HW1_pds_2023_V2

September 28, 2023

1 Start Here

[Please Complete the following form with your details]

Student Name - Devora dos Santos Cavaleiro Student id - 20230974 Contact e-mail - devoracavaleiro27@hotmail.com

Student Name - Carlos Eduardo Rodrigues da Costa Student id - 20230543 Contact e-mail - carlos_edu75@hotmail.com

2 Part 1 - Variable Declaration and Manipulation

2.1 Exercise I - of Lists and Random numbers

1: Declare a variable X that stores a list of 100 integers randomly sampled between -100 and 100. Note: You are not allowed to use third party libraries – such as Numpy, Pandas, or Scipy – in this exercise.

[1]:
```
import random

X = []


for numb in range(0,100): # Range will define the lenght of the list, that is␣
  ↪100
    random_number = random.randint(-100, 100) # Store the random numbers␣
  ↪between -100 and 100, both included, on random_number variable
    X.append(random_number) # Add each random number to the list X
print(X)
```

```
[54, -88, -52, 89, 16, 26, 95, -58, -73, -34, 81, 17, -82, -59, 65, 62, -24, 22,
-91, 28, 80, 19, 75, 80, 9, -50, -72, -40, 80, -50, -25, 86, -86, 27, 25, 80,
-15, 88, -2, -83, -18, 82, 79, -40, -61, -44, 96, 47, -84, 68, -90, 89, -100,
56, 94, -86, -93, -30, 19, 28, -70, -32, 89, 72, 96, -17, -96, 20, -69, -81,
-45, 49, -27, 85, -20, -71, -63, 56, -81, -53, -59, 6, 43, 77, -43, 51, -68,
-18, 85, -67, 30, 15, -67, 63, -18, -55, 71, -41, -17, -30]
```

2: How many odd numbers are in the list X?

[2]:
```
list_odd = []


for numb in X:
    if numb % 2 != 0: # verify which values are odd numbers
        list_odd.append(numb) # add odd numbers to list_odd
print("There are", len(list_odd), "odd numbers.")
```

```
There are 48 odd numbers.
```

2.1 Check if there are as many even numbers as there are odd numbers, else discard the list and generate a new one.

[3]:
```
list_even = []


for numb in X:
    if numb % 2 == 0: # verify which values are even numbers
        list_odd.append(numb) #add even numbers to list_even
```

[4]:
```
if len(list_even) == len(list_odd): # Verify if the lenght of the odd and even␣
 ↪number list are the same, so it will have the same number of elements on both
    print("There are as many even numbers as there are odd numbers.")
else: # Create a new list, in the same terms as exercise 1
    for numb in range(0,100):
        random_number = random.randint(-100, 100)
        X.append(random_number)
    print("The new list X is", X)
```

```
The new list X is [54, -88, -52, 89, 16, 26, 95, -58, -73, -34, 81, 17, -82,
-59, 65, 62, -24, 22, -91, 28, 80, 19, 75, 80, 9, -50, -72, -40, 80, -50, -25,
86, -86, 27, 25, 80, -15, 88, -2, -83, -18, 82, 79, -40, -61, -44, 96, 47, -84,
68, -90, 89, -100, 56, 94, -86, -93, -30, 19, 28, -70, -32, 89, 72, 96, -17,
-96, 20, -69, -81, -45, 49, -27, 85, -20, -71, -63, 56, -81, -53, -59, 6, 43,
77, -43, 51, -68, -18, 85, -67, 30, 15, -67, 63, -18, -55, 71, -41, -17, -30,
-34, -13, -85, 28, 48, -43, 66, -83, 84, -40, 15, -32, -20, -89, -69, -46, -6,
-93, 32, -45, 69, -22, 47, -67, -95, -30, 27, 66, 47, 81, -60, -74, -32, -6,
-91, -82, -57, 18, -94, -62, 66, 29, 81, -19, -49, -49, 8, -27, -50, 67, 12,
-36, -27, 52, 89, 90, -33, 8, -83, -100, -80, 93, -56, -77, 98, 93, 2, 86, 29,
74, -93, 100, -80, 71, 19, -66, -2, -27, -86, -22, 74, -45, -80, 40, 94, 1, -57,
97, 85, 32, -27, -79, 1, 15, -66, -87, -34, -51, -62, 25]
```

2.2 Create a pipeline to automatically perform the tasks above such that you avoid having to run the cells multiple times to reach the desired outcome.

[5]:
```
import random

def random_lista():
    r_list = []
    #Choose random 100 numbers between [-100,100] with -100 and 100 included.
    for i in range(100):
        r_list.append(random.randint(-100,100))

    return r_list

X = random_lista()
print(X)
```

```
[-55, 74, -70, -38, 21, 96, -44, -46, -21, 64, -26, 73, -99, 44, 63, -19, 87,
13, 7, -21, -73, -80, -6, -31, 27, 36, 33, -35, -25, 34, 24, 82, -97, -38, 55,
36, 1, -69, -39, 58, 17, -51, 7, -56, -71, 57, 9, 34, 65, 14, -46, 53, 52, 40,
23, -80, -67, 64, -4, -74, 70, 97, -84, -39, -56, -18, 17, -25, 65, -41, 40,
-73, -87, 46, 25, -21, 73, -5, 20, -5, -36, 56, -42, -40, 45, -37, 42, 33, -57,
-70, 94, -42, -80, 21, 69, -31, 61, 25, 94, -57]
```

[6]:
```
# Create a full automatizate cicle untill the number of "odd numbers" and "even␣
 ↪numbers" match.
while True:
    X = random_lista() # Call function defined in Exercice 2,2
    even_numbers_count = 0 # Initialize counters for even numbers
    odd_numbers_count = 0 # Initialize counters for odd numbers

    for number in X:
        if number % 2 != 0:
            odd_numbers_count += 1
        else:
            even_numbers_count += 1

    if even_numbers_count == odd_numbers_count: # If the number of even and odd␣
 ↪are equal, print and stop the loop
        print("Total number of odd:", odd_numbers_count)
        print("Total number of even:", even_numbers_count)
        break
```

```
Total number of odd: 50
Total number of even: 50
```

3: Print the number of digits that the 5th and 100th element of the list have. *Note: For instance, the number 1 contains one digit, the number 10 contains two digits, the number -2 contains one digit.*

[7]:
```
# Considere the list X

# Convert the integer in string to get the lenght of the string. The function␣
 ↪abs() allows us to get the absolute value
# to get rid of the negative sign
```

```
five_element = X[4]
digit_counter_5 = len(str(abs(five_element)))

hundred_element = X[99]
digit_counter_100 = len(str(abs(hundred_element)))

print("The 5th element has", digit_counter_5, "digits.")
print("The 100th element has", digit_counter_100, "digits.")
```

```
The 5th element has 2 digits.
The 100th element has 2 digits.
```

4: Is the sum total of all the numbers in the list even or odd?

[8]:
```
sum_list = sum(X)
if sum_list % 2 == 0:
    print("The sum total of all the numbers in the list is even.")
else:
    print("The sum total of all the numbers in the list  is odd.")
```

```
The sum total of all the numbers in the list is even.
```

5: What is the average of the list X?

[9]:
```
mean = sum(X)/len(X)
print("The average of the list X is", mean)
```

```
The average of the list X is -5.26
```

5.1. What is the population standard deviation?

[10]:
```
# sd is ((sum(X-mean)**2)/len(X))**(1/2)
soma = 0

for num_sd in X:
    num_sd = num_sd - mean
    num_sd2 = num_sd**2
    soma += num_sd2
divison = soma/len(X)
sd = divison**(1/2)
print("The population standard deviation is", sd)
```

```
The population standard deviation is 57.65893165850369
```

6: Sort list X in descending order and store the result in variable Xsort.

[11]:
```
Xsort = sorted(X, reverse=True) # The function sorted() by default sort in␣
 ↪ascending order, so to do the inverse we put reverse = True
print(Xsort)
```

```
[99, 95, 95, 94, 90, 90, 87, 85, 76, 71, 65, 65, 63, 62, 61, 61, 60, 60, 59, 58,
56, 53, 48, 46, 46, 43, 43, 40, 39, 37, 34, 33, 30, 30, 28, 26, 23, 21, 19, 15,
15, 12, 11, 10, 6, -3, -5, -5, -6, -8, -10, -13, -14, -15, -15, -17, -19, -19,
-21, -22, -26, -27, -28, -32, -40, -41, -41, -42, -48, -48, -49, -50, -50, -51,
-54, -57, -58, -59, -60, -60, -62, -66, -68, -70, -72, -72, -74, -77, -85, -86,
-87, -89, -91, -93, -93, -94, -95, -99, -100, -100]
```

6.1 Then replace each value in Xsort with index i as the sum of the values with index i-1 and i.

Note: Consider that Xsort[-1] = 0.

[12]:
```
for i in range(1,len(Xsort)): # Xsort[-1] = 0, so the 1st position won't change
    Xsort[i] = Xsort[i-1] + Xsort[i]
print(Xsort)
```

```
[99, 194, 289, 383, 473, 563, 650, 735, 811, 882, 947, 1012, 1075, 1137, 1198,
1259, 1319, 1379, 1438, 1496, 1552, 1605, 1653, 1699, 1745, 1788, 1831, 1871,
1910, 1947, 1981, 2014, 2044, 2074, 2102, 2128, 2151, 2172, 2191, 2206, 2221,
2233, 2244, 2254, 2260, 2257, 2252, 2247, 2241, 2233, 2223, 2210, 2196, 2181,
2166, 2149, 2130, 2111, 2090, 2068, 2042, 2015, 1987, 1955, 1915, 1874, 1833,
1791, 1743, 1695, 1646, 1596, 1546, 1495, 1441, 1384, 1326, 1267, 1207, 1147,
1085, 1019, 951, 881, 809, 737, 663, 586, 501, 415, 328, 239, 148, 55, -38,
-132, -227, -326, -426, -526]
```

2.2 Exercise II - we have a gamer in the room

7: Consider the dictionaries *purchases* and *clients* that are declared in the cell below. Create a list with the names of the clients who bought more than one videogame. Print the List.

[13]:
```
purchases = {

    "1539":"Red dead redemption II",
    "9843":"GTA V,FarCry 5, watchdogs II",
    "8472":"Canis Canem Edit",
    "3874":"Watchdogs II,South Park: The Stick of Truth",
    "5783":"AC: The Ezio Collection, watchdogs ii",
    "9823":"For Honor,The Forest,South Park: The Fractured but whole"

}
```

[14]:
```
clients = {

    "1539":"Rick Sanchez",
    "9843":"Morty Smith",
    "8472":"Eve Polastri",
    "3874":"Mildred Ratched",
    "5783":"Alex Vause",
    "9823":"Sheldon Cooper"

}
```

[15]:
```
clients_with_more_games = []

for client_key in clients:
    games = purchases[client_key].split(",") # return a list with game names
    if (len(games) > 1): # list games > 1, bought more that one game
        clients_with_more_games.append(clients[client_key]) # add to the list␣
 ↪clients_with_more_games the values of key in clients with the buys.

print(clients_with_more_games)
```

```
['Morty Smith', 'Mildred Ratched', 'Alex Vause', 'Sheldon Cooper']
```

7.1 What is the name of the client that bought more videogames?

Tip: You will want to check the methods associated with string manipulation. See the link: https://python-reference.readthedocs.io/en/latest/docs/unicode/index.html

[16]:
```
best_buyer = []
maximum = 0
client = ""

for client_key in clients:
    games = purchases[client_key].split(",") # return a list with game names
    if len(games) > maximum: # verify that this len(games) is greater than the␣
 ↪current maximum
        maximum = len(games) #if true substitutes
        client = clients[client_key]
print(client)
```

```
Morty Smith
```

7.2 What is the name of the most popular videogame?

[17]:
```
maximum = 0
list_games = []
pop_game = ""

for keys in purchases:# adds to list_games all games
    games = purchases[keys].split(",")
    for game in games:
        list_games.append(game.lower().strip()) # all in lower case and without␣
 ↪spaces, count case sensitive and space sensitive too.

for game in list_games:
    game_count = list_games.count(game) #counts each game
    if game_count > maximum: # verify that the count of each game is greater␣
 ↪than the current max
        maximum = game_count #if true substitutes
        pop_game = game
```

```
print("The most popular videogame is", pop_game)
```

```
The most popular videogame is watchdogs ii
```

3 Part 2 - Data loading and analysis

3.1 Exercise I - Alice what do you have to say?

8: Load the Alice text file into a variable called Alice.

Note: Use a relative filepath in relation to the location of your notebook.

[18]:
```
# Using open() it's possible to push a file into a variable called Alice, first␣
↪parameter, defines the path were the files is, secound parameter defines␣
↪what action i'm doing r,w,c...(read,writing,copying...)
# We're using the with to automatically close the file
with open("./alice.txt", "r") as file:
    Alice = file.read() # Create a object data with all information from Alice

Alice
```

[18]:
```
'Alice was beginning to get very tired of sitting by her sister on the bank, and
of having nothing to do: once or twice she had peeped into the book her sister
was reading, but it had no pictures or conversations in it, <<and what is the
use of a book>>. thought Alice <<without pictures or conversation?>>.'
```

9: Create a list in which each element is a word from the file *Alice*. Store that list in a variable called *wAlice*.

Note: You will need to do some text parsing here. In particular to split the sentences into words. It is also a good practice to normalize words so that words "Hello" and "hello" become identical, by making all letters lower case.

Tip: check the following links for a discussion on regular expressions. Also consult the methods available for string manipulation.

https://docs.python.org/3/library/re.html

https://stackoverflow.com/questions/1276764/stripping-everything-but-alphanumeric-chars-from-a-string-in-python

[19]:
```
import re

# get rid of all non alphanumeric characters and non whitespaces replacing it␣
↪to nothing (empty string).
new_str = re.sub(r"[^\w\s]", "", Alice) # ^ - negative (\w - alphanumeric␣
↪characters and underscore, \s - whitespaces)

print(new_str)
```

```
Alice was beginning to get very tired of sitting by her sister on the bank and
of having nothing to do once or twice she had peeped into the book her sister
was reading but it had no pictures or conversations in it and what is the use of
a book thought Alice without pictures or conversation
```

[20]:
```python
wAlice = new_str.lower().split()
print(wAlice)
```

```
['alice', 'was', 'beginning', 'to', 'get', 'very', 'tired', 'of', 'sitting',
'by', 'her', 'sister', 'on', 'the', 'bank', 'and', 'of', 'having', 'nothing',
'to', 'do', 'once', 'or', 'twice', 'she', 'had', 'peeped', 'into', 'the',
'book', 'her', 'sister', 'was', 'reading', 'but', 'it', 'had', 'no', 'pictures',
'or', 'conversations', 'in', 'it', 'and', 'what', 'is', 'the', 'use', 'of', 'a',
'book', 'thought', 'alice', 'without', 'pictures', 'or', 'conversation']
```

Using the list *wAlice* answer the following questions: **10**: How many words are in the file Alice.txt?

[21]:
```python
print("There are", len(wAlice), "words in Alice.txt") # Calculate the lenght of␣
 ↪the previous list"
```

```
There are 57 words in Alice.txt
```

11: What is the longest and smallest word in the text file?

Note: Length in this case is measured in terms of the number of characters.

[22]:
```python
print("The longest word is", max(wAlice, key=len))
print("The smallest word is", min(wAlice, key=len))
```

```
The longest word is conversations
The smallest word is a
```

12: Delete all the repeated words from *wAlice*.

[23]:
```python
alice_word_set = set(wAlice) # sets don't accept repetitions
alice_word_set
```

[23]:
```
{'a',
 'alice',
 'and',
 'bank',
 'beginning',
 'book',
 'but',
 'by',
 'conversation',
 'conversations',
 'do',
 'get',
 'had',
 'having',
 'her',
 'in',
 'into',
 'is',
 'it',
 'no',
 'nothing',
 'of',
 'on',
 'once',
 'or',
 'peeped',
 'pictures',
 'reading',
 'she',
 'sister',
 'sitting',
 'the',
 'thought',
 'tired',
 'to',
 'twice',
 'use',
 'very',
 'was',
 'what',
 'without'}
```

12.1:How many different words does the text contain?

[24]:
```
print("There are", len(alice_word_set), "different words in the text.") #␣
 ↪Calculate the lenght of the previous list, that has no repetions."
```

```
There are 41 different words in the text.
```

3.2 Exercise II - of Countries I Love

Consider the list countries in the cell below. It consists of a list of the 3-digit ISO codes of a set of countries of interest.

[25]:
```
# We changed this list a little. We Fixed the iso code for Israel and Denmark
countries = [

('PRT','Portugal','Europe'),
('ISR','Israel','Asia'),
('COL','Colombia','South America'),
('JPN','Japan','Asia'),
('RUS','Russia','Europe'),
('DNK','Denmark','Europe'),
('NOR','Norway','Europe')

]
```

Now consider the file *cdata.csv* that you should download.

The file contains several information about countries, and is organized as follows: 1. Column 1 is the 3-digit **ISO Code** 2. Column 2 is the **Full Name** of the country 3. Column 3 is the **Continent** of the country 4. Column 4 is the **Population Size** in 2010 5. Column 5 is the **GDP per capita** in 2010

13: Using the Library CSV from Python STL, load the file *cdata.csv* into an object called *raw*.

[26]:
```
import csv

# We also did some changes in the csv file.
# We added a whitespace for south america and north america and also fixed the␣
↪continent of Russia (it was Asia before the fix)
with open('./cdata.csv', 'r') as file:
    reader = csv.reader(file)
    raw = list(reader)
```

13.1: Create a dictionary called *cData* in which the key corresponds to the 3-digit ISO Code and the value is a tuple with the information contained in the 2nd to the 5th of column of the cdata file.

