Aske Svane Qvist

W37Mon: Flow Control and More - Exercise

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

1. Conditional Flow Control (aka If Statements)

If statements are a form of flow control. *Flow control* means writing code that controls the way data or information flows through the program. These are fundamental structures to programming. Recall that they also have a long history in how programming languages have been designed. The concepts of flow control should be recognizable outside of coding as well. For example when you go shopping you might want to buy ice cream, but only **if** it's sunny and warm outside. **else** you will buy hot chocolate. These kinds of logic come up everywhere in coding; self driving cars should go forward only **if** the light is green, items should be listed for sale in a web shop only **if** they are in stock.

Recall that in Python the conditional flow control can use three conditional statements: `if`, `elif` (else if) and `else`. The `if` and `elif` statements should be followed by a logical statement or condition that is either `True` or `False`. The code that should be executed `if` the logic is `True` is written on the lines below the `if`, and should be indented one TAB (or 4 spaces). Finally all flow control statements must end with a colon.

> **Task 1.1:** Write a **pseudocode** workflow that includes conditions, for a daily activity.
>
> *Note*: No coding is needed here.

In [1]:
```
#if hungry:
#    check the fridge
#    if "food in the fridge" == True:
#        cook
#    else:
#        go to the supermarket
#        buy food
#        cook
#    eat the food
#else:
#    pass
```

> **Task 1.2**: Imagine that we have a car that can fit 5 passengers, and we want to know, if the car can fit more passengers. Using `if` and `else`, write code that prints out "The car is full" if passengers is 5 or more, and "The car is not full yet" if

passengers are less than 5.

In [2]:
```
# Number of passengers
passengers = 5

# Include your solution in a new cell below.
# You can test your solution by changing the value of passengers, running thi
# and then running your solution with the new value
```

In [3]:
```
if passengers >= 5:
    print("The car is full")
elif passengers < 5:
    print("The car is not full yet")
```

```
The car is full
```

Task 1.3: Write a conditional flow logic which produces statements about numbers that you give it, with the following conditions:

- if the number is over 100 print "Phew, that's a big number."
- if the number is even, print "This one is even."
- if the number is even and over 100 print "Stop. I can't even."
- if the number doesn't satisfy any of these conditions, print the number.

Recall: Multiple conditions can be inserted with elif statement(s). And remember to think through your solution first before jumping to write your code!

Hint: The different if statements are executed in a given order. Think carefully about the different options you have and what they imply.

In [4]:
```
number = 102
# Most complex condition first
if (number > 100) & ((number % 2) == 0):
    print("Stop. I can't even.")
# I chose to have 'larger than 100' first
elif number > 100:
    print("Phew, that's a big number.")
# Even number
elif (number % 2) == 0:
    print("This one is even.")
else:
    print(number)
```

```
Stop. I can't even.
```

2. Lists

Recall that lists are a type of container. They are declared using square brackets e.g. `[x,y,z]` and lists can change, as in you can add elements to lists, you can change the value of a specific element, or you can remove elements. You access elements by their index, which starts at 0.

We have defined in the next cell a list that contains the the population of Denmark from 1901 - 2020 (population in 1000s). The data is from here: https://www.statbank.dk/10021 (https://www.statbank.dk/10021).

In [5]:
```
# 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]
```

Task 2.1: Take the list above. Use list methods and indices to answer the following:

- How many elements are in the list?
- What is the range (e.g. the max - min) of the first 10 items?
- What is the range (e.g. the max - min) for elements between indices 80 and 90 (i.e. 1981 to 1991)?
- What was the population in 1904 (Hint: This is the fourth element in the list)

Bonus: Can you make the cell return full length responses to each of the questions?

In [6]:
```
# How many elements are in the list?
length = len(DK_population) # 120
print(f"There are {length} elements in the list")
```

```
There are 120 elements in the list
```

In [7]:
```
# What is the range (e.g. the max - min) of the first 10 items?
first_10 = DK_population[0:9]
Range = max(first_10) - min(first_10) # 240
print(f"The range of the first 10 items is {Range}.")
```

```
The range of the first 10 items is 240.
```

In [8]:
```
# What is the range (e.g. the max - min) for elements between indices 80 and
second_sub = DK_population[80:90]
max(second_sub) - min(second_sub) # 24
print(f"The range between 1981 and 1991 is {Range}.")
```

```
The range between 1981 and 1991 is 240.
```

In [9]:
```
# What was the population in 1904 (Hint: This is the fourth element in the li
print(f"The population in 1904 was {DK_population[3]}.")
```

```
The population in 1904 was 2532.
```

Task 2.2: Run the cell below, which loads the SODAS DataFrame. Then, retrieve the names of the people at SODAS and turn them into a list.

Hint: You will want to identify the column in the DataFrame that contains the names and then use a Pandas Series method to transform it into a list. You can find the documentation for Series here (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html)

In [10]:
```
# 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)

sodas_people_df.head(2)
```

Out[10]:

	description	role	twitter	google_scholar	
0	David Dreyer Lassen is the Director of SODAS a...	SODAS steering committee	https://twitter.com/daviddlassen	https://scholar.google.dk/citations?user=aRBQc...	david.dre
1	Morten Axel Pedersen is Deputy Director of SOD...	SODAS steering committee	NaN	https://scholar.google.ca/citations?user=4vDlk...	

2 rows × 25 columns

In [11]:
```
sodas_people = sodas_people_df['name'].tolist()
sodas_people
```

Out[11]:
```
['David Dreyer Lassen',
 'Morten Axel Pedersen',
 'Rebecca Adler-Nissen',
 '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',
 'Anna Rogers',
 'Kristoffer Pade Glavind',
 'Asger Andersen',
 'Yangliu Fan',
 'Thyge Enggaard',
 'Kelton Ray Minor',
 'Sigriður Svala Jónasdóttir',
 'Terne Sasha Thorn Jakobsen',
 'Eva Iris Otto',
 'Anna Helene Kvist Møller',
 'Nicklas Johansen',
 'Germans Savčišens',
 'Snorre Ralund',
 'Annika Isfeldt',
 'Yevgeniy Golovchenko',
 'Sophie Smitt Sindrup Grønning',
 'Louise Marie Lausten',
 'Jonas Skjold Raaschou-Pedersen',
 'Zoe Anna Skovby Burke',
 'Frederik Carl Windfeld',
 'Sofie Læbo Astrupgaard',
 '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',
 'Clara Rosa Sandbye',
 'Nete Schwennesen',
 'Simon Westergaard Lex']
```

Task 2.3: How many people are in SODAS based on this list?

In [12]:
```
len(sodas_people) # 51
```

Out[12]:
```
51
```

Task 2.4: What is the index of 'Morten Axel Pedersen' in `sodas_people` ?

Hint: one way is to just look at the list and count, but this is an impractical solution for long lists and doesn't work if your program needs to know the answer. There is also a list method that helps find the index of a specific element - you can search Google or look up Python documentation to find it.

In [13]:
```
sodas_people.index("Morten Axel Pedersen")
# Index 1
```

Out[13]:
```
1
```

Task 2.5: Sometimes in lists you want to access only a few specific elements and put them in a new list. Create a list called `some_sodas_people` that contains only the 8th, 9th, and 10th elements of `sodas_people` .

In [14]:
```
some_sodas_people = sodas_people[7:10]
some_sodas_people
```

Out[14]:
```
['Andreas Bjerre-Nielsen', 'Frederik Hjorth', 'Kristoffer Albris']
```

Task 2.6: Say we want to make a broader list of people related to SDS, which then includes SODAS members and SDS students like your group. First make a new list, `sds_group` with the names of your group members, including you. Note that each name should be a string.

In [15]:
```
sds_group = ["Silvia", "Korbinian", "Isha", "Aske"]
```

Task 2.7: Now make a list called `SDS` which consist of first `sodas_people` , then `sds_group` .

> *Hint: You might be inclined to use a list method here, but don't forget that there are also operators that might be suitable.*

In [21]:
```
SDS = sodas_people + sds_group
SDS[0:5]
```

Out[21]:
```
['David Dreyer Lassen',
 'Morten Axel Pedersen',
 'Rebecca Adler-Nissen',
 'Sune Lehmann',
 'Anders Blok']
```

> **Task 2.8**: Create a list called `groups` that contains nested lists `sds_group` and a new list `sds_group2` with the names of the group beside you.

