Aske Svane Qvist

Exercises W39Mon: Midway Review

We have now covered most of the fundamental constructs in learning to program!

And so, today we are going to review them so that you have a solid foundation moving forward. Remember when solving a problem that you should stop and think about it first. Break down the problem into smaller parts and solve it 'inside out'. Solve the smallest pieces first then use those to solve the bigger pieces, and so on. It often helps to write out pseudocode of what your program should do.

0. Feedback on each other's code

The exercise consists on going through the different exercises in a previous notebook - we recommend you use the loops exercises - to look for different or unexpected lines of code compared to the ones you used - but remember that there are different approaches to code, which are not necessarily better or worse. This part should be of around 10-15 minutes. If you are in a group of 4 people, you can change notebooks in pairs, if you are in a group of 3 you can swap 3 ways.

Afterwards, you should give feedback to each other. For instance, if you have suggestions for improving, e.g. better commenting, clearer variable naming, etc. This last part should also take 10-15 minutes.

In a markdown cell below, write down the suggestions you recieved and/or new functions, methods or approaches that you may have learnt.

This exercise should take about 30 minutes in total.

Type *Markdown* and LaTeX: α^2

1. Indexing and Methods

Accessing the exact element or piece of data we want in any object is a crucial basic skill in programming. You have learned about a host of different Python objects and their respective ways of indexing including strings, lists, tuples, and dictionaries. As they are all different structures, indexing works slightly different for each of them. The next couple of tasks are meant to help you internalize these different techniques.

1.1. Strings

Let's pick up our string from earlier and see how we can access different elements in it.

```
In [1]: a_title = 'Honorable'
```

> **Task 1.1.1** Retrieve the word 'Honor' from the object `a_title` using string slicing.

```
In [2]: a_title[0:5]
        # Starts at 0 and the end index is not included

Out[2]: 'Honor'
```

> **Task 1.1.2** Retrieve the word 'able' from the object `a_title` using string slicing.

```
In [3]: a_title[-4:len(a_title)]

Out[3]: 'able'
```

> **Task 1.1.3** Retrieve the word 'elba' from the object `a_title` by indexing backwards from the back of the string.

```
In [16]: a_title[8:4:-1]

Out[16]: 'elba'
```

> **Task 1.1.4** In your own words, describe briefly how indexing for strings works refering to the role of 1) numbers, 2) colons (`:`), and 3) minus signs (`-`).

- `Numbers` indicate the index. Always starts at zero. The ending index is not included
- `colons` indicate a spectrum - from one index to another. Additionally, a third colon facilitates the posibility of adding '-1' which will reverse the element.
- `Minus signs` will let you count from the right.

Type *Markdown* and LaTeX: α^2

1.2. Lists

> **Task 1.2.1** Use string methods to turn the object `a_title` into a list containing the elements 'Honor' and ' ble' assigning the result to a new object called `title_elements` .
>
> *Note:* Remember that you can use auto-completion to see methods and attributes of an object by placing the cursor at the end of your code–in this case after `a_title.` –and pressing `Tab` . And to find out more about any of the suggestions, use the `help` function, e.g. `help(a_title.count)`

In [17]:

```
title_elements = a_title.split("a")
title_elements
```

Out[17]:

```
['Honor', 'ble']
```

> **Task 1.2.2** Retrieve the element 'Honor' via list indexing from `title_elements` .

In [19]:

```
title_elements[0]
```

Out[19]:

```
'Honor'
```

Oftentimes lists we work with are significantly longer than our `title_elements` object. In such case, it is not feasible to identify the relevant index for the element we are interested in by eyeballing the string. In such cases, the built-in list method `index` allows us to retrieve the first the index for the element we are interested in.

In [20]:

```
longer_list = a_title.split('a')[::-1] * 5 + a_title.split('a')[:1] * 5 + [a_titl
```

```
In [21]: longer_list
```

```
Out[21]: ['ble',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'Honor',
          'Honor',
          'Honor',
          'Honor',
          'Honor',
          'Honorable',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'ble',
          'Honor',
          'ble']
```

> **Task 1.2.3** Use the `index` method to first retrieve the index of the element 'Honorable' from `longer_list` and second to retrieve only the 'Honorable' element from `longer_list` .
>
> *Note:* There are at least two ways of doing this. Can you find both?

```
In [23]: longer_list.index("Honorable")
```

```
Out[23]: 15
```

```
In [24]: longer_list[15]
```

```
Out[24]: 'Honorable'
```

1.3. Dictionaries

The intuition behind the `index` function for lists is similar to how we access elements in dictionaries, i.e. by name or key rather than by position. Let's work with a familiar dataset but turn it into a dictionary this time.