[27]:
```
cData = {}
# Here we need to be careful because the first row of our data is the header.
# So we need to skip it
for row in raw[1:]:
    iso_code = row[0].upper()
    data = tuple(row[1:5])
    cData[iso_code] = data

cData
```

[27]:
```
{'AGO': ('Angola', 'Africa', '23.369131088256836', '5988.534997149481'),
 'BDI': ('Burundi', 'Africa', '8.766929626464844', '731.4232803265862'),
 'BEN': ('Benin', 'Africa', '9.199258804321289', '1919.9969479963038'),
 'BFA': ('Burkina Faso', 'Africa', '15.605216979980469', '1327.2165314775982'),
 'BWA': ('Botswana', 'Africa', '2.0148661136627197', '12256.14159052444'),
 'CAF': ('Central African Republic',
  'Africa',
  '4.4485249519348145',
  '865.4421867263901'),
 'CMR': ('Cameroon', 'Africa', '19.970495223999023', '2684.972890247272'),
 'COD': ('Democratic Republic of the Congo',
  'Africa',
  '64.52326202392578',
  '634.9713963013496'),
 'COG': ('Republic of the Congo',
  'Africa',
  '4.386693000793457',
  '4979.990496069269'),
 'COM': ('Comoros', 'Africa', '0.6896920204162598', '2292.997969425929'),
 'CPV': ('Cape Verde', 'Africa', '0.5023840069770813', '5644.821253339075'),
 'DJI': ('Djibouti', 'Africa', '0.8511459827423096', '2124.4420625972525'),
 'DZA': ('Algeria', 'Africa', '36.117637634277344', '12590.224223813591'),
 'EGY': ('Egypt', 'Africa', '84.10760498046875', '9148.839753298033'),
 'ETH': ('Ethiopia', 'Africa', '87.70266723632812', '1100.1831875876223'),
 'GAB': ('Gabon', 'Africa', '1.6402100324630737', '11961.168226448879'),
 'GHA': ('Ghana', 'Africa', '24.512104034423828', '3931.2918319728856'),
 'GIN': ('Guinea', 'Africa', '10.794170379638672', '1742.7346742167783'),
 'GMB': ('Gambia', 'Africa', '1.692149043083191', '2681.8586168502143'),
 'GNB': ('Guinea-Bissau',
  'Africa',
  '1.5558799505233765',
  '1387.2497310879905'),
 'GNQ': ('Equatorial Guinea',
  'Africa',
  '0.9511039853096008',
  '31416.887137501908'),
 'KEN': ('Kenya', 'Africa', '41.35015106201172', '2484.0349094725366'),
 'LBR': ('Liberia', 'Africa', '3.948124885559082', '786.6702306741524'),
 'LSO': ('Lesotho', 'Africa', '2.040550947189331', '2432.7976546880823'),
 'MAR': ('Morocco', 'Africa', '32.409637451171875', '6421.937677907418'),
 'MDG': ('Madagascar', 'Africa', '21.151639938354492', '1459.9156124535612'),
 'MLI': ('Mali', 'Africa', '15.075084686279297', '1873.2809618195865'),
 'MOZ': ('Mozambique', 'Africa', '24.221405029296875', '969.2307622526259'),
 'MRT': ('Mauritania', 'Africa', '3.6095430850982666', '3082.647028528842'),
 'MUS': ('Mauritius', 'Africa', '1.2479549646377563', '15178.325920597827'),
 'MWI': ('Malawi', 'Africa', '15.167095184326172', '972.0456815116237'),
 'NAM': ('Namibia', 'Africa', '2.1731700897216797', '7689.171239820006'),
 'NER': ('Niger', 'Africa', '16.42557716369629', '845.8693682212395'),
 'NGA': ('Nigeria', 'Africa', '158.57826232910156', '5186.304307542348'),
 'RWA': ('Rwanda', 'Africa', '10.246842384338379', '1379.6993252327009'),
 'SDN': ('Sudan', 'Africa', '34.385963439941406', '3608.7856176907358'),
 'SEN': ('Senegal', 'Africa', '12.916229248046875', '2741.121456198507'),
 'SLE': ('Sierra Leone', 'Africa', '6.4587202072143555', '1161.8796942144647'),
 'SWZ': ('Swaziland', 'Africa', '1.2028429508209229', '7042.524359035093'),
 'SYC': ('Seychelles', 'Africa', '0.0914049968123436', '17960.353075613308'),
 'TCD': ('Chad', 'Africa', '11.887202262878418', '1880.3748762799714'),
 'TGO': ('Togo', 'Africa', '6.5029520988464355', '1222.8150529133295'),
 'TUN': ('Tunisia', 'Africa', '10.639930725097656', '10647.834980284626'),
 'TZA': ('Tanzania', 'Africa', '44.82893753051758', '1979.1718644815185'),
 'UGA': ('Uganda', 'Africa', '33.91513442993164', '1723.5943355766267'),
 'ZAF': ('South Africa', 'Africa', '51.58466339111328', '11388.640515995063'),
 'ZMB': ('Zambia', 'Africa', '13.850032806396484', '2870.8872656342314'),
 'ZWE': ('Zimbabwe', 'Africa', '14.08631706237793', '1479.0305839163732'),
 'ARE': ('United Arab Emirates',
  'Asia',
  '8.270684242248535',
  '59707.412565389604'),
 'ARM': ('Armenia', 'Asia', '2.8773109912872314', '8222.945234202914'),
 'AZE': ('Azerbaijan', 'Asia', '9.03245735168457', '12947.102198682365'),
 'BGD': ('Bangladesh', 'Asia', '152.14910888671875', '2411.1020691099325'),
 'BHR': ('Bahrain', 'Asia', '1.2408620119094849', '37045.81310919623'),
 'BRN': ('Brunei', 'Asia', '0.3886620104312897', '67320.89957245934'),
 'BTN': ('Bhutan', 'Asia', '0.7276409864425659', '7235.18974818538'),
 'CHN': ('China', 'Asia', '1359.755126953125', '9337.290772677254'),
 'CYP': ('Cyprus', 'Asia', '0.829446017742157', '28046.373877741087'),
 'GEO': ('Georgia', 'Asia', '4.231660842895508', '7966.602756952667'),
 'HKG': ('Hong Kong', 'Asia', '7.02522087097168', '41687.946418899235'),
 'IND': ('India', 'Asia', '1230.980712890625', '4357.0597360582315'),
 'IRN': ('Iran', 'Asia', '74.56751251220703', '17328.457882894658'),
 'IRQ': ('Iraq', 'Asia', '30.7627010345459', '9344.543768350652'),
 'ISR': ('Israel', 'Asia', '7.42595911026001', '28638.9037385305'),
 'JOR': ('Jordan', 'Asia', '7.182390213012695', '9351.387283068141'),
 'JPN': ('Japan', 'Asia', '128.5518798828125', '36595.63364058582'),
 'KAZ': ('Kazakhstan', 'Asia', '16.398975372314453', '17908.83993251286'),
 'KGZ': ('Kyrgyzstan', 'Asia', '5.422337055206299', '3382.4639059259293'),
 'KHM': ('Cambodia', 'Asia', '14.30873966217041', '2330.127207632253'),
 'KOR': ('South Korea', 'Asia', '49.5528564453125', '31589.705161145695'),
 'KWT': ('Kuwait', 'Asia', '2.9980831146240234', '67029.51680017101'),
 'LAO': ('Laos', 'Asia', '6.246273994445801', '4316.969858587265'),
 'LBN': ('Lebanon', 'Asia', '4.337141036987305', '18025.24893894264'),
 'LKA': ('Sri Lanka', 'Asia', '20.198352813720703', '8390.429701766443'),
 'MAC': ('Macau', 'Asia', '0.536969006061554', '91982.39517540816'),
 'MDV': ('Maldives', 'Asia', '0.36451101303100586', '12140.803390739566'),
 'MMR': ('Burma', 'Asia', '50.1558952331543', '3422.241340266976'),
 'MNG': ('Mongolia', 'Asia', '2.7126500606536865', '7670.714034382584'),
 'MYS': ('Malaysia', 'Asia', '28.112289428710938', '17913.164410782432'),
 'NPL': ('Nepal', 'Asia', '27.023136138916016', '1996.1961554516242'),
 'OMN': ('Oman', 'Asia', '3.0414600372314453', '40472.674942509126'),
 'PAK': ('Pakistan', 'Asia', '170.5601806640625', '4171.416562353057'),
 'PHL': ('Philippines', 'Asia', '93.72662353515625', '5391.233965240031'),
 'QAT': ('Qatar', 'Asia', '1.7796759605407715', '123128.40152283433'),
 'SAU': ('Saudi Arabia', 'Asia', '27.425676345825195', '42331.65885721999'),
 'SGP': ('Singapore', 'Asia', '5.074252128601074', '58618.43503468222'),
 'SYR': ('Syria', 'Asia', '21.01883316040039', '5700.329438992399'),
 'THA': ('Thailand', 'Asia', '67.20880889892578', '12496.244186726299'),
 'TJK': ('Tajikistan', 'Asia', '7.641630172729492', '2784.5145587763627'),
 'TKM': ('Turkmenistan', 'Asia', '5.087210178375244', '16061.429412691556'),
 'TUR': ('Turkey', 'Asia', '72.32691192626953', '17930.678781392427'),
 'TWN': ('Taiwan', 'Asia', '23.140947341918945', '37188.89452901009'),
 'UZB': ('Uzbekistan', 'Asia', '28.606294631958008', '6574.51672506692'),
 'VNM': ('Vietnam', 'Asia', '88.4725112915039', '4428.42140209062'),
 'YEM': ('Yemen', 'Asia', '23.606779098510742', '4553.557258443674'),
 'ALB': ('Albania', 'Europe', '2.9405250549316406', '9544.73991912729'),
 'AUT': ('Austria', 'Europe', '8.40994930267334', '40489.80635849457'),
 'BEL': ('Belgium', 'Europe', '10.938738822937012', '38177.94724875522'),
 'BGR': ('Bulgaria', 'Europe', '7.404590129852295', '14906.784964031736'),
 'BIH': ('Bosnia and Herzegovina',
  'Europe',
  '3.7220840454101562',
  '9049.364732987471'),
 'BLR': ('Belarus', 'Europe', '9.473071098327637', '16457.02937904922'),
 'CHE': ('Switzerland', 'Europe', '7.831971168518066', '55688.020214268225'),
 'CZE': ('Czech Republic',
  'Europe',
  '10.536286354064941',
  '26129.56935189834'),
 'DEU': ('Germany', 'Europe', '80.89478302001953', '40627.23054942445'),
 'DNK': ('Denmark', 'Europe', '5.554843902587891', '43416.22338615916'),
 'ESP': ('Spain', 'Europe', '46.788631439208984', '31610.980157042286'),
 'EST': ('Estonia', 'Europe', '1.3321019411087036', '20265.572596704937'),
 'FIN': ('Finland', 'Europe', '5.365781784057617', '38394.06158150017'),
 'FRA': ('France', 'Europe', '65.14578247070312', '35786.16161450548'),
 'GBR': ('United Kingdom', 'Europe', '63.30684280395508', '34810.28120173958'),
 'GRC': ('Greece', 'Europe', '11.446004867553711', '25815.80076797162'),
 'HRV': ('Croatia', 'Europe', '4.328153133392334', '19305.008088868373'),
 'HUN': ('Hungary', 'Europe', '9.927840232849121', '20477.79856764035'),
 'IRL': ('Ireland', 'Europe', '4.626927852630615', '47823.50815113634'),
 'ISL': ('Iceland', 'Europe', '0.3203279972076416', '37729.01607603124'),
 'ITA': ('Italy', 'Europe', '59.72980880737305', '34727.67268834705'),
 'LTU': ('Lithuania', 'Europe', '3.123802900314331', '18475.371145517096'),
 'LUX': ('Luxembourg', 'Europe', '0.5078889727592468', '57882.81020537233'),
 'LVA': ('Latvia', 'Europe', '2.1188480854034424', '16943.688322829523'),
 'MDA': ('Moldova', 'Europe', '4.0844807624816895', '4173.634753274572'),
 'MKD': ('Macedonia', 'Europe', '2.0707390308380127', '11265.643175619889'),
 'MLT': ('Malta', 'Europe', '0.41611000895500183', '22983.44992639125'),
 'MNE': ('Montenegro', 'Europe', '0.6242849826812744', '14960.94855221103'),
 'NLD': ('Netherlands', 'Europe', '16.68291664123535', '44004.146024770846'),
 'NOR': ('Norway', 'Europe', '4.885878086090088', '73262.68174170727'),
 'POL': ('Poland', 'Europe', '38.323402404785156', '21006.027309294514'),
 'PRT': ('Portugal', 'Europe', '10.652320861816406', '25788.2739652247'),
 'ROU': ('Romania', 'Europe', '20.44034767150879', '16775.609043478133'),
 'RUS': ('Russia', 'Europe', '143.15386962890625', '21754.067899615977'),
 'SRB': ('Serbia', 'Europe', '7.291436195373535', '12453.482551916944'),
 'SVK': ('Slovakia', 'Europe', '5.404294013977051', '23061.12099058148'),
 'SVN': ('Slovenia', 'Europe', '2.045167922973633', '25831.33128718208'),
 'SWE': ('Sweden', 'Europe', '9.390168190002441', '40421.906436576974'),
 'UKR': ('Ukraine', 'Europe', '45.79249954223633', '8713.259627419637'),
 'ABW': ('Aruba', 'North America', '0.10166899859905243', '37059.34143869512'),
 'AIA': ('Anguilla',
  'North America',
  '0.013768999837338924',
  '21098.057152999143'),
 'ATG': ('Antigua and Barbuda',
  'North America',
  '0.09466099739074707',
  '17162.16102781353'),
 'BHS': ('Bahamas',
  'North America',
  '0.36083200573921204',
  '29504.581346144332'),
 'BLZ': ('Belize', 'North America', '0.3216080069541931', '7145.952547655193'),
 'BMU': ('Bermuda',
  'North America',
  '0.06395599991083145',
  '51447.147481910884'),
 'BRB': ('Barbados',
  'North America',
  '0.27956900000572205',
  '13995.612896540895'),
 'CAN': ('Canada', 'North America', '34.16866683959961', '40269.03526728651'),
 'CRI': ('Costa Rica',
  'North America',
  '4.5452799797058105',
  '12106.99191566891'),
 'CUW': ('Curaçao',
  'North America',
  '0.14760799705982208',
  '24337.816177887613'),
 'CYM': ('Cayman Islands',
  'North America',
  '0.05550700053572655',
  '36670.678481483985'),
 'DMA': ('Dominica',
  'North America',
  '0.07143999636173248',
  '9101.993729581038'),
 'DOM': ('Dominican Republic',
  'North America',
  '9.897985458374023',
  '11500.13134401987'),
 'GRD': ('Grenada',
  'North America',
  '0.1046769991517067',
  '9791.291133092265'),
 'GTM': ('Guatemala',
  'North America',
  '14.630416870117188',
  '6359.262684615136'),
 'HND': ('Honduras',
  'North America',
  '8.194778442382812',
  '3789.9102248753848'),
 'HTI': ('Haiti', 'North America', '9.999616622924805', '1650.2697128700804'),
 'JAM': ('Jamaica',
  'North America',
  '2.8172099590301514',
  '6675.176956450172'),
 'KNA': ('Saint Kitts and Nevis',
  'North America',
  '0.05144499987363815',
  '17837.349758162076'),
 'LCA': ('Saint Lucia',
  'North America',
  '0.17258000373840332',
  '9178.266990673521'),
 'MEX': ('Mexico',
  'North America',
  '117.31893920898438',
  '14507.011923865603'),
 'MSR': ('Montserrat',
  'North America',
  '0.00494399992749095',
  '15385.002991478228'),
 'NIC': ('Nicaragua',
  'North America',
  '5.737722873687744',
  '3992.7619078961748'),
 'PAN': ('Panama',
  'North America',
  '3.6432220935821533',
  '15055.237591848218'),
 'SLV': ('El Salvador',
  'North America',
  '6.164626121520996',
  '6096.958357099287'),
 'TCA': ('Turks and Caicos Islands',
  'North America',
  '0.0309939999913334846',
  '7230.966074605446'),
 'TTO': ('Trinidad and Tobago',
  'North America',
  '1.3280999660491943',
  '27510.236190041924'),
 'USA': ('United States',
  'North America',
  '308.6413879394531',
  '49500.62628346238'),
 'VCT': ('Saint Vincent and the Grenadines',
  'North America',
  '0.10931500047445297',
  '8095.21923068351'),
 'VGB': ('British Virgin Islands',
  'North America',
  '0.027224000543355942',
  '21183.279165646647'),
 'AUS': ('Australia', 'Oceania', '22.12006378173828', '44854.90004866657'),
 'FJI': ('Fiji', 'Oceania', '0.8599500060081482', '6902.0687720944225'),
 'NZL': ('New Zealand', 'Oceania', '4.370061874389648', '30867.05437525178'),
 'ARG': ('Argentina',
  'South America',
  '41.2238883972168',
  '15841.658208158999'),
 'BOL': ('Bolivia', 'South America', '9.918242454528809', '4806.608983377068'),
 'BRA': ('Brazil', 'South America', '196.7962646484375', '13541.462561602328'),
 'CHL': ('Chile', 'South America', '16.99335479736328', '18092.94007371081')}
```

14: Using *cData*, identify what is the most common Continent among the nations in the list *countries*.

[28]:
```
# We're using constants here just to make the code more readable.
COUNTRY_ISO_CODE_INDEX = 0
C_DATA_CONTINENT_INDEX = 1
C_DATA_POPULATION_INDEX = 2
C_DATA_COUNTRY_NAME_INDEX = 0
C_DATA_GPD_INDEX = 3

continent_counts = {}

# Here we have two options, the first one is to create temporary variables to␣
 ↪store the most common continent and use it
# for each iteration. The second one is use the loop just to compute the dict␣
 ↪and then use the max function to get the
# continent with the highest count. We'll use the second approach here.
```

```
for country in countries:
    country_data = cData.get(country[COUNTRY_ISO_CODE_INDEX])
    if country_data:
        continent = country_data[C_DATA_CONTINENT_INDEX]
        if continent not in continent_counts:
            continent_counts[continent] = 0
        continent_counts[continent] += 1

max(continent_counts.items(), key=lambda x: x[1])[0]
```

[28]:
```
'Europe'
```

15: Using *cData*, identify what is the the most populated nation in the list *countries*.

[29]:
```
highest_population = 0
country_with_highest_population = None

for country in countries:
    country_data = cData.get(country[COUNTRY_ISO_CODE_INDEX])
    if country_data:
        population = float(country_data[C_DATA_POPULATION_INDEX])
        if population > highest_population:
            highest_population = population
            country_with_highest_population =␣
 ↪country_data[C_DATA_COUNTRY_NAME_INDEX]

print(f"The most populated nation is {country_with_highest_population}")
```

```
The most populated nation is Russia
```

16: Compare the average GDP per capita of the nations in the list *countries* with the average GDP of the countries in cdata file. What can you conclude?

[30]:
```
# Esse ta incompleto ainda... Nada faz sentido e é melhor esperar para␣
 ↪confirmar com o professor durante a aula
total_gpd_countries = 0
total_countries_found = 0

for country in countries:
    country_data = cData.get(country[COUNTRY_ISO_CODE_INDEX])
    if country_data:
        total_gpd_countries += float(country_data[C_DATA_GPD_INDEX])
        total_countries_found += 1

average_gpd_countries = total_gpd_countries / total_countries_found

total_gpd_cdata = 0
for row in cData.values():
    total_gpd_cdata += float(row[C_DATA_GPD_INDEX])

average_gpd_cdata = total_gpd_cdata / len(cData.values())
print(f"Average GPD in countries list: {average_gpd_countries}")
print(f"Average GPD in cData: {average_gpd_cdata}")
```

```
Average GPD in countries list: 38242.63072863724
Average GPD in cData: 18014.65454394603
```

We can conclude that the average GDP in cData is less than the average GDP in countries list.

If we check the two datasets, we'll see that, although the cData contains a lot of outliers pushing the average up, there are also a lot of countries that have a very poor GDP.

This the inverse of the countries list, which almost all countries have a higher GDP if comparing the the countries contained in the cData list. For example: in the countries data, the lowest GDP (Russia with 21754.068) have the GDP close to the 75% quantile of cData (21033,63).

4 Part 3 - Functions hurt nobody

4.1 Exercise I - I hate math

Consider the following equation:

$$y = 6x^2 + 3x + 2 \tag{1}$$

17: Write a function called *f* that takes one argument, x, and returns y according to the equation above.

[31]:
```
def f (x):
    y = (6 * x **2) + (3 * x) + 2
    return y
```

17.1: Call the function for x = 2 and print the answer.

[32]:
```
f(2)
```

[32]:
```
32
```

Consider the following sequence of numbers:

$$x_0 = 0; \qquad x_n = x_{n-1} - n; \quad if \;\; x_{n-1} - n > 0 \quad and\ is\ not\ already\ in \tag{2}$$

17.2: Write a function that returns the nth digit of the above defined sequence.

Note: the above sequence is also known as the Recamán's sequence, and it was invnted by Bernardo Recamán Santos (Bogotá, Colombia)

[33]:
```
def Recaman(n):
    sequence = [0] # The first element is zero by definition
    for i in range(1, n): # Iterate skipping the first element
        next_element = sequence[i-1] - i

        if next_element > 0 and next_element not in sequence:
            next_element = next_element # This is dummy, but we need it to make␣
↪the python interpreter happy
        else:
            next_element = sequence[-1] + i

        sequence.append(next_element)

    return sequence
```

[34]:
```
Recaman(10)
```

[34]:
```
[0, 1, 3, 6, 2, 7, 13, 20, 12, 21]
```

17.3 Write a function named isPrime that takes an integer as input and outputs True if the number is prime and False if the number if not prime.

[35]:
```
def isPrime(integer):
    #A number is prime when it is possible to divide it by 1 and itself.
    if integer <= 1:
        return False
    for count in range(2, integer): #(2, integer-1)
        if integer % count == 0: # count is divisible by integer, which would␣
↪indicate that num is not a prime number.
            return False
    return True # divisible by itself
```

17.4 Test your function against some examples to show that it works as expected.

[36]:
```
integer = 643
isPrime(integer)
```

[36]:
```
True
```

4.2 Exercise II - Monty Hall Problem

18: The Monty Hall Problem is a probability puzzle loosely based on the American television game show "Let's Make a Deal" and named after its original host, Monty Hall. The puzzle can be stated as follows

> Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice? +info: https://en.wikipedia.org/wiki/Monty_Hall_problem

While intuitively your guess might be that the chances of winning the car is 1/3 independently of the choice to switch doors or not. Howewver, the odds of winning if you switch doors are 2/3 and greater than if you decide not to switch. While there are theoretical solutions to this problem that allow us to estimate the correct odds of winning in each scenario – switching or not doors – an alternative way to proof such outcome is through computer simulations. That is the goal of this exercise, to write the necessary components to simulate the Monty Hall problem and validate the theoretical results through simulations.

In this exercise, start by implementing a function that simulates one instance of the Monty Hall problem. In that sense, write a function called MontyHall that accepts **one argument**: - A boolean (True/False, or 1/0) that specifies if the player switches doors or not (after the host has opened his door, which contains a goat). The function should return True if the player wins the car or False if not.

Naturally, for the function to reproduce the contest it needs to consider and perform some actions, such as: - Set up the game, that is, create the necessary variables to store the information about three doors and what they have behind, as well as an indicator to track the choice of the player. - Given the initial choice of the player, simulate the opening of one of the two remaining doors that contains a goat. - Given the last two doors left, simulate if the player wants to switch or not his/her choice given their strategy (defined by the input boolean argument of the function). - Output the result, if the contestant correctly guesses which door hides the car or if not.

The function MontyHall should simulate one instance of the contest given the choice of the player (to switch or not).

The goal is to understand, statistically, which action leads to the highest probability of winning. *Note: You should use the library random to solve this exercise.*

[37]:
```
import random

def MontyHall (change_door): # true if changes door or false if not
    doors = [False, False, False]
    price_position = random.randint(0, 2) # index of the price (car)
    doors[price_position] = True

    door_selected = random.randint(0, 2) # door chosen by the player
    if not change_door:
        return doors[door_selected]

    # Here we're getting the possible goat positions that the host can open, it␣
 ↪cannot be the same as the price position or the
    # already chosen door position
    goat_positions = [idx for idx, value in enumerate(doors) if idx not␣
 ↪in[price_position, door_selected]]
    door_opened_by_host = random.choice(goat_positions)

    # We're selection the door that is different from the one that was opened␣
 ↪by the host and also different from the selected before
```

```
    door_selected = [idx for idx, value in enumerate(doors) if idx not in␣
↪[door_opened_by_host, door_selected]]

    return door_selected[0] == price_position
```

[ ]:

```
```

18.1 Now that you have a function that simulates an instance of the Monty Hall problem, implement an experiment where you repeat many times (thousands of times) for each of the two possible scenarios: player switching the door, and player not switcing the door. Keep track of the results and estimate the frequencies of wins for each scenario and discuss if the results are inline with the theoretical odds.

[38]:

```
results_function_false = []
results_function_true = []

TEST_TIMES = 10000
for result in range(TEST_TIMES): #test 100 times
    result = MontyHall(False)
    results_function_false.append(result)
filtered_wins_false = list(filter(lambda x : x == True,␣
↪results_function_false)) # filters the wins when the player doen't change␣
↪door
percentage_of_wins = len(filtered_wins_false) / TEST_TIMES * 100
print("The frequencies of wins when player doesn't change door is",␣
↪len(filtered_wins_false), "(", percentage_of_wins, "%)")

for result in range(TEST_TIMES):
    result = MontyHall(True)
    results_function_true.append(result)
filtered_wins_true = list(filter(lambda x : x == True, results_function_true))␣
↪# filters the wins when the player changes door
percentage_of_wins = len(filtered_wins_true) / TEST_TIMES * 100
print("The frequencies of wins when player changes door is",␣
↪len(filtered_wins_true), "(", percentage_of_wins ,"%)")
```

```
The frequencies of wins when player doesn't change door is 3398 ( 33.98 %)
The frequencies of wins when player changes door is 6671 ( 66.71000000000001 %)
```

4.3 Exercise III

19: Consider the list A, declared below. Use the function map() to change the values of A by adding 1 to each value if it is even and subtracting 1 to it otherwise.

[39]:

```
A =␣
↪[460,3347,3044,490,699,1258,1804,973,2223,3416,2879,1058,2915,2422,351,1543,1020,208,643,79
```

[40]:
```python
def f_add(integer):
    if integer % 2 == 0: # function that verify if an integer is prime or not.
        return integer + 1
    else:
        return integer - 1

add_function = list(map(f_add, A)) # interate with each value of the list A and␣
 ↪verify the conditions in f_add
print(add_function)
```

```
[461, 3346, 3045, 491, 698, 1259, 1805, 972, 2222, 3417, 2878, 1059, 2914, 2423,
350, 1542, 1021, 209, 642, 794, 3336, 2584, 470, 2622, 1076]
```

20: Create a list B with the same size of A, where each element is True if the associated value in list A is greater than 700, else is False.

[41]:
```python
B = []

for value in A: # for each value in A verify the following conditions
    if value > 700:
        B.append(True)
    else:
        B.append(False)
print(B)
```

```
[False, True, True, False, False, True, True, True, True, True, True, True,
True, True, False, True, True, False, False, True, True, True, False, True,
True]
```

21: Create a list L that contains the Logarithm of base 10 of each value in A.

Note: You should use the module math to solve this exercise.

[42]:
```python
L = []

import math # library math functions

for value in A:
    log_A = math.log10(value) # log10 of each value in A
    L.append(log_A) #add to list L
print(L)
```

```
[2.662757831681574, 3.5246557123577773, 3.4834446480985353, 2.690196080028514,
2.8444771757456815, 3.09968064110925, 3.256236533205923, 2.988112840268352,
3.3469394626989906, 3.5335178620169674, 3.459241664878082, 3.024485667699167,
3.464638559095033, 3.3841741388070337, 2.545307116465824, 3.188365926063148,
3.0086001717619175, 2.3180633349627615, 2.808210972924222, 2.9003671286564705,
3.5233562066547925, 3.412460547429961, 2.673020907128896, 3.4187982905903533,
3.0322157032979815]
```

22: How many numbers in A are greater than 1000?

Note: You should use the function filter().

```
[43]: new_A = list(filter(lambda x: x > 1000, A)) # only the values >1000 from A
      print("There are", len(new_A), "numbers in A are greater than 1000.")
```

```
There are 16 numbers in A are greater than 1000.
```

23 Use the function isPrime, that you declared in 17.3, to identify which numbers from A are prime. Note that depending on your implementation of isPrime this task can take more or less time. Use filter to achieve this task.

```
[44]: # isPrime returns true or false if a number is prime or not
      prime_A = list(filter(isPrime, A))
      print(prime_A)
```

```
[3347, 2879, 1543, 643]
```

4.3.1 Congratulations, it is done!