In [22]:
```
sds_group2 = ["Anton", "Annika"]
groups = [sds_group, sds_group2]
groups
```

Out[22]:
```
[['Silvia', 'Korbinian', 'Isha', 'Aske'], ['Anton', 'Annika']]
```

> **Task 2.9:** How would you access the element with your own name?
>
> *Hint: Here, we just want you to think about how indices in nested lists work. You can count the correct indices. Take as example the following code:*

In [23]:
```
xy = ['x','y']
letters = [['x','y'],['z','w']]
# counting here the xy list is inside list letters
print(letters.index(xy))
```

```
0
```

In [24]:
```
print(letters[0].index('x'))
```

```
0
```

First index the list inside the bigger list and subsequently do whatever you want with the chosen list.

In [25]:
```
# Print my name based on location
print(groups[0][3])

# Or find the index within the nested list
print(groups[0].index("Aske"))
```

```
Aske
3
```

Task 2.10: Maybe sometime in the future you become the Head of Studies of the SDS program and take over from Andreas Bjerre-Nielsen. Replace Andreas's name with your name (as a string) in `sodas_people` .

Hint: First find the index with Andreas's name

In [26]:
```
sodas_people.index("Andreas Bjerre-Nielsen")
sodas_people[7] = "Aske Svane Qvist"
```

In [27]:
```
sodas_people[0:8]
```

Out[27]:
```
['David Dreyer Lassen',
 'Morten Axel Pedersen',
 'Rebecca Adler-Nissen',
 'Sune Lehmann',
 'Anders Blok',
 'Søren Kyllingsbæk',
 'Robert Böhm',
 'Aske Svane Qvist']
```

Task 2.11: What are the values of `SDS` ? Where is your name and why?

In [28]:
```
print(SDS)
SDS.index("Aske") # at index 54 as I appended our names to the end of the lis
```

```
['David Dreyer Lassen', 'Morten Axel Pedersen', 'Rebecca Adler-Nissen', 'Su
ne Lehmann', 'Anders Blok', 'Søren Kyllingsbæk', 'Robert Böhm', 'Andreas Bj
erre-Nielsen', 'Frederik Hjorth', 'Kristoffer Albris', 'Friedolin Merhout',
'Gregory Eady', 'Samantha Breslin', 'Hjalmar Alexander Bang Carlsen', 'Patr
ice Wangen', 'Kristin Anabel Eggeling', 'Helene Willadsen', 'Mette My Madse
n', 'Anna Rogers', 'Kristoffer Pade Glavind', 'Asger Andersen', 'Yangliu Fa
n', 'Thyge Enggaard', 'Kelton Ray Minor', 'Sigriður Svala Jónasdóttir', 'Te
rne Sasha Thorn Jakobsen', 'Eva Iris Otto', 'Anna Helene Kvist Møller', 'Ni
cklas Johansen', 'Germans Savčišens', 'Snorre Ralund', 'Annika Isfeldt', 'Y
evgeniy Golovchenko', 'Sophie Smitt Sindrup Grønning', 'Louise Marie Lauste
n', 'Jonas Skjold Raaschou-Pedersen', 'Zoe Anna Skovby Burke', 'Frederik Ca
rl Windfeld', 'Sofie Læbo Astrupgaard', 'Malene Hornstrup Jespersen', 'Vikt
or Due Pedersen', 'Cathrine Valentin Kjær', 'Esben Brøgger Lemminger', 'Sar
a Vera Marjanovic', 'Asger Hans Thomsen', 'Emilie Munch-Gregersen', 'Tobias
Priesholm Gårdhus', 'Jonathan Holm Salka', 'Clara Rosa Sandbye', 'Nete Schw
ennesen', 'Simon Westergaard Lex', 'Silvia', 'Korbinian', 'Isha', 'Aske']
```

Out[28]: 54

3. Conditional Flow Control in Practice

There are a variety of applications of conditional logic when working in Python, such as when you are collecting data or conducting quality assurance on existing data. Next, we'll have two tasks that introduce potential applications to give you an opportunity to practice more meaningful uses and think through when you might want to use flow control.

When trying to load files in your own computer into a pandas dataframe in python you may encounter different types of errors. In the following task, you will be using `if` statements and booleans to check whether a file is present in an existing folder, specifically in the one you have stored your Jupyter Notebook in.

To set up for the task, run the next cell which returns a boolean value indicating whether a file called "file_name_to_check.txt" is present in the directory where your Jupyter Notebook is stored. If the file is present in the folder a `True` value will be stored in the variable called `boolean`. Otherwise, a `False` value will be stored.

In [29]:
```
import os.path
fname = "file_name_to_check.txt"
boolean = os.path.isfile(fname)
boolean
```

Out[29]: False

Task 3.1: Use the `boolean` variable you just created and write a flow control logic that prints a message to describing whether the file exists in your folder or not.

Note: If you are uncertain about the code above or are having problems with it, you can also just imagine that the file does not exists and make defining the variable `boolean` part of your solution.

In [30]:
```
if boolean == True:
    print(f"The file {fname} exists in the folder.")
else:
    print(f"The file {fname} does not exist in the folder.")
```

```
The file file_name_to_check.txt does not exist in the folder.
```

Now let's get more practice with string methods and how to combine them with if statements.

Task 3.2: Define the string `string = "I-love-programming-in-python"` and use string methods to do the following:

1. Replace the characters "-" with spaces (" ").
2. Convert the string to a title.

In [31]:
```
# Define string
string = "I-love-programming-in-python"
# Replace dashes with spaces
string = string.replace("-", " ")
# Change to title (all words start with a capital letter)
string = string.title()
string
```

Out[31]:
```
'I Love Programming In Python'
```

Task 3.3: Combine string methods and conditional flow control into code that does the following:

- Check whether the string is a title and the number of "-" characters is equal to 0.
- If both conditions are met, store each word of the string into a separate element in a list. Plus, print the length of both the list and the string. Explain why they have the same lenght or not.
- If the conditions are not met, print "Something went wrong with the string methods."

In [32]:
```
if (string.istitle() == True) & (string.count('-') == 0):
    words = string.split()
    print(f"the number of characters in the list 'string' is {len(string)}.")
    print(f"the number of words it contains is {len(words)}.")
else:
    print("Something went wrong with the string methods.")
```

```
the number of characters in the list 'string' is 28.
the number of words it contains is 5.
```

4. Practicing Python Built-in Functions and String Methods

As you know by now, Python like any other language requires practice and repitition to acquire ease and fluency in using it. The next couple of tasks are intended to give you some opportunity to do just that with a specific focus on built-in functions and more string methods.

> **Task 4.1**: Pick three built-in functions and apply them to an object of your choosing. For instance, you could try the following functions on lists:

```
list_of_numbers = [0, 4, 3, -7]

max(list_of_numbers)

min(list_of_numbers)
```

> *Hint: You can find documentation on the built-in functions in Python here (https://www.w3schools.com/python/python_ref_functions.asp).*

In [33]:
```
sds_group = ["Silvia", "Korbinian", "Isha", "Aske"]
```

In [34]:
```
# first built-in function
sds_group.append("Anders")
# second built-in function
letters = "".join(sds_group)
print(f"the total number of letters in the list is {len(letters)}.")
# third built-in function b
print(f"The person with the longest name is {max(sds_group, key=len)}.")
```

```
the total number of letters in the list is 29.
The person with the longest name is Korbinian.
```

> **Task 4.2**: Next, check out the String Methods documentation (https://www.w3schools.com/python/python_ref_string.asp). Then, define a string and try out at least two string methods that we haven't used in class before. As an

example, you could use the replace method.

```
string = "Hello"

string.replace("ello", "i")
```

In [35]:
```
# define string
string = "beast"
# first string method
string.replace("b", "f")

# second string method
string.upper()
```

Out[35]:
```
'BEAST'
```

5. Tracing and Debugging

While you are getting steadly more at ease with working in Python, error messages will remain a constant companion. Therefore, dealing with error messages and resolving them is a key part of writing and working with code. While you will gain most of the knowledge and facility of how to do this tracing and debugging while practicing coding yourself, there are some best practices that can help you develop this facility. These are:

1. Treat error messages like guide posts not stop signs–they are intended to help you solve an error not discourage you from coding.
2. Read the guide posts carefully, starting from the last line describing the error, then looking at the last marker– ^ or `---->` –to identify the location of the error.
3. Remember and make use of the layers of support in this class and the overall programming community:
   A. Consult your readings, earlier notebooks, own comments, and Python documentation (https://docs.python.org/3/index.html)
   B. Look up error messages in only forums (such as Stack Overflow (https://stackoverflow.com/questions) and the internet more broadly)
   C. Involve your Base Camp group
   D. Draw on the TAs during exercise sessions
   E. Pose the problem to instructors on the Absalon Discussion Forum (https://absalon.ku.dk/courses/45629/discussion_topics/282640)
4. Sometimes running a section of code with an error will get stuck in the memory (cache) of your Python session. If you are confident that you have resolved the error in your code but it persists, try restarting your session by clicking 'Kernel' > 'Restart'. *Note:* This will also clear everything you did in earlier cells from the memory (cache).
5. Other times the errors will cause a process to run endlessly without actually raising an error. You can determine this by looking at the dot on the top right corner, if it is solid Python is busy, if it is hollow Python is idle. If Python is busy for an unreasonably long amount of time, you can interrupt the process by clicking 'Kernel' > 'Interrupt'. *Note:* This will also likely require you to run everything you did so far over again.

Task 5.1: Look at the three cells and their error messages below. For each,

- write out the error message
- identify the location of the error
- revise the code *in any way that resolves the error* and rerun the cell
- write a one to three sentence reflection on how you approached and resolved the error

Task 5.1.1

In [36]:
```
10 * (1/0)
# It says devision by zero - that is not posssible
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-36-62037aa55103> in <module>
----> 1 10 * (1/0)
      2 # It says devision by zero - that is not posssible

ZeroDivisionError: division by zero
```

Task 5.1.2

In [37]:
```
100 %% 3
# Invalid syntax - only one %
```

```
  File "<ipython-input-37-58c9eaf87cb4>", line 1
    100 %% 3
         ^
SyntaxError: invalid syntax
```

In [38]:
```
100%3
```

Out[38]: 1

Task 5.1.3

In [39]:
```
a_title = 'Honorable'

a_title.istitl()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-39-9fe068237cef> in <module>
      1 a_title = 'Honorable'
      2
----> 3 a_title.istitl()

AttributeError: 'str' object has no attribute 'istitl'
```

In [40]:
```
# Nothing is called 'istitl' - an 'e' is lacking.
a_title.istitle()
```

Out[40]: True

Task 5.1.4

In [41]:
```
a_list = ['first', 'second', 'last']

a_list[3]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-41-acdf254a7b7b> in <module>
      1 a_list = ['first', 'second', 'last']
      2
----> 3 a_list[3]

IndexError: list index out of range
```

In [42]:
```
# Out of range - index starts at 0.
a_list[2]
```

Out[42]: 'last'

Task 5.1.5

In [43]:
```
if a_list[-1] == 'last':
print('The last element is ' + a_list[-1] + '.')
else:
    print('The last element is not ' + a_list[-1] + '.')
```

```
  File "<ipython-input-43-d793faf1722e>", line 2
    print('The last element is ' + a_list[-1] + '.')
    ^
IndentationError: expected an indented block
```

In [44]:
```
# Expected an indented block
if a_list[-1] == 'last':
    print('The last element is ' + a_list[-1] + '.')
else:
    print('The last element is not ' + a_list[-1] + '.')
```

```
The last element is last.
```

6. 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 have done some coding in python before, so this was very basic :)

7. *If You Have Time* Become a Python CEO and CFO

> **Task 7.1**: Imagine you run you own company. You would like now to create a program that outputs if the average monthly income is as you expected it to be, greater or less and if applicable the difference between them. Moreover, assume your target monthly income is 40000 DKK, but your income fluctuates throughout the year in the following way:

Month	Income
January	45000
February	55000
March	65000
April	35000
May	45000
June	67000
July	23000

Month	Income
August	56000
September	30000
October	33000
November	32000
December	55000

Now, try writing down out code that tests how your income compares to your goals and prints the results.

Note: think about calculating the expected and real income first.

In [45]:
```
# Income
income = [45000, 55000,65000,35000,45000,67000,23000,56000,30000,33000,32000,
income_avg = round(sum(income) / len(income),2)

# Expected
expected = 40000

# calculating the difference
abs(round(income_avg - expected,2))
```

Out[45]: 5083.33

In [ ]:

In [ ]: