Aske Svane Qvist

# Exercise W37Wed: Structures

Builds on exercises provided with Programming with Python for Social Science by Phillip Brooker and from Social Data Science 2019 by Andreas Bjerre-Nielson.

## 1. Strings

You learned basic things about strings last week. You can also break strings into sub-strings by slicing them with the `[]` notation, like you would with a list. The notation is `string[start:end]` where start is included and end is excluded.

> *Recall that Python has zero-based indexing, see explanation [here (https://softwareengineering.stackexchange.com/questions/110804/why-are-zero-based-arrays-the-norm)](https://softwareengineering.stackexchange.com/questions/110804/why-are-zero-based-arrays-the-norm).*

Strings also have a variety of methods that help you manipulate them in various useful ways.

> *Remember that you can consult the [documentation for strings (https://docs.python.org/2.5/lib/string-methods.html)](https://docs.python.org/2.5/lib/string-methods.html) to find relevant methods.*

> **Task 1.1**: Let `s1 = 'Chameleon` . From the string `s1` select the last four characters. What is the index of the character `a` in `s1` ?

In [6]:

```
s1 = "Chameleon"
last4 = s1[-4:]
print(last4)
```

```
leon
```

In [7]:

```
s1.index("a") # index 2
```

Out[7]:

```
2
```

> **Task 1.2**: Create a string `my_url` that is a URL of your choice. Retrieve the part of a string before a specified character. For example, if your URL is the link for our course: [https://kurser.ku.dk/course/ASDK20001U](https://kurser.ku.dk/course/ASDK20001U)

(https://kurser.ku.dk/course/ASDK20001U) and specific character is the last '/', then you should retrieve https://kurser.ku.dk/course (https://kurser.ku.dk/course).

In [13]:

```
my_url = "https://www.dr.dk/nyheder/udland/ord-ord-von-der-leyen-gjorde-statu

# Split from the right - split twice thus I have three strings
url_sub = my_url.rsplit("/", 2)
# Take the first string in the list.
url_sub = url_sub[0]
url_sub
```

Out[13]:

```
'https://www.dr.dk/nyheder'
```

## 2. Dictionaries

Dictionaries are useful when you want to associate pieces of data together. They are also another building block underlying DataFrames.

**Task 2.1**: Suppose you want to make a new dictionary that stores information about you. Make a dictionary called `my_dictionary` that stores:

- Your favourite animal
- Your favourite colour
- Your favourite TV Show
- Your favourite film

In [39]:

```
my_dictionary = {"animal": "cat", "color": "green", "tv": "Borgen", "film": "
```

In [40]:

```
my_dictionary
```

Out[40]:

```
{'animal': 'cat', 'color': 'green', 'tv': 'Borgen', 'film': 'druk'}
```

**Task 2.2**: Add a key and value to the your dictionary with your favourite band or artist.

In [41]:

```
my_dictionary["band"] = "Major Lazer"
```

In [42]:

```
my_dictionary
```

Out[42]:

```
{'animal': 'cat',
 'color': 'green',
 'tv': 'Borgen',
 'film': 'druk',
 'band': 'Major Lazer'}
```

**Task 2.3**:

1. Delete the favourite film field.
2. Create an empty dictionary with the variable name as your favourite movie
3. Populate the dictionary with the following keys (and give them values):
   - Year
   - Genre
4. Do steps 2 and 3 for a two or three movies.
5. Add the dictionaries of movies to a new dictionary called `favourite_movies` with the movie title as the key for each entry.
6. Add `favourite_movies` to your `my_dictionary` under the key "Favourite movies"

In [43]:

```
# 1
# I use pop to remove key from dict.
my_dictionary.pop('film', None)
```

Out[43]:

```
'druk'
```

In [44]:

```
# 2
druk = {}
```

In [45]:

```
# 3
druk["Year"] = 2020
druk["Genre"] = "Drama/Comedy"
```

In [46]:

```
# 4
respect = {}
respect["Year"] = 2021
respect["Genre"] = "Drama"

avengers = {}
avengers["Year"] = 2012
avengers["Genre"] = "Action"
```

In [47]:

```
# 5
favourite_movies = {}
favourite_movies["respect"] = respect
favourite_movies["druk"] = druk
favourite_movies["avengers"] = avengers
favourite_movies
```

Out[47]:

```
{'respect': {'Year': 2021, 'Genre': 'Drama'},
 'druk': {'Year': 2020, 'Genre': 'Drama/Comedy'},
 'avengers': {'Year': 2012, 'Genre': 'Action'}}
```

In [48]:

```
my_dictionary["Favourite Movies"] = favourite_movies
my_dictionary
```

Out[48]:

```
{'animal': 'cat',
 'color': 'green',
 'tv': 'Borgen',
 'band': 'Major Lazer',
 'Favourite Movies': {'respect': {'Year': 2021, 'Genre': 'Drama'},
  'druk': {'Year': 2020, 'Genre': 'Drama/Comedy'},
  'avengers': {'Year': 2012, 'Genre': 'Action'}}}
```

> **Task 2.4**: See if you can print out different keys and values from `my_dictionary` .

In [64]:

```
# print 'first layer' key
my_dictionary["animal"]
# Dig down in the nested dictionaries
my_dictionary["Favourite Movies"]["avengers"]["Year"]
```

Out[64]:

```
2012
```

> **Task 2.5:** In previous exercises you created the SODAS Dataframe. Here, you are given the code to convert part of that dataframe into a dictionary. This should be useful as later, when you start using the Twitter API, you might want to check if a person has a twitter handle before trying to search for them on Twitter. Using the `sodas_twitter_dict` given below, write an if statement that checks if a person has a Twitter handle. If they do, print out the handle. If they don't, print out 'No Twitter handle available.'
>
> *Hint: First create a variable `name` , which is equal to the name of the person you want to search. You can then change the value of the variable to test your if-statement*
>
> *Hint 2: You'll need to use the pandas `pd.isna(value)` to check the value of the twitter handle, which returns True if the value is equivalent to nan (i.e. nothing) and False otherwise.*

> *Note: You should not yet be able to convert the pandas dataframe into a dictionary. This is why the code is already given below.*

In [111]:

```
# Loading the pandas module
import pandas as pd

# Loading the SODAS dataframe

url = 'https://dl.dropboxusercontent.com/s/9war4suj1s5j1ah/sodas_people_twitt

sodas_people_df = pd.read_csv(url)

# selecting the columns we are interested in

sodas_people_twitter_df = sodas_people_df[['name', 'twitter_handle']]

# converting to dictionary

sodas_twitter_dict = sodas_people_twitter_df.set_index('name')['twitter_handl
```

In [112]:

```
# define name
name = 'David Dreyer Lassen'

# Test whether the twitter handle is 'nan'
if pd.isna(sodas_twitter_dict[name]) == True:

    # if true print message
    print("No Twitter handle available.")

else:
    # print the twitter handle
    print(sodas_twitter_dict[name])
```

```
daviddlassen
```

## 4. Tuples

As you may already know, tuples are immutable objects. In this part of the exercise, our aim is to understand how tuple operations work. For that purpose we will be working with the following tuple:

In [113]:

```
my_tuple = ("Samantha", [50, 60, 70], (1, 2, 3))
```

> **Task 4.1**: Reverse the tuple.

In [114]:

```
my_tuple = my_tuple[::-1]
my_tuple
```

Out[114]:

```
((1, 2, 3), [50, 60, 70], 'Samantha')
```

**Task 4.2**: Access the value 3 of your tuple.

In [115]:

```
# index start at 0
my_tuple[2]
```

Out[115]:

```
'Samantha'
```

**Task 4.3**: Copy the element "Samantha" and the inner tuple, that is (1, 2, 3), into a new tuple.

In [116]:

```
new_tuple = (my_tuple[0], my_tuple[2])
new_tuple
```

Out[116]:

```
((1, 2, 3), 'Samantha')
```

**Task 4.4**: Modify the second element of the list that is inside my_tuple. Then modify the third element that is inside my_tuple. What happens?

In [117]:

```
# Overwrite element from the list in the tuple
my_tuple[1][1] = 61
```

In [118]:

```
my_tuple[0][2] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-118-21c769fe5d0f> in <module>
----> 1 my_tuple[0][2] = 4

TypeError: 'tuple' object does not support item assignment
```

It works when I try to modify an element in the list in the tuple but not when I try to modify the tuple in the tuple.

# 5. Sets

> **Task 5.1**: the following line of code selects the 'role' column from the SODAS people dataframe:
>
> ```
> sodas_people_df['role']
> ```
>
> Think about whether you would store the information in this column in a list or in a set and why. Try doing both and see what happens.

In [119]:

```
role_list = list(sodas_people_df['role'])
role_list[0:3]
```

Out[119]:

```
['SODAS steering committee',
 'SODAS steering committee',
 'SODAS steering committee']
```

In [120]:

```
role_set = set(role_list)
role_set
```

Out[120]:

```
{'Administration',
 'Assistant Professor',
 'PhD Student',
 'Postdoctoral Researcher',
 'Research Assistant',
 'SODAS Affiliated',
 'SODAS steering committee',
 'Student Assistant'}
```

When storing the information as 'set', all duplicates disappear

- fast overview of the different roles employees have

When storing as a list, we keep all values

- we have every the role for every employee

> **Task 5.2**: Check if SODAS people role set has any elements in common with a new role set that contains 'Teaching Assistant', Friedolin Merhout and Helene Willadsen's roles. If yes, display the common elements between them. If not, print "These two sets have no roles in common".
>
> *Note 1*: See python set methods (https://www.w3schools.com/python/python_ref_set.asp) for help.
>
> *Note 2*: Please do not code in order to obtain Friedolin and Helene roles. You only need to look at the dataframe.

In [128]:

```
new_roles = {"Teaching Assistant", "Postdoctoral Researcher", "Assistant Prof

# if there exist any values that are teh same in the two sets - print d
if len(role_set.intersection(new_roles)) > 0:
    print(role_set.intersection(new_roles))

else:
    print("These two elements have no roles in common")
```

```
{'Assistant Professor', 'Postdoctoral Researcher'}
```

## 6. Variable names

> **Task 6.1**: Choose three variables that you have defined in this notebook and explain why you have named them like that. Do you think the names are meaningful and enough for you and others to understand the content they store?

**role_set:** I named one variable role_set because it is a set of roles as opposed to role_list. In that way, I know whether I am working with a set of roles or a list of roles.

**role_list:** In relation, I named the list of roles for 'role_list' to make them distinguishable.

**new_tuple:** I make another tuple apart from the predefined 'my_tuple'. I thought i made sense to call it 'new_tuple'. However, it might end up being problemtic in case I were to create even more tuples. Luckely, that was not the case

*I generally use underscores when writing variable names to make it clear when I use more than one word*

## 7. Reflection

> Write a brief paragraph reflecting on your experience learning programming today. What did you struggle with? What did you enjoy? What surprised you?

- I think it was super nice to work through sets and dictionaries - I never did that and it is nice to get an intuitive good basic understanding of how these containers work (and not just lists) :D
- I enjoyed that we were asked to use ifelse statements so we are using the stuff from earlier lectures
- It is nice that we have to look up online in order to get through. It is a nice practice.

## 8. Bonus exercise

> **Task 8.1**: Imagine you are interested in retriving followers information from the SODAS dataset. Store the the names of the SODAS people in a list and in a separate list store the number of followers of each person. Retrive the number of followers of David Dreyer Lassen and Sune Lehmann and calculate the difference in the number of followers between them (in absolute values).
>
> Now try calculating the same for David Dreyer Lassen and Kristin Anabel Eggeling. What happens?
>
> Discuss whether there is an ethical issue by using this dataframe to compare followers between different persons on the SODAS database?

In [130]:

```
# Create list of names
sodas_names = list(sodas_people_df["name"])
```

In [138]:

```
# create list of followers
sodas_followers = list(sodas_people_df["twitter_followers_n"])
```

In [145]:

```
# Get name index in the list to retrieve followers in the other list
sodas_names.index("David Dreyer Lassen")
sodas_names.index("Sune Lehmann")
```

Out[145]:

```
3
```

In [150]:

```
# Get their followers
Lassen_followers = sodas_followers[sodas_names.index("David Dreyer Lassen")]
Lehmann_followers = sodas_followers[sodas_names.index("Sune Lehmann")]

# calculate the difference
followers_dif = abs(Lassen_followers - Lehmann_followers)
followers_dif
```

Out[150]:

```
1890.0
```

In [151]:

```
# Get their followers
Eggeling_followers = sodas_followers[sodas_names.index("Kristin Anabel Eggeli

# calculate the difference
followers_dif = abs(Lassen_followers - Eggeling_followers)
followers_dif
```

Out[151]:

```
nan
```

When calculating the difference between Lassen and Eggeling, the result is just 'nan' - Eggeling's number of followers is probably a missing value.

### Ethical issue

> In this specific context, it is arguably not an ethical issue, since the comparison is not used in any way apart from the purpose of learning to code. However, in other cases it could be problematic. Even though participants have given consent, comparing things such as followers and likes while keeping their names in the data could cause harm and humiliation. In various social groups, social status is to some extent understood as reflected in ones social media profiles and the degree to which other people show interest in them.

In [ ]: