('00175', 'France', 'Thomas
Matin', 'Medical Aid'), ('00176', 'France',
'GrÃ©gory Leroy', 'Medical Aid'), ('00177',
'France', 'Maxime Bernard', 'Medical Aid'),
('00178', 'France', 'Marie Bernard', 'Social
Worker'), ('00179', 'Germany', 'Michael Fischer',
'Medical Aid'), ('00180', 'Germany', 'Manfred
Schneider', 'Social Worker'), ('00181', 'Germany',
'Wolfgang MÃ¼ller', 'Medical Aid'), ('00182',
'Switzerland', 'NoÃ©mie Fischer', 'Social
Worker'), ('00183', 'Switzerland', 'Urli Steiner',
'Medical Aid'), ('00184', 'Italy', 'Leonardo
Bianchi', 'Medical Aid'), ('00185', 'Germany',
'Andreas MÃ¼ller', 'Medical Aid'), ('00186',
'France', 'Manon Fournier', 'Social Worker'),
('00187', 'France', 'Clementine Lambert', 'Medical
Aid'), ('00188', 'Germany', 'Stefanie Fischer',
'Social Worker'), ('00189', 'Germany', 'Monika
Meyer', 'Social Worker'), ('00190', 'France',
'Nicolas Camille', 'Social Worker'), ('00191',
'France', 'Alexandre Leroy', 'Social Worker'),
('00192', 'Germany', 'Michael Schneider', 'Social
Worker'), ('00193', 'France', 'Laura Fournier',
'Security Aid'), ('00194', 'Switzerland', 'NoÃ©mie
Schneider', 'Social Worker'), ('00195', 'France',
'Clementine Dubois', 'Social Worker'), ('00196',
'Germany', 'Manfred Weber', 'Security Aid'),
('00197', 'France', 'Marie Dubois', 'Security
Aid'), ('00198', 'France', 'Alexandre Dubois',
'Medical Aid'), ('00199', 'France', 'Nicolas
Fontaine', 'Medical Aid'), ('00200',
'Switzerland', 'Laura Schmid', 'Medical Aid')]

with open("disaster_mission.txt", "w") as fh:
    fh.write("Reference
number,Country,Name,Function\n")
    for el in helpers:
        fh.write(",".join(el) + "\n")
```

In [ ]: