Aske Svane Qvist

Exercises W38Wed: Loops

Builds on exercises from Social Data Science 2020 by Andreas Bjerre-Nielson.

1. Loops

Loops are another form of control flow. Loops do something repeatedly. In python you can loop through anything that is *iterable*, e.g. anything where it makes sense to say "for each element in this item, do whatever." Lists, tuples and dictionaries are all iterable, and can thus be looped over. This kind of loop is called a *for loop*. The basic syntax is:

```
for element in SomeIterable:
    dosomething(element)
```

where `element` is a temporary name given to the current element reached in the loop, and `dosomething()` can be any valid python code.

Example - try the following code:

In [1]:
```
A = []

for i in [1,3,5]:
    i_squared = i**2
    A.append(i_squared)

print(A)
```

```
[1, 9, 25]
```

Notice above, you can define a list right in the middle of your for loop. Another way to do this would be to declare [1,3,5] beforehand and assign it to a variable, say `odd_numbers` . So your code would look like:

```
odd_numbers = [1,3,5]
for i in odd_numbers:
    #etc.
```

A key thing with loops, like with if-statements, to help your Python Interpreter know what to include in the loop, you use white space. This should be four whitespaces or a tab.

Exercise 1.1: Write a for loop that runs 10 times and prints out "This is x time through the loop" where x is the time/iteration, starting the count at 1.

Hint: You can easily make a list that's a specific range of numbers using Python's range function.

In [4]:
```python
for number in (range(10)):
    print(f"This is {number + 1} time through the loop")
```

```
This is 1 time through the loop
This is 2 time through the loop
This is 3 time through the loop
This is 4 time through the loop
This is 5 time through the loop
This is 6 time through the loop
This is 7 time through the loop
This is 8 time through the loop
This is 9 time through the loop
This is 10 time through the loop
```

Exercise 1.2: Notice that after the first time, the grammar in your sentence is a bit off. Using your for loop above, add a check so that if it's the first time through the loop it prints "This is 1 time through the loop" but if it's more than the first time, print "This is 2 times through the loop" (you want to add an "s" to times).

In [5]:
```python
for number in (range(10)):
    if number == 0:
        print(f"This is {number + 1} time through the loop")
    else:
         print(f"This is {number + 1} times through the loop")
```

```
This is 1 time through the loop
This is 2 times through the loop
This is 3 times through the loop
This is 4 times through the loop
This is 5 times through the loop
This is 6 times through the loop
This is 7 times through the loop
This is 8 times through the loop
This is 9 times through the loop
This is 10 times through the loop
```

Another way to do this is with a `while` loop. Instead of looping over an iterable the `while` loop continues going as long as a supplied logical condition is True. Most commonly the while loop is combined with a counting variable that keeps track of how many times the loop has been run. One specific application where a while loop can be useful is data collection on the internet (scraping) which is often open ended. Another application is when we are computing something we do not know how long will take to compute, e.g. for a model to converge.

The basic syntax is seen in the example below. This code will run 100 times before stopping. At each iteration it checks that `i` is smaller than 100. If it is, it does something and adds 1 to the variable `i` before repeating.

```
i = 0
while i < 100:
    dosomething()
    i = i + 1
```

Exercise 1.3: Try exercise 1.2 again, but this time with a while loop. Remember that you'll want a variable to keep track of how many times you've gone through the loop. This is really convenient for this question, because you also want to print out how many times you've gone through the loop.

Hint: Remember to increment your counter at each iteration, otherwise your program will run forever. If this happens, press the stop button.

In [7]:
```
count = 0
while count < 10:
    count += 1
    if count == 1:
        print(f"This is {count} time through the loop")
    else:
         print(f"This is {count} times through the loop")
```

```
This is 1 time through the loop
This is 2 times through the loop
This is 3 times through the loop
This is 4 times through the loop
This is 5 times through the loop
This is 6 times through the loop
This is 7 times through the loop
This is 8 times through the loop
This is 9 times through the loop
This is 10 times through the loop
```

Exercise 1.4: Now imagine you want to run your program unitl you have reach a number of iterations. For instance, you want to have on a list only 100 entries. Try adding consecutive numbers from 1 to 100 in a list using a while loop. That is you should end up with a list that contains numbers from 1 to 100.

In [10]:
```
count = 0
list100 = []
while count < 100:
    count += 1
    list100.append(count)
```

In [14]:
```
list100[:10]
```

Out[14]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Exercise 1.5: Imagine you have a list of words as input. You aim at writing a program that prints something while you search for an exact match of one word such as "python" in the given list. Take as example the following list:

```
["data", "science", "python", "programming"]
```

Your program should output:

"data does not match python"

"science does not match python"

"python matches python"

"programming does not match python"

In [16]:
```
wordlist = ["data", "science", "python", "programming"]

for word in wordlist:
    if word == "python":
        print(f"python matches python")
    else:
        print(f"{word} does not match python")
```

```
data does not match python
science does not match python
python matches python
programming does not match python
```

Exercise 1.6: Given the following code. Try to see what is going on during the iterations.

Note: Do not execute the code, just explain with your own words the steps in a markdown cell below.

```
    for element in list(range(20, 30, 3)):
        element += 3
        if element % 2 == 0:
            print(str(element) + " is even")
        else:
            print(str(element) + " is odd")
```

My explanation

- The loop goes over every third number in the range between 20 and 30.
- Then 3 is added to the number
- The if-else statement then tests whether this number is even or odd and the number is printed out with text saying whether it is odd or even

Exercise 1.7: Recall our list of population size `DK_population` from previous exercises. Using a loop, calculate the average population of Denmark from year 1901 to 2020 (the whole list). Round to the nearest whole number.

Hint: there's a function for that.

In [17]:

```
# Data from https://www.statbank.dk/10021
DK_population =[2447,2477,2506,2532,2560,2589,2621,2652,2687,2722,
                2757,2788,2820,2851,2886,2921,2958,2991,3027,3061,
                3265,3306,3340,3372,3406,3439,3467,3487,3510,3531,
                3557,3590,3620,3651,3683,3711,3738,3765,3794,3826,
                3849,3882,3926,3973,4023,4075,4123,4168,4211,4252,
                4285,4315,4349,4389,4424,4454,4479,4501,4532,4566,
                4601,4630,4666,4703,4741,4779,4820,4855,4879,4907,
                4951,4976,5008,5036,5054,5065,5080,5097,5112,5122,
                5124,5119,5116,5112,5111,5116,5125,5129,5130,5135,
                5146,5162,5181,5197,5216,5251,5275,5295,5314,5330,
                5349,5368,5384,5398,5411,5427,5447,5476,5511,5535,
                5561,5581,5603,5627,5660,5707,5749,5781,5806,5823]
```

In [24]:

```
sum_population = 0
for year in range(len(DK_population)):
    sum_population += DK_population[year]


round(sum_population / len(DK_population))
```

Out[24]:

```
4363
```

Exercise 1.8: Do the same as in Exercise 1.4, but instead of using the list

`DK_population` , now create and use a dictionary called `DK_population_dict` . Remember when going through the dictionary that you want to calculate the average of the population, not the years.

In [40]:
```
# Creating the dictionary
DK_population_dict = {}

year = []
for i in range(len(DK_population)):
    # Get year
    year = (i + 1901)
    # append the i values (population) to the key (the given year)
    DK_population_dict[str(year)] = DK_population[i]
```

In [49]:
```
# Calculating the average
pop_sum = 0
for year in DK_population_dict.keys():
    pop = DK_population_dict[str(year)]
    pop_sum += pop

round(pop_sum / len(DK_population_dict))
```

Out[49]:
```
4363
```

Loops and lists work together really well, since often you want to go through your list and either count elements or make an average of them, for example. Let's recreate the dictionary you made last week of sodas people and Twitter handles.

In [54]:
```
# 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)

# Get a list of the people at SODAS
sodas_people = sodas_people_df.name.tolist()

# Create a list of twitter handles from the SODAS data frame
twitter_handles = sodas_people_df.twitter_handle.tolist()

# Zipping together the names and twitter handles
sodas_twitter_dict = dict(zip(sodas_people, twitter_handles))
sodas_twitter_dict
```

Out[54]:
```
{'David Dreyer Lassen': 'daviddlassen',
 'Morten Axel Pedersen': nan,
 'Rebecca Adler-Nissen': 'rebadlernissen',
 'Sune Lehmann': 'suneman',
 'Anders Blok': nan,
 'Søren Kyllingsbæk': nan,
 'Robert Böhm': 'robert_bohm',
 'Andreas Bjerre-Nielsen': 'andbjn',
 'Frederik Hjorth': 'fghjorth',
 'Kristoffer Albris': nan,
 'Friedolin Merhout': 'fmerhout',
 'Gregory Eady': 'GregoryEady',
 'Samantha Breslin': nan,
 'Hjalmar Alexander Bang Carlsen': nan,
 'Patrice Wangen': nan,
 'Kristin Anabel Eggeling': nan,
 'Helene Willadsen': nan,
 'Mette My Madsen': nan,
 'Anna Rogers': nan,
 'Kristoffer Pade Glavind': nan,
 'Asger Andersen': nan,
 'Yangliu Fan': nan,
 'Thyge Enggaard': nan,
 'Kelton Ray Minor': 'keltonminor',
 'Sigriður Svala Jónasdóttir': nan,
 'Terne Sasha Thorn Jakobsen': nan,
 'Eva Iris Otto': nan,
 'Anna Helene Kvist Møller': nan,
 'Nicklas Johansen': nan,
 'Germans Savčišens': nan,
 'Snorre Ralund': 'SnorreRalund',
 'Annika Isfeldt': nan,
 'Yevgeniy Golovchenko': 'Golovchenko_Yev',
 'Sophie Smitt Sindrup Grønning': nan,
 'Louise Marie Lausten': nan,
 'Jonas Skjold Raaschou-Pedersen': nan,
 'Zoe Anna Skovby Burke': nan,
 'Frederik Carl Windfeld': nan,
 'Sofie Læbo Astrupgaard': nan,
 'Malene Hornstrup Jespersen': nan,
 'Viktor Due Pedersen': nan,
 'Cathrine Valentin Kjær': nan,
 'Esben Brøgger Lemminger': nan,
 'Sara Vera Marjanovic': nan,
 'Asger Hans Thomsen': nan,
 'Emilie Munch-Gregersen': nan,
 'Tobias Priesholm Gårdhus': 'gaardhus',
 'Jonathan Holm Salka': nan,
 'Clara Rosa Sandbye': nan,
 'Nete Schwennesen': nan,
 'Simon Westergaard Lex': nan}
```

Exercise 1.9: Using your new expertise with loops, count how many SODAS people have Twitter handles in our SODAS DataFrame.

Hint: You can reuse some of your code from last week that checks if a person has a Twitter handle.

In [71]:

```
count = 0
for person in sodas_twitter_dict.keys():
    if pd.isna(sodas_twitter_dict[person]) == False:
        count += 1
    else:
        pass

# Check the amount of twitter handles
count
```

Out[71]:

```
12
```

Exercise 1.10: As we can see, one thing loops are good for is counting things! One easy and common way of analyzing text (speeches, tweets, etc.) are by counting the different words in the text.

First, make sure you've downloaded the text file `Speech_2019-4.txt` from Absalon and saved it in the same folder as your exercise notebook. This is the speech given by DK Prime Minister Mette Frederiksen at the opening of parliament. Original found here (http://www.stm.dk/_p_14878.html).

Next, run the cell below that reads in the text file and puts it in a list.

Note: we'll cover how to do this yourselves next week!

In [73]:
```python
# Opening file
file = open('data/Speech_2019-4.txt', 'r', errors = 'ignore')

# Reading the text
text = file.read()

# Turning the text into a lowercase list of words
# Note because text.lower() returns a string,
# The split() method can be then called on that string
speech_words = text.lower().split()
```

Now, count the number of each word on the list.

Hint: You'll need to combine several things we've learned so far, including loops and containers. Think about what kind of container would be most useful to hold all of the unique words and their count.

In [80]:
```python
speech_dict = {}

for word in speech_words:

    if word in speech_dict.keys():
        speech_dict[word] = speech_dict[word] + 1
    else:
        speech_dict[word] = 1

speech_dict
```

```
 'as': 30,
 'rosy': 1,
 'that?': 1,
 'wish': 5,
 'could': 4,
 'no.': 1,
 'cannot.': 1,
 'though': 10,
 'frame': 1,
 'around': 1,
 'stable.': 1,
 'painting': 1,
 'faded': 1,
 'somewhat.': 2,
 'now': 8,
 'video': 1,
 'surveillance.': 1,
 'paradox,': 1,
 'overall': 1,
 'wealth': 1,
```

Next, think about what kind of loop and containers did you use and why. Answer this in a markdown cell below.

- I used a regular for-in loop since at had the total list of words I wanted to loop over
- I used a dictionary as the container since I can assign a value (count) to a key (word) in a super ordered manner. In addition, the values can be changed in a dictionary so I could constantly update the count when encountering the same word again.

Type *Markdown* and LaTeX: α^2

2. List Comprehension

Exercise 2.1: Let's try using a more complex way of looping over list with list comprehension.

First, use the list you defined above called DK_population and select only those entries that are under 5000.

```
In [86]: DK_population_sub = [pop for pop in DK_population if pop < 5000]
```

Now, select those words stored in the variable called speech_words that are over 5 letters.

```
In [90]: len(speech_words[0])

Out[90]: 5

In [92]: long_words = [word for word in speech_words if len(word) > 5]
```

3. Daily reflections

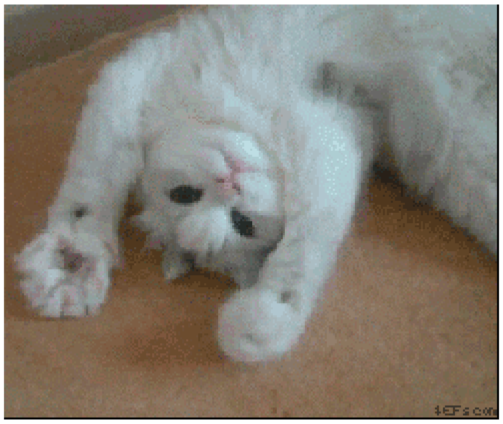
Exercise Take a moment to reflect on your learning experience today and take notes. You *can* use these prompts to inspire your reflections:

- What (if anything) did you take away from the lecture and exercise today?
- What did you find interesting or challenging?
- Are there any things you would have wanted to spend less or more time on? What are they?
- Is there anything you have become inspired to follow up on from today's lecture and exercise? Anything you are looking forward to learning more about?

It was really nice to go over list comprehension. I feel like knowing how to do that enhance my coding skills. It was also good to work with while-loops, since I have never done that before. I am looking forward to apply these skills in useful ways :D

Type *Markdown* and LaTeX: α^2

This is all for today. 🙌



4. Bonus exercise

Exercise 4.1: Write a program that outputs the following pattern:

9 8 7 6 5

8 7 6 5

7 6 5

6 5

5

In [ ]:

```

```

Exercise 4.2: Look for a text file (with .txt extension) that you find interesting and read the file using Python commands. Then, count how many words there are and how many are starting with the letter a.

Hint: You can check the code above to see how to read in a text file.

Hint: Remember that Python is case sensitve.

If you want to complicate things a bit more, you can try to count how many words starting with the different vowels there are.

Oliver Twist

In [109]:

```
# importing the data

# Opening file
file = open('data/Dickens_Oliver_1839.txt', 'r', errors = 'ignore')

# Reading the text
text = file.read()
```

In [107]:

```
import string

# Remove puncturation
exclude = set(string.punctuation)
text = ''.join(ch for ch in text if ch not in exclude)

# Lower case and split
twist_text = text.lower().split()
```

In [113]:
```
# Create dictionary container and a vowel list
vowel_dict = {}
vowel_dict["words_total"] = 0
vowels = ["a","e","i","o","y"]

# Loop over every word in the list of words
for word in twist_words:

    # Count the number of words in the text
    vowel_dict["words_total"] = vowel_dict["words_total"] + 1

    # get the first letter
    first_letter = word[0]

    # If the first letter appears in the list of vowels...
    if first_letter in vowels:
        # and if the letter already exists as a key in the dictionary..
        if first_letter in vowel_dict.keys():
            # Then increase the value count of that key with 1.
            vowel_dict[first_letter] = vowel_dict[first_letter] + 1
        else:
            # else, create a new key named after the vowel and let the value
            vowel_dict[first_letter] = 1

    else:
        pass
```

In [114]:
```
vowel_dict
```

Out[114]:
```
{'words_total': 158587, 'o': 9876, 'i': 9699, 'a': 17742, 'e': 2584, 'y': 2
876}
```

In [ ]:
```
```