In [264]:
```
import pandas as pd

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

sodas_people_df = pd.read_csv(url)
```

In [265]:
```
# set names as indices for DataFrame
sodas_people_df = sodas_people_df.set_index('name')

# turn DataFrame into dictionary of dictionaries using `to_dict` and `index` orie
sodas_people_dict = sodas_people_df.to_dict('index')
```

> **Task 1.3.1** Show how you find the relevant key and draw the information for Hjalmar from the `sodas_people_dict` dictionary.

In [266]:
```
# Check the keys because I don't remember Hjalmars lastname
sodas_people_dict.keys()
```

Out[266]:
```
dict_keys(['David Dreyer Lassen', 'Morten Axel Pedersen', 'Rebecca Adler-Nisse
n', 'Sune Lehmann', 'Anders Blok', 'Søren Kyllingsbæk', 'Robert Böhm', 'Andreas
Bjerre-Nielsen', 'Frederik Hjorth', 'Kristoffer Albris', 'Friedolin Merhout',
'Gregory Eady', 'Samantha Breslin', 'Hjalmar Alexander Bang Carlsen', 'Patrice
Wangen', 'Kristin Anabel Eggeling', 'Helene Willadsen', 'Mette My Madsen', 'Ann
a Rogers', 'Kristoffer Pade Glavind', 'Asger Andersen', 'Yangliu Fan', 'Thyge E
nggaard', 'Kelton Ray Minor', 'Sigriður Svala Jónasdóttir', 'Terne Sasha Thorn
Jakobsen', 'Eva Iris Otto', 'Anna Helene Kvist Møller', 'Nicklas Johansen', 'Ge
rmans Savčišens', 'Snorre Ralund', 'Annika Isfeldt', 'Yevgeniy Golovchenko', 'S
ophie Smitt Sindrup Grønning', 'Louise Marie Lausten', 'Jonas Skjold Raaschou-P
edersen', 'Zoe Anna Skovby Burke', 'Frederik Carl Windfeld', 'Sofie Læbo Astrup
gaard', 'Malene Hornstrup Jespersen', 'Viktor Due Pedersen', 'Cathrine Valentin
Kjær', 'Esben Brøgger Lemminger', 'Sara Vera Marjanovic', 'Asger Hans Thomsen',
'Emilie Munch-Gregersen', 'Tobias Priesholm Gårdhus', 'Jonathan Holm Salka', 'C
lara Rosa Sandbye', 'Nete Schwennesen', 'Simon Westergaard Lex'])
```

In [35]:
```
# Retrieve information on Hjalmar
sodas_people_dict['Hjalmar Alexander Bang Carlsen']
```

Out[35]:
```
{'description': 'Hjalmar Alexander Bang Carlsen\xa0works in the intersection be
tween social data science, political sociology and pragmatism.\xa0He\xa0has 2 m
ain projects 1) activists patterns of engagement 2) methodological issues withi
n quantitative and qualitative text analysis, and especially their combination.
Social Data Science Skills: Text Methodology, Digital Mixed Methods, Interactio
nism, Social Media SODAS courses: Digital Methods: From Ethnography to Supervis
ed Machine Learning,\xa0Re-tooling Social Analysis: Behaviors, Networks, Ideas
in the digital age SODAS projects: Solidarity and Volunteering in the Corona Cr
isis,\xa0The Dynamics of Political Discourse and Attention during the COVID-19
outbreak,\xa0COVID-19 Snapshot MOnitoring in Denmark (COSMO Denmark) hc@sodas.k
u.dkGoogle Scholar',
 'role': 'Postdoctoral Researcher',
 'twitter': nan,
 'google_scholar': 'https://scholar.google.dk/citations?user=um6jqCgAAAAJ&hl=e
n',
 'mail': 'hc@soc.ku.dk',
 'twitter_handle': nan,
 'twitter_created_at': nan,
 'twitter_bio': nan,
 'twitter_followers_n': nan,
 'twitter_friends_n': nan,
 'twitter_listed_n': nan,
 'twitter_location': nan,
 'twitter_status_n': nan,
 'twitter_name': nan,
 'twitter_verified': nan,
 'gs_cites_all_time': 72.0,
 'gs_cites_since_2015': 71.0,
 'gs_h_index_all_time': 5.0,
 'gs_i10_index_all_time': 3.0,
 'gs_most_cited_title': "The World of Edgerank: Rhetorical Justifications of Fa
cebook's News Feed Algorithm",
 'gs_most_cited_authors': 'A Birkbak, HB Carlsen',
 'gs_most_cited_outlet': 'Computational Culture (5), Special Issue on Rhetoric
and Computation, 2016',
 'gs_most_cited_year': 2016.0,
 'gs_most_cited_cites': 21.0}
```

Task 1.3.2 Retrieve the title of Søren Kyllingsbæk's most cited article.

Note: You are working with a dictionary of dictionaries, and at each level you can use dictionary methods to identify the keys and retrieve information.

In [39]:
```
sodas_people_dict['Søren Kyllingsbæk']['gs_most_cited_title']
```

Out[39]:
```
'A neural theory of visual attention: bridging cognition and neurophysiology.'
```

2. Loops

> **Task 2.1**: Use a for loop to check in sodas_people_dict for two chosen people and count how many nan values they have and in which fields. It may be a good idea to write some pseudocode first on how to approach this exercise.
>
> *Hint 1*: you can try selecting the first two entries in the dictionary with a for or while loop.
>
> *Hint 2*: recall pd.isna().
>
> *Hint 3*: nested loops and if/else statements may be a good solution to the problem.

Get the first two names with a for loop.

In [125]:

```python
# List of all people in sodas
researcher_all = list(sodas_people_dict.keys())
# empty container
researcher_list = []

# Loop over the first 2 in list
for i in range(3,7):
    # Append to empty container
    researcher_list.append(people_all[i])
```

In [126]:

```python
#Dictionary container
dictionary = {}


for researcher in researcher_list:

    # Create empty dictionary for information
    info_dict = {}

    # Retrieve information for the researcher
    info = sodas_people_dict[researcher]

    # List the value and key information in two seperate lists
    value_info = list(info.values())
    key_info = list(info.keys())

    # Store na categories
    booleans = pd.isna(value_info)

    # Create category container
    nan_categories = []

    # loop over every boolean. If it is true, append the category to 'nan_categor
    for i, boolean in enumerate(booleans):
        if boolean == True:
            nan_categories.append(key_info[i])
        else:
            pass

    # Store na count and na categories in the empty info_dict
    info_dict["count"] = sum(booleans)
    info_dict["nan_categories"] = nan_categories

    # Create key based on person and append dictionary with info
    dictionary[researcher] = info_dict

#dictionary
dictionary
```

Out[126]:

```
{'Sune Lehmann': {'count': 0, 'nan_categories': []},
 'Anders Blok': {'count': 21,
  'nan_categories': ['twitter',
   'google_scholar',
   'twitter_handle',
   'twitter_created_at',
   'twitter_bio',
   'twitter_followers_n',
   'twitter_friends_n',
   'twitter_listed_n',
   'twitter_location',
   'twitter_status_n',
   'twitter_name',
   'twitter_verified',
   'gs_cites_all_time',
   'gs_cites_since_2015',
   'gs_h_index_all_time',
   'gs_i10_index_all_time',
   'gs_most_cited_title',
   'gs_most_cited_authors',
   'gs_most_cited_outlet',
   'gs_most_cited_year',
   'gs_most_cited_cites']},
 'Søren Kyllingsbæk': {'count': 11,
  'nan_categories': ['twitter',
   'twitter_handle',
   'twitter_created_at',
   'twitter_bio',
   'twitter_followers_n',
   'twitter_friends_n',
   'twitter_listed_n',
   'twitter_location',
   'twitter_status_n',
   'twitter_name',
   'twitter_verified']},
 'Robert Böhm': {'count': 10,
  'nan_categories': ['google_scholar',
   'gs_cites_all_time',
   'gs_cites_since_2015',
   'gs_h_index_all_time',
   'gs_i10_index_all_time',
   'gs_most_cited_title',
   'gs_most_cited_authors',
   'gs_most_cited_outlet',
   'gs_most_cited_year',
   'gs_most_cited_cites']}}
```

> **Task 2.2**: Use a for loop to write a program that prints the sum of the squares of the integers from 1 to 10 to the power of 4.
>
> *Hint*: The range() function may be handy for you to solve the exercise.

In [137]:

```python
Sum = 0

# loop over the range 1-10
for i in range(1,11):

    #Add the square of every integer to the total sum
    Sum = Sum + i**2

# Print the sum to the power of 4.
print(Sum**4)
```

```
21970650625
```

> **Task 2.3**: Use a while loop to write a program that after asking the user for a number (positive integer > 0), counts the number of digits in the given integer.

Hint:

In [154]:

```
30//10
```

Out[154]:

```
True
```

In [167]:

```
# Input only integers
num = int(input())

# my while loop continues until receiving positive int
while num < 0:
    # If negative message is printed
    print("The number is not a positive integer")
    # next try
    num = int(input())

# If conditions are met, print message:
print(f"The number '{num}' contains {len(str(num))} digits.")
```

```
3
The number '3' contains 1 digits.
```

3. If/else statements

Task 3.1: write a program that from this list of numbers [60, 199, 40, 405, 57, 377, 570, 73] displays only the numbers that are:

1. divisible by 2
2. if number is bigger than 67, does nothing and continues (if there are more elements in the list)
3. if number is bigger than 405, stops reading the list

In [175]:
```
num_list = [60, 199, 40, 405, 57, 377, 570, 73]

# Loop over every number
for num in num_list:

    # First, break if the number is bigger than 405. Then nothing else matters
    if num > 405:
        break

    # print if it lives up to both conditions under 67 and divisible by 2.
    elif num%2 == 0 and num < 67:
        print(num)
```

```
60
40
```

4. Pandas and dictionaries

> **Task 4.1**: Read in the sales.xlsx file using pandas. Explore what kind of information is present in the file. You should see that the data contains 4 columns - product name, number of sales, price and total amount. The last column should be number of sales times the price for each row/product.
>
> Now, think about how you can construct a dictionary from the data and try to correct any information contained in it that does not add up to you. Print the final dictionary.
>
> *Hint*: you can take a look here (https://pandas.pydata.org/pandas-docs/stable/reference/io.html) to see how to read in an excel file into a pandas dataframe.

In [181]:
```
# Import data
data = pd.read_excel("data/sales.xlsx")

data
```

Out[181]:

	product name	number of sales	price	total amount
0	product1	2	3000	6000
1	product2	4	450	1800
2	product3	5	300	67
3	product4	1	34	68

In [233]:

```
# create header list
header = list(data)

# create empty container dict
sales_dict = {}

# loop over every row in the data
for i, r in data.iterrows():

    # Empty row dict
    row_dict = {}
    # Extract the info from the row and use the header list to create keys in dic
    row_dict[header[1]] = r[1]
    row_dict[header[2]] = r[2]
    row_dict[header[3]] = r[3]

    # save in container dict
    sales_dict[r[0]] = row_dict

sales_dict
```

Out[233]:

```
{'product1': {'number of sales': 2, 'price': 3000, 'total amount': 6000},
 'product2': {'number of sales': 4, 'price': 450, 'total amount': 1800},
 'product3': {'number of sales': 5, 'price': 300, 'total amount': 100},
 'product4': {'number of sales': 1, 'price': 34, 'total amount': 68}}
```

In [241]:

```
# Correct dictionary
sales_dict["product3"]["total amount"] = sales_dict["product3"]['number of sales'
sales_dict["product4"]["total amount"] = sales_dict["product4"]['number of sales'

# Print
sales_dict
```

Out[241]:

```
{'product1': {'number of sales': 2, 'price': 3000, 'total amount': 6000},
 'product2': {'number of sales': 4, 'price': 450, 'total amount': 1800},
 'product3': {'number of sales': 5, 'price': 300, 'total amount': 1500},
 'product4': {'number of sales': 1, 'price': 34, 'total amount': 34}}
```

5. Sets

Task 5.1: Write a program that takes as input a phrase - that does not contain any punctuation mark, that is contains only words separated by white spaces - which removes all repeated words and sorts the rest alphabetically. In the end ouput the resulting information.

Hint: You can use list comprehension and try to output its result as a string by using string methods.

Example:

```
>Input string: hello again how are you doing bye again

>Output string: again are bye doing hello how you
```

In [254]:

```python
phrase = "hello again how are you doing bye again"

def funny_function(string):

    # Split the phrase
    string = string.split()

    # Change to set to remove duplicates, change back to list to sort alphabetica
    string = sorted(list(set(string)))

    # Join all the strings
    string = " ".join(string)

    return(string)

funny_function(phrase)
```

Out[254]:

```
'again are bye doing hello how you'
```

6. JSON files

> **Task 6.1**: Create a file in your current folder called sodas_data.json that contains the information present on the sodas dataframe and where the information is ordered alphabetically according to SODAS people names.
>
> *Hint*: it may be a good idea to not use the dataframe directly to write the JSON file. Think about which types of variables are good when working with these types of files - is it lists, dictionaries, tuples, sets, etc.?

In [273]:

```
# Import json module
import json

# get a list with all dictionary items (one item is one researcher)
dictionary_items = sodas_people_dict.items()

# Sort them alphabetically
dictionary_items = sorted(dictionary_items)

# Change the list back into a dictionary
sodas_data = dict(dictionary_items)

# check alphabetic order of the keys
sodas_data.keys()
```

Out[273]:

```
dict_keys(['Anders Blok', 'Andreas Bjerre-Nielsen', 'Anna Helene Kvist Møller',
'Anna Rogers', 'Annika Isfeldt', 'Asger Andersen', 'Asger Hans Thomsen', 'Cathr
ine Valentin Kjær', 'Clara Rosa Sandbye', 'David Dreyer Lassen', 'Emilie Munch-
Gregersen', 'Esben Brøgger Lemminger', 'Eva Iris Otto', 'Frederik Carl Windfel
d', 'Frederik Hjorth', 'Friedolin Merhout', 'Germans Savčišens', 'Gregory Ead
y', 'Helene Willadsen', 'Hjalmar Alexander Bang Carlsen', 'Jonas Skjold Raascho
u-Pedersen', 'Jonathan Holm Salka', 'Kelton Ray Minor', 'Kristin Anabel Eggelin
g', 'Kristoffer Albris', 'Kristoffer Pade Glavind', 'Louise Marie Lausten', 'Ma
lene Hornstrup Jespersen', 'Mette My Madsen', 'Morten Axel Pedersen', 'Nete Sch
wennesen', 'Nicklas Johansen', 'Patrice Wangen', 'Rebecca Adler-Nissen', 'Rober
t Böhm', 'Samantha Breslin', 'Sara Vera Marjanovic', 'Sigriður Svala Jónasdótti
r', 'Simon Westergaard Lex', 'Snorre Ralund', 'Sofie Læbo Astrupgaard', 'Sophie
Smitt Sindrup Grønning', 'Sune Lehmann', 'Søren Kyllingsbæk', 'Terne Sasha Thor
n Jakobsen', 'Thyge Enggaard', 'Tobias Priesholm Gårdhus', 'Viktor Due Pederse
n', 'Yangliu Fan', 'Yevgeniy Golovchenko', 'Zoe Anna Skovby Burke'])
```

In [272]:

```
# I use the data in a dictionary form, since the structure is the same as json.
with open('data/sodas_data.json', 'w') as outfile:
    json.dump(sodas_people_dict, outfile)
```

7. Daily reflections

> **Task 7.1** 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 concepts, ideas, or topics are still unclear?
> - 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?

- I was really surprised with the difficulty of this exercise. I have some experience with python, but I had to think a lot to get through. That was really nice.

- I am getting more used to working with loops and I have learned to use enumerate() and have to counts in a loop (index and content)

If You Have Time RELAX AND TREAT YO SELF

8. Bonus exercise

> **Task 8.1**: Write a program that outputs this pattern:

```
(
( (
( ( (
( ( ( (
( ( ( ( (
( ( ( (
( ( (
( (
(
```

In [ ]: