Programming for Data Science 2023

Homework Assigment One

Homework activities aim not only at testing your ability to put into practice the concepts you have learned during the Lectures and Labs, but also your ability to explore the Python documentation as a resource.

Above all, it is an opportunity for you to challenge yourself and practice. If you are having difficulties with the assignment reach out for support.

The Homework Assigment One is divided into **three parts**:

1. Explore the core building blocks of programming, focusing in variables, data structures, and their manipulation;
2. Explore data loading and analysis using the core elements of Python, without fancy third-party libraries;
3. Explore functional programming by defining different functions and operating with the map() and filter().

Your submission will be graded according to the following guidelines:

1. **Execution** (does your program do what is asked from the exercise?);
2. **Objectivity** (are you using the adequate libraries?);
3. **Readibility** of your code (that includes comments, naming of variables, supporting text, etc ...).

This assignment is to be done in groups of two, groups that are caught cheating will obtain a score of 0 points. \

Oral Examination will be conducted to a subset of groups picked randomly and or that have been flagged during the grading

The Homeworking Assignment One is worth **20%** of your final grade.

The submission package should correspond to a .zip archive (**.rar files are not accepted**) with the following files:

1. Jupyter Notebook with the output of all the cells;
2. PDF/HTML of your Jupyter Notebook (on Jupyter go to File -> Download As -> PDF);
3. All text or .csv files exported as part of the exercises.

Important Notes:

- **You are ONLY allowed to use any library from the Python STL (see here https://docs.python.org/3/library/).**
- **Comment your code properly, which includes naming your variables in a meaningful manner. Badly documented code will be penalized.**</b>

- **Submissions in non .zip format will be penalized with 2 points.**

Submission is done through the respective Moodle activity.\ Deadline is **20223-09-28**, a Thursday, at 23:59.\ A penality of 1 point per day late will be applied to late deliveries.

Start Here

[Please Complete the following form with your details]

Student Name - Pedro Cerejeira
Student id - 20230442
Contact e-mail - 20230442@novaims.unl.pt

Student Name - Maria Almeida
Student id - 20230489
Contact e-mail - 20230489@novaims.unl.pt

Part 1 - Variable Declaration and Manipulation

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.

In [1]:

```python
import random
X = []

for r in range(100): # range 100 to generate 100 entries for the list
    i = random.randint(-100, 100) # random number between -100 and 100
    X.append(i)

print('X:',X)
```

```
X: [-45, 79, -16, -40, -8, -64, -89, -19, 8, -38, 16, -48, 79, -86, -52, 63, -89,
26, 93, 9, -63, 62, 89, 26, -64, 76, 95, 56, 19, -70, -47, 95, -94, 81, -36, -17,
15, -27, 99, -20, 30, 38, -32, 60, -80, 97, -90, 30, 98, -89, -64, 74, -30, -9, 2
3, 3, 41, 90, 65, -33, 98, -51, -83, 25, -32, -14, 31, -71, -97, 71, 62, -48, 64,
60, -54, -29, 17, 39, 20, -17, 48, 52, -62, 29, -59, 66, -18, -21, 82, 97, -100, 3
5, 88, -17, 74, -66, -64, -22, 37, 27]
```

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

In [2]:

```python
odd = 0
for i in X: # iterating through X and counting the odd numbers
    if i%2 != 0:
        odd += 1

print("There are", odd, "odd numbers.")
```

```
There are 47 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.

In [3]:

```
even = 0

for i in X: # iterating through X and counting the even numbers
    if i%2 == 0:
        even +=1

if (even != odd): # comparing number of even with odd numbers
    X = []
    print("The number of even numbers is different from the odd numbers, please ge

else:
    print("The number of even and odd numbers in the list is the same!")
```

```
The number of even numbers is different from the odd numbers, please generate a ne
w list
```

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.

In [5]:

```
import random

# simply defining functions for everything mentioned above, so the tasks can be per

def generateRandomList():
    list1 = []

    for r in range(100):
        i = random.randint(-100, 100)
        list1.append(i)

    return list1

def checkOdd(list1):
    odd = 0
    for i in list1:
        if i%2 != 0:
            odd += 1
    return odd

def compareEvenOdd(list1, odd):
    even = 0

    for i in list1:
        if i%2 == 0:
            even +=1

    if (even != odd):
        list1 = []
        print("The number of even numbers is different from the odd numbers, please
    else:
        print("The number of even and odd numbers in the list is the same!")

list1 = generateRandomList()
compareEvenOdd(list1, checkOdd(list1))
```

```
The number of even numbers is different from the odd numbers, please generate a ne
w list
```

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.

In [6]:

```
list1 = generateRandomList()

digits_5th = len(str(list1[4])) # index 4 will be the 5th element, since the index
digits_100th = len(str(list1[99])) # index 99 will be the 100th element, since the

if str(list1[4])[0] == "-": # converting to string and, since '-' is also a charact
                            # if it exists
    digits_5th -= 1

if str(list1[99])[0] == "-":
    digits_100th -= 1

print("The 5th element has", digits_5th, "digits.\nThe 100th element has", digits_1
```

```
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?

In [8]:

```
def sumList(list1):
    total = sum(list1)

    if total % 2 == 0:
        print("The sum of all the list's elements is even!")
    else:
        print("The sum of all the list's elements is odd!")
    return total

list1 = generateRandomList()
sumList(list1)
```

```
The sum of all the list's elements is odd!
```

Out[8]:

```
79
```

5: What is the average of the list X?

In [9]:

```
def averageList(list1):
    total = sum(list1)

    average = total / len(list1)

    print("The average of the List X is:", average)
    return average

X = generateRandomList()
averageList(X)
```

```
The average of the List X is: -3.3
```

Out[9]:

```
-3.3
```

5.1. What is the population standard deviation?

In [11]:

```
from math import sqrt

def variance(list1): # calculating the variance
    aux = 0
    size = len(list1)
    mean = averageList(list1)

    for x in list1:
        aux += (x - mean) ** 2

    return aux / size

def standardDeviation(variance): # calculating the standard deviation
    stdDev = sqrt(variance) # this would be the same as variance**(1/2)
    print("The standard deviation of the list is:", stdDev)
    return stdDev

list1 = generateRandomList()
variance = variance(list1)
standardDeviation(variance)
```

```
The average of the List X is: -5.65
The standard deviation of the list is: 56.76642933988361
```

Out[11]:

```
56.76642933988361
```

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

In [20]:

```
X = generateRandomList()
def sortReverse(list1):
    X.sort(reverse=True) # reverse since we want descending order
    Xsort = X

    print("The result of XSort is:", Xsort, "\n")
    return Xsort
Xsort = sortReverse(X)
```

```
The result of XSort is: [99, 96, 95, 92, 90, 89, 89, 88, 85, 84, 65, 62, 61, 61, 5
5, 54, 53, 51, 49, 48, 44, 44, 41, 40, 38, 36, 35, 32, 27, 24, 23, 23, 22, 22, 22,
17, 16, 15, 14, 13, 12, 11, 10, 6, 4, 3, 1, -1, -2, -8, -9, -10, -10, -11, -11, -1
2, -13, -14, -16, -17, -22, -24, -25, -28, -29, -30, -31, -35, -36, -37, -37, -39,
-42, -44, -48, -49, -49, -50, -53, -57, -59, -60, -61, -62, -68, -70, -70, -71, -7
1, -72, -74, -78, -79, -85, -85, -86, -89, -92, -96, -98]
```

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

Note: Per definition consider that Xsort[-1] = 0.

In [21]:

```
for i in range (len(X)):
    if (i-1 < 0):
        Xsort[i] = 0 + Xsort[i]
    else:
        Xsort[i] = Xsort[i-1] + Xsort[i] # it is cumulative because Professor Nuno

print ("The new Xsort is:", Xsort)
```

```
The new Xsort is: [99, 195, 290, 382, 472, 561, 650, 738, 823, 907, 972, 1034, 109
5, 1156, 1211, 1265, 1318, 1369, 1418, 1466, 1510, 1554, 1595, 1635, 1673, 1709, 1
744, 1776, 1803, 1827, 1850, 1873, 1895, 1917, 1939, 1956, 1972, 1987, 2001, 2014,
2026, 2037, 2047, 2053, 2057, 2060, 2061, 2060, 2058, 2050, 2041, 2031, 2021, 201
0, 1999, 1987, 1974, 1960, 1944, 1927, 1905, 1881, 1856, 1828, 1799, 1769, 1738, 1
703, 1667, 1630, 1593, 1554, 1512, 1468, 1420, 1371, 1322, 1272, 1219, 1162, 1103,
1043, 982, 920, 852, 782, 712, 641, 570, 498, 424, 346, 267, 182, 97, 11, -78, -17
0, -266, -364]
```

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.

In [22]:

```
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"

}
```

In [23]:

```
clients = {

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

}
```

In [24]:

```
listAux = []
multipleMovieClient = []

for key, value in purchases.items():
    values = value.split(',') # splitting the movies by each ',' so we can figure o
    if (len(values) > 1):
        multipleMovieClient.append(clients[key])

print("The clients who bought more than 1 videogame are:", multipleMovieClient)
```

```
The clients who bought more than 1 videogame are: ['Morty Smith', 'Mildred Ratche
d', '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

In [25]:

```
finalList = []
multipleMovieClient = {}

for key, value in purchases.items():
    values = value.split(',')
    multipleMovieClient[clients[key]] = len(values) # appending the number of movie

maxGames = max(multipleMovieClient.values()) # figuring out the maximum number of m

for key, value in multipleMovieClient.items(): # we are also using a Loop here, so
                                                # if both bought the max number of m
    if (value == maxGames):
        finalList.append(key)
```

```
if (len(finalList) > 1):
    print("There was more than 1 client buying the maximum number of games, they ar
else:
    print("The client buying more games was:", finalList)
```

```
There was more than 1 client buying the maximum number of games, they are: ['Morty
Smith', 'Sheldon Cooper']
```

7.2 What is the name of the most popular videogame?

In [26]:

```
games = {}

for value in purchases.values():
    values = value.split(',')
    for i in range(len(values)):
        values[i] = values[i].strip().lower() # picking each videogame, stripping
                                              # (because of str.split()) and placi

    for game in values:
        if (game in games.keys()): # if game is already in the dict, add one more
                                   # add the game to the dictionary with 1 sold
            games[game] += 1
        else:
            games[game] = 1

topGame = max(games, key=games.get) # we could also solve this with a loop, but th
print('The most popular videogame is:',topGame)
```

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

Part 2 - Data loading and analysis

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.

In [28]:

```
import os

filePath = "alice.txt" # defining the file

# Open the file for reading
with open(filePath, 'r') as file: # Read the contents of the file 'r'
    Alice = file.read()

    print(Alice) # Print or process the file contents as needed
```

```
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 re
ading, but it had no pictures or conversations in it, <<and what is the use of a b
ook>>. 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

In [29]:

```
# For this exercise, we assumed we want to keep repeated words, otherwise we would

file_path = "alice.txt"

wAlice = []

with open(file_path, 'r') as file: # Open the file for reading
    Alice = file.read()

    # Split the text into words using whitespace as the delimiter
    words = Alice.split()

    for word in words: # placing every word in lowercase
        cleaned_word = ''.join(char.lower() for char in word if char.isalpha()) # c
                                                                                # c
        if cleaned_word: # if there are no alphanumeric characters, don't append
            wAlice.append(cleaned_word)

print('The words from the file Alice are:', wAlice)
```

```
The words from the file Alice are: ['alice', 'was', 'beginning', 'to', 'get', 'ver
y', 'tired', 'of', 'sitting', 'by', 'her', 'sister', 'on', 'the', 'bank', 'and',
'of', 'having', 'nothing', 'to', 'do', 'once', 'or', 'twice', 'she', 'had', 'peepe
d', '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', 'conver
sation']
```

Using the list *wAlice* answer the following questions:

10: How many words are in the file Alice.txt?

In [30]:

```
words = len(wAlice)
print(words)
```

```
57
```

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.

In [31]:

```
minLetters = 9**99 # initializing with a big number
minWords = {}
maxLetters = 0 # initializing with a small number
maxWords = {}
for word in wAlice:
    if (len(word) < minLetters): # handling smallest word
        minLetters = len(word)
        minWords = {word}
    elif (len(word) == minLetters):
        minWords.add(word)
    if (len(word) > maxLetters): # handling, in parallel the biggest word
        maxLetters = len(word)
        maxWords = {word}
    elif (len(word) == maxLetters):
        maxWords.add(word)

print(maxWords)
print(minWords)
```

```
{'conversations'}
{'a'}
```

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

In [32]:

```
wAliceNoDups = set(wAlice) # converting to set, so we eliminate repeated words

print(wAliceNoDups)
```

```
{'what', 'bank', 'very', 'once', 'reading', 'pictures', 'she', 'is', 'get', 'havin
g', 'and', 'had', 'conversations', 'the', 'was', 'it', 'book', 'sister', 'but', 'i
n', 'nothing', 'use', 'of', 'a', 'into', 'or', 'by', 'twice', 'on', 'alice', 'peep
ed', 'conversation', 'to', 'do', 'no', 'without', 'thought', 'beginning', 'tired',
'sitting', 'her'}
```

12.1:How many different words does the text contain?

In [33]:

```
words = len(wAliceNoDups)
print(words)
```

```
41
```

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.

In [34]:

```
countries = [

('PRT','Portugal','Europe'),
('ISL','Israel','Asia'),
('COL','Colombia','South America'),
('NEV','Nevada','North America'),
('JPN','Japan','Asia'),
('RUS','Russia','well... which part?'),
('DEN','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*.

In [38]:
```
import csv

raw = []

with open('cdata.csv', 'r') as file: # open the file in read mode
    rawFile = csv.reader(file)

    for row in rawFile:
        raw.append(row) # append each row of the file

print(raw)
```

```
[['iso', 'countryname', 'continent', 'population(mil)', 'gdp'], ['ago', 'Angola',
'Africa', '23.369131088256836', '5988.534997149481'], ['bdi', 'Burundi', 'Africa',
'8.766929626464844', '731.4232803265862'], ['ben', 'Benin', 'Africa', '9.199258804
321289', '1919.9969479963038'], ['bfa', 'Burkina Faso', 'Africa', '15.605216979980
469', '1327.2165314775982'], ['bwa', 'Botswana', 'Africa', '2.0148661136627197',
'12256.14159052444'], ['caf', 'Central African Republic', 'Africa', '4.44852495193
48145', '865.4421867263901'], ['cmr', 'Cameroon', 'Africa', '19.970495223999023',
'2684.972890247272'], ['cod', 'Democratic Republic of the Congo', 'Africa', '64.52
326202392578', '634.9713963013496'], ['cog', 'Republic of the Congo', 'Africa',
'4.386693000793457', '4979.990496069269'], ['com', 'Comoros', 'Africa', '0.6896920
204162598', '2292.997969425929'], ['cpv', 'Cape Verde', 'Africa', '0.5023840069770
813', '5644.821253339075'], ['dji', 'Djibouti', 'Africa', '0.8511459827423096', '2
124.4420625972525'], ['dza', 'Algeria', 'Africa', '36.117637634277344', '12590.224
223813591'], ['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', '1.692149
043083191', '2681.858616850214'], ['gnb', 'Guinea-Bissau', 'Africa', '1.555879950
5233765', '1387.2497310879905'], ['gnq', 'Equatorial Guinea', 'Africa', '0.9511039
853096008', '31416.887137501908'], ['ken', 'Kenya', 'Africa', '41.35015106201172',
'2484.0349094725366'], ['lbr', 'Liberia', 'Africa', '3.948124885559082', '786.6702
306741524'], ['lso', 'Lesotho', 'Africa', '2.040550947189331', '2432.797654688082
3'], ['mar', 'Morocco', 'Africa', '32.409637451171875', '6421.937677907418'], ['md
g', 'Madagascar', 'Africa', '21.151639938354492', '1459.9156124535612'], ['mli',
'Mali', 'Africa', '15.075084686279297', '1873.2809618195865'], ['moz', 'Mozambiqu
e', 'Africa', '24.221405029296875', '969.2307622526259'], ['mrt', 'Mauritania', 'A
frica', '3.6095430850982666', '3082.647028528842'], ['mus', 'Mauritius', 'Africa',
'1.2479549646377563', '15178.325920597827'], ['mwi', 'Malawi', 'Africa', '15.16709
5184326172', '972.0456815116237'], ['nam', 'Namibia', 'Africa', '2.17317008972167
97', '7689.171239820006'], ['ner', 'Niger', 'Africa', '16.42557716369629', '845.869
3682212395'], ['nga', 'Nigeria', 'Africa', '158.5782623291015 6', '5186.3043075423
48'], ['rwa', 'Rwanda', 'Africa', '10.246842384338379', '1379.699325232700 9'], ['sd
n', 'Sudan', 'Africa', '34.385963439941406', '3608.7856176907358'], ['sen', 'Seneg
al', 'Africa', '12.916229248046875', '2741.121456198507'], ['sle', 'Sierra Leone',
'Africa', '6.4587202072143555', '1161.8796942144647'], ['swz', 'Swaziland', 'Afric
a', '1.2028429508209229', '7042.524359035093'], ['syc', 'Seychelles', 'Africa',
'0.0914049968123436', '17960.353075613308'], ['tcd', 'Chad', 'Africa', '11.8872022
62878418', '1880.374876279971 4'], ['tgo', 'Togo', 'Africa', '6.5029520988464355',
'1222.8150529133295'], ['tun', 'Tunisia', 'Africa', '10.639930725097656', '10647.8
34980284626'], ['tza', 'Tanzania', 'Africa', '44.82893753051758', '1979.1718644815
185'], ['uga', 'Uganda', 'Africa', '33.91513442993164', '1723.5943355766267'], ['z
af', 'South Africa', 'Africa', '51.58466339111328', '11388.640515995063'], ['zmb',
'Zambia', 'Africa', '13.850032806396484', '2870.8872656342314'], ['zwe', 'Zimbabw
e', 'Africa', '14.08631706237793', '1479.0305839163732'], ['are', 'United Arab Emi
rates', 'Asia', '8.270684242248535', '59707.412565389604'], ['arm', 'Armenia', 'As
ia', '2.8773109912872314', '8222.945234202914'], ['aze', 'Azerbaijan', 'Asia', '9.
03245735168457', '12947.102198682365'], ['bgd', 'Bangladesh', 'Asia', '152.1491088
8671875', '2411.1020691099325'], ['bhr', 'Bahrain', 'Asia', '1.2408620119094849',
'37045.81310919623'], ['brn', 'Brunei', 'Asia', '0.3886620104312897', '67320.89957
245934'], ['btn', 'Bhutan', 'Asia', '0.7276409864425659', '7235.18974818538'], ['c
hn', 'China', 'Asia', '1359.755126953125', '9337.29077267725 4'], ['cyp', 'Cyprus',
'Asia', '0.829446017742157', '28046.373877741087'], ['geo', 'Georgia', 'Asia', '4.
231660842895508', '7966.602756952667'], ['hkg', 'Hong Kong', 'Asia', '7.0252208709
7168', '41687.946418899235'], ['ind', 'India', 'Asia', '1230.980712890625', '4357.
0597360582315'], ['irn', 'Iran', 'Asia', '74.56751251220703', '17328.45788289465
8'], ['irq', 'Iraq', 'Asia', '30.7627010345459', '9344.543768350652'], ['isr', 'Is
rael', 'Asia', '7.42595911026001', '28638.9037385305'], ['jor', 'Jordan', 'Asia',
'7.182390213012695', '9351.387283068141'], ['jpn', 'Japan', 'Asia', '128.551879882
8125', '36595.63364058582'], ['kaz', 'Kazakhstan', 'Asia', '16.398975372314453',
'17908.83993251286'], ['kgz', 'Kyrgyzstan', 'Asia', '5.422337055206299', '3382.463
9059259293'], ['khm', 'Cambodia', 'Asia', '14.30873966217041', '2330.12720763225
3'], ['kor', 'South Korea', 'Asia', '49.5528564453125', '31589.70516114569 5'], ['k
wt', 'Kuwait', 'Asia', '2.9980831146240234', '67029.5168001710 1'], ['lao', 'Laos',
'Asia', '6.246273994445801', '4316.969858587265'], ['lbn', 'Lebanon', 'Asia', '4.3
37141036987305', '18025.24893894264'], ['lka', 'Sri Lanka', 'Asia', '20.1983528137
20703', '8390.429701766443'], ['mac', 'Macau', 'Asia', '0.536969006061554', '9198
2.39517540816'], ['mdv', 'Maldives', 'Asia', '0.36451101303100586', '12140.8033907
39566'], ['mmr', 'Burma', 'Asia', '50.1558952331543', '3422.241340266976'], ['mn
g', 'Mongolia', 'Asia', '2.7126500606536865', '7670.714034382584'], ['mys', 'Malay
sia', 'Asia', '28.112289428710938', '17913.164410782432'], ['npl', 'Nepal', 'Asi
a', '27.023136138916016', '1996.1961554516242'], ['omn', 'Oman', 'Asia', '3.041460
0372314453', '40472.674942509126'], ['pak', 'Pakistan', 'Asia', '170.560180664062
5', '4171.416562353057'], ['phl', 'Philippines', 'Asia', '93.72662353515625', '539
1.233965240031'], ['qat', 'Qatar', 'Asia', '1.7796759605407715', '123128.401522834
33'], ['sau', 'Saudi Arabia', 'Asia', '27.425676345825195', '42331.65885721999'],
['sgp', 'Singapore', 'Asia', '5.074252128601074', '58618.43503468222'], ['syr', 'S
yria', 'Asia', '21.01883316040039', '5700.329438992399'], ['tha', 'Thailand', 'Asi
a', '67.20880889892578', '12496.244186726299'], ['tjk', 'Tajikistan', 'Asia', '7.6
41630172729492', '2784.5145587763627'], ['tkm', 'Turkmenistan', 'Asia', '5.0872101
78375244', '16061.429412691556'], ['tur', 'Turkey', 'Asia', '72.32691192626953',
'17930.678781392427'], ['twn', 'Taiwan', 'Asia', '23.140947341918945', '37188.8945
2901009'], ['uzb', 'Uzbekistan', 'Asia', '28.606294631958008', '6574.5167250669
2'], ['vnm', 'Vietnam', 'Asia', '88.4725112915039', '4428.42140209062'], ['yem',
'Yemen', 'Asia', '23.606779098510742', '4553.557258443674'], ['alb', 'Albania', 'E
urope', '2.9405250549316406', '9544.73991912729'], ['aut', 'Austria', 'Europe',
'8.40994930267334', '40489.80635849457'], ['bel', 'Belgium', 'Europe', '10.9387388
22937012', '38177.94724875522'], ['bgr', 'Bulgaria', 'Europe', '7.40459012985229
5', '14906.784964031736'], ['bih', 'Bosnia and Herzegovina', 'Europe', '3.72208404
54101562', '9049.364732987471'], ['blr', 'Belarus', 'Europe', '9.473071098327637',
'16457.02937904922'], ['che', 'Switzerland', 'Europe', '7.831971168518066', '5568
8.020214268225'], ['cze', 'Czech Republic', 'Europe', '10.536286354064941', '2612
9.56935189834'], ['deu', 'Germany', 'Europe', '80.89478302001953', '40627.23054942
445'], ['dnk', 'Denmark', 'Europe', '5.554843902587891', '43416.22338615916'], ['e
sp', 'Spain', 'Europe', '46.788631439208984', '31610.980157042286'], ['est', 'Esto
nia', 'Europe', '1.3321019411087036', '20265.572596704937'], ['fin', 'Finland', 'E
urope', '5.365781784057617', '38394.06158150017'], ['fra', 'France', 'Europe', '6
5.14578247070312', '35786.16161450548'], ['gbr', 'United Kingdom', 'Europe', '63.3
0684280395508', '34810.28120173958'], ['grc', 'Greece', 'Europe', '11.446004867553
711', '25815.80076797162'], ['hrv', 'Croatia', 'Europe', '4.328153133392334', '193
05.008088868373'], ['hun', 'Hungary', 'Europe', '9.927840232849121', '20477.798567
64035'], ['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', 'Europ
e', '0.5078889727592468', '57882.81020537233'], ['lva', 'Latvia', 'Europe', '2.118
8480854034424', '16943.688322829523'], ['mda', 'Moldova', 'Europe', '4.08448076248
16895', '4173.634753274572'], ['mkd', 'Macedonia', 'Europe', '2.0707390308380127',
'11265.643175619889'], ['mlt', 'Malta', 'Europe', '0.41611000895500183', '22983.44
992639125'], ['mne', 'Montenegro', 'Europe', '0.6242849826812744', '14960.94855221
103'], ['nld', 'Netherlands', 'Europe', '16.68291664123535', '44004.14602477084
6'], ['nor', 'Norway', 'Europe', '4.885878086090088', '73262.68174170727'], ['po
l', 'Poland', 'Europe', '38.323402404785156', '21006.027309294514'], ['prt', 'Port
ugal', 'Europe', '10.652320861816406', '25788.2739652247'], ['rou', 'Romania', 'Eu
rope', '20.44034767150879', '16775.609043478133'], ['rus', 'Russia', 'Asia', '143.
15386962890625', '21754.067899615977'], ['srb', 'Serbia', 'Europe', '7.29143619537
3535', '12453.482551916944'], ['svk', 'Slovakia', 'Europe', '5.404294013977051',
'23061.12099058148'], ['svn', 'Slovenia', 'Europe', '2.045167922973633', '25831.33
128718208'], ['swe', 'Sweden', 'Europe', '9.390168190002441', '40421.90643657697
4'], ['ukr', 'Ukraine', 'Europe', '45.79249954223633', '8713.259627419637'], ['ab
w', 'Aruba', 'NorthAmerica', '0.10166899859905243', '37059.34143869512'], ['aia',
'Anguilla', 'NorthAmerica', '0.013768999837338924', '21098.057152999143'], ['atg',
'Antigua and Barbuda', 'NorthAmerica', '0.09466099739074707', '17162.1610278135
3'], ['bhs', 'Bahamas', 'NorthAmerica', '0.36083200573921204', '29504.58134614433
2'], ['blz', 'Belize', 'NorthAmerica', '0.3216080069541931', '7145.952547655193'],
['bmu', 'Bermuda', 'NorthAmerica', '0.06395599991083145', '51447.147481910884'],
['brb', 'Barbados', 'NorthAmerica', '0.27956900000572205', '13995.612896540895'],
['can', 'Canada', 'NorthAmerica', '34.16866683959961', '40269.03526728651'], ['cr
```

```
i', 'Costa Rica', 'NorthAmerica', '4.5452799797058105', '12106.99191566891'], ['cu
w', 'CuraÃ§ao', 'NorthAmerica', '0.14760799705982208', '24337.816177887613'], ['cy
m', 'Cayman Islands', 'NorthAmerica', '0.05550700053572655', '36670.67848148398
5'], ['dma', 'Dominica', 'NorthAmerica', '0.07143999636173248', '9101.99372958103
8'], ['dom', 'Dominican Republic', 'NorthAmerica', '9.897985458374023', '11500.131
34401987'], ['grd', 'Grenada', 'NorthAmerica', '0.1046769991517067', '9791.2911330
92265'], ['gtm', 'Guatemala', 'NorthAmerica', '14.630416870117188', '6359.26268461
5136'], ['hnd', 'Honduras', 'NorthAmerica', '8.194778442382812', '3789.91022487538
48'], ['hti', 'Haiti', 'NorthAmerica', '9.999616622924805', '1650.2697128700804'],
['jam', 'Jamaica', 'NorthAmerica', '2.8172099590301514', '6675.176956450172'], ['k
na', 'Saint Kitts and Nevis', 'NorthAmerica', '0.05144499987363815', '17837.349758
162076'], ['lca', 'Saint Lucia', 'NorthAmerica', '0.17258000373840332', '9178.2669
90673521'], ['mex', 'Mexico', 'NorthAmerica', '117.31893920898438', '14507.0119238
65603'], ['msr', 'Montserrat', 'NorthAmerica', '0.004943999927490950', '15385.00299
1478228'], ['nic', 'Nicaragua', 'NorthAmerica', '5.737722873687744', '3992.7619078
961748'], ['pan', 'Panama', 'NorthAmerica', '3.6432220935821533', '15055.237591848
218'], ['slv', 'El Salvador', 'NorthAmerica', '6.164626121520996', '6096.958357099
287'], ['tca', 'Turks and Caicos Islands', 'NorthAmerica', '0.030993999913334846',
'7230.966074605446'], ['tto', 'Trinidad and Tobago', 'NorthAmerica', '1.3280999660
491943', '27510.236190041924'], ['usa', 'United States', 'NorthAmerica', '308.6413
879394531', '49500.62628346238'], ['vct', 'Saint Vincent and the Grenadines', 'Nor
thAmerica', '0.10931500047445297', '8095.21923068351'], ['vgb', 'British Virgin Is
lands', 'NorthAmerica', '0.027224000543355942', '21183.279165646647'], ['aus', 'Au
stralia', 'Oceania', '22.12006378173828', '44854.90004866657'], ['fji', 'Fiji', 'O
ceania', '0.8599500060081482', '6902.0687720944225'], ['nzl', 'New Zealand', 'Ocea
nia', '4.370061874389648', '30867.05437525178'], ['arg', 'Argentina', 'SouthAmeric
a', '41.2238883972168', '15841.658208158999'], ['bol', 'Bolivia', 'SouthAmerica',
'9.918242454528809', '4806.608983377068'], ['bra', 'Brazil', 'SouthAmerica', '196.
7962646484375', '13541.462561602328'], ['chl', 'Chile', 'SouthAmerica', '16.993354
79736328', '18092.94007371081']]
```

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.

In [37]:

```
cData = {}
aux = 0
for element in raw:
    if aux == 0:
        aux = 1
        continue
    cData[element[0]] = (element[1], element[2], element[3], element[4])
print(cData)
```

```
{'ago': ('Angola', 'Africa', '23.369131088256836', '5988.534997149481'), 'bdi':
('Burundi', 'Africa', '8.766929626464844', '731.4232803265862'), 'ben': ('Benin',
'Africa', '9.199258804321289', '1919.9969479963038'), 'bfa': ('Burkina Faso', 'Afr
ica', '15.605216979980469', '1327.2165314775982'), 'bwa': ('Botswana', 'Africa',
'2.0148661136627197', '12256.14159052444'), 'caf': ('Central African Republic', 'A
frica', '4.4485249519348145', '865.4421867263901'), 'cmr': ('Cameroon', 'Africa',
'19.970495223999023', '2684.972890247272'), 'cod': ('Democratic Republic of the Co
ngo', 'Africa', '64.52326202392578', '634.9713963013496'), 'cog': ('Republic of th
e Congo', 'Africa', '4.386693000793457', '4979.990496069269'), 'com': ('Comoros',
'Africa', '0.6896920204162598', '2292.997969425929'), 'cpv': ('Cape Verde', 'Afric
a', '0.5023840069770813', '5644.821253339075'), 'dji': ('Djibouti', 'Africa', '0.8
511459827423096', '2124.4420625972525'), 'dza': ('Algeria', 'Africa', '36.11763763
4277344', '12590.22422381359l'), 'egy': ('Egypt', 'Africa', '84.10760498046875',
'9148.839753298033'), 'eth': ('Ethiopia', 'Africa', '87.70266723632812', '1100.183
1875876223'), 'gab': ('Gabon', 'Africa', '1.6402100324630737', '11961.16822644887
9'), 'gha': ('Ghana', 'Africa', '24.512104034423828', '3931.2918319728856'), 'gi
n': ('Guinea', 'Africa', '10.794170379638672', '1742.7346742167783'), 'gmb': ('Gam
bia', 'Africa', '1.692149043083191', '2681.8586168502143'), 'gnb': ('Guinea-Bissa
u', 'Africa', '1.5558799505233765', '1387.2497310879905'), 'gnq': ('Equatorial Gui
nea', 'Africa', '0.9511039853096008', '31416.887137501908'), 'ken': ('Kenya', 'Afr
ica', '41.35015106201172', '2484.0349094725366'), 'lbr': ('Liberia', 'Africa', '3.
948124885559082', '786.6702306741524'), 'lso': ('Lesotho', 'Africa', '2.0405509471
89331', '2432.7976546880823'), 'mar': ('Morocco', 'Africa', '32.409637451171875',
'6421.937677907418'), 'mdg': ('Madagascar', 'Africa', '21.151639938354492', '1459.
9156124535612'), 'mli': ('Mali', 'Africa', '15.075084686279297', '1873.28096181958
65'), 'moz': ('Mozambique', 'Africa', '24.221405029296875', '969.2307622526259'),
'mrt': ('Mauritania', 'Africa', '3.6095430850982666', '3082.647028528842'), 'mus':
('Mauritius', 'Africa', '1.2479549646377563', '15178.325920597827'), 'mwi': ('Mala
wi', 'Africa', '15.167095184326172', '972.0456815116237'), 'nam': ('Namibia', 'Afr
ica', '2.1731700897216797', '7689.171239820006'), 'ner': ('Niger', 'Africa', '16.4
2557716369629', '845.8693682212395'), 'nga': ('Nigeria', 'Africa', '158.5782623291
0156', '5186.304307542348'), 'rwa': ('Rwanda', 'Africa', '10.246842384338379', '13
79.6993252327009'), 'sdn': ('Sudan', 'Africa', '34.385963439941406', '3608.7856176
907358'), 'sen': ('Senegal', 'Africa', '12.916229248046875', '2741.121456198507'),
'sle': ('Sierra Leone', 'Africa', '6.4587202072143555', '1161.8796942144647'), 'sw
z': ('Swaziland', 'Africa', '1.2028429508209229', '7042.524359035093'), 'syc': ('S
eychelles', 'Africa', '0.0914049968123436', '17960.353075613308'), 'tcd': ('Chad',
'Africa', '11.887202262878418', '1880.3748762799714'), 'tgo': ('Togo', 'Africa',
'6.5029520988464355', '1222.8150529133295'), 'tun': ('Tunisia', 'Africa', '10.6399
30725097656', '10647.834980284626'), 'tza': ('Tanzania', 'Africa', '44.82893753051
758', '1979.1718644815185'), 'uga': ('Uganda', 'Africa', '33.91513442993164', '172
3.5943355766267'), 'zaf': ('South Africa', 'Africa', '51.58466339111328', '11388.6
40515995063'), 'zmb': ('Zambia', 'Africa', '13.850032806396484', '2870.88726563423
14'), 'zwe': ('Zimbabwe', 'Africa', '14.08631706237793', '1479.0305839163732'), 'a
re': ('United Arab Emirates', 'Asia', '8.270684242248535', '59707.412565389604'),
'arm': ('Armenia', 'Asia', '2.8773109912872314', '8222.945234202914'), 'aze': ('Az
erbaijan', 'Asia', '9.03245735168457', '12947.102198682365'), 'bgd': ('Banglades
h', 'Asia', '152.14910888671875', '2411.1020691099325'), 'bhr': ('Bahrain', 'Asi
a', '1.2408620119094849', '37045.81310919623'), 'brn': ('Brunei', 'Asia', '0.38866
20104312897', '67320.89957245934'), 'btn': ('Bhutan', 'Asia', '0.727640986442565
9', '7235.18974818538'), 'chn': ('China', 'Asia', '1359.755126953125', '9337.29077
2677254'), 'cyp': ('Cyprus', 'Asia', '0.829446017742157', '28046.373877741087'),
'geo': ('Georgia', 'Asia', '4.231660842895508', '7966.602756952667'), 'hkg': ('Hon
g Kong', 'Asia', '7.02522087097168', '41687.946418899235'), 'ind': ('India', 'Asi
a', '1230.980712890625', '4357.0597360582315'), 'irn': ('Iran', 'Asia', '74.567512
51220703', '17328.457882894658'), 'irq': ('Iraq', 'Asia', '30.7627010345459', '934
4.543768350652'), 'isr': ('Israel', 'Asia', '7.42595911026001', '28638.903738530
5'), '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', '14.308739
66217041', '2330.127207632253'), 'kor': ('South Korea', 'Asia', '49.552856445312
5', '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', 'Asi
a', '0.536969006061554', '91982.39517540816'), 'mdv': ('Maldives', 'Asia', '0.3645
1101303100586', '12140.803390739566'), 'mmr': ('Burma', 'Asia', '50.155895233154
3', '3422.241340266976'), 'mng': ('Mongolia', 'Asia', '2.7126500606536865', '7670.
714034382584'), 'mys': ('Malaysia', 'Asia', '28.112289428710938', '17913.164410782
432'), 'npl': ('Nepal', 'Asia', '27.023136138916016', '1996.1961554516242'), 'om
n': ('Oman', 'Asia', '3.0414600372314453', '40472.674942509126'), 'pak': ('Pakista
n', 'Asia', '170.5601806640625', '4171.416562353057'), 'phl': ('Philippines', 'Asi
a', '93.72662353515625', '5391.233965240031'), 'qat': ('Qatar', 'Asia', '1.7796759
605407715', '123128.40152283433'), 'sau': ('Saudi Arabia', 'Asia', '27.42567634582
5195', '42331.65885721999'), 'sgp': ('Singapore', 'Asia', '5.074252128601074', '58
618.43503468222'), 'syr': ('Syria', 'Asia', '21.01883316040039', '5700.32943899239
9'), 'tha': ('Thailand', 'Asia', '67.20880889892578', '12496.244186726299'), 'tj
k': ('Tajikistan', 'Asia', '7.641630172729492', '2784.5145587763627'), 'tkm': ('Tu
rkmenistan', 'Asia', '5.087210178375244', '16061.429412691556'), 'tur': ('Turkey',
'Asia', '72.32691192626953', '17930.678781392427'), 'twn': ('Taiwan', 'Asia', '23.
140947341918945', '37188.89452901009'), 'uzb': ('Uzbekistan', 'Asia', '28.60629463
1958008', '6574.51672506692'), 'vnm': ('Vietnam', 'Asia', '88.4725112915039', '442
8.42140209062'), 'yem': ('Yemen', 'Asia', '23.606779098510742', '4553.55725844367
4'), 'alb': ('Albania', 'Europe', '2.9405250549316406', '9544.73991912729'), 'au
t': ('Austria', 'Europe', '8.40994930267334', '40489.80635849457'), 'bel': ('Belgi
um', 'Europe', '10.938738822937012', '38177.94724875522'), 'bgr': ('Bulgaria', 'Eu
rope', '7.404590129852295', '14906.784964031736'), 'bih': ('Bosnia and Herzegovin
a', 'Europe', '3.7220840454101562', '9049.364732987471'), 'blr': ('Belarus', 'Euro
pe', '9.473071098327637', '16457.02937904922'), 'che': ('Switzerland', 'Europe',
'7.831971168518066', '55688.020214268225'), 'cze': ('Czech Republic', 'Europe', '1
0.536286354064941', '26129.56935189834'), 'deu': ('Germany', 'Europe', '80.8947830
2001953', '40627.23054942445'), 'dnk': ('Denmark', 'Europe', '5.554843902587891',
'43416.2233861591 6'), 'esp': ('Spain', 'Europe', '46.788631439208984', '31610.9801
57042286'), 'est': ('Estonia', 'Europe', '1.3321019411087036', '20265.5725967049 3
7'), 'fin': ('Finland', 'Europe', '5.365781784057617', '38394.06158150017'), 'fr
a': ('France', 'Europe', '65.14578247070312', '35786.1616145054 8'), 'gbr': ('Unite
d Kingdom', 'Europe', '63.30684280395508', '34810.28120173958'), 'grc': ('Greece',
'Europe', '11.446004867553711', '25815.80076797162'), 'hrv': ('Croatia', 'Europe',
'4.328153133392334', '19305.008088868373'), 'hun': ('Hungary', 'Europe', '9.927840
232849121', '20477.79856764035'), 'irl': ('Ireland', 'Europe', '4.62692785263061
5', '47823.5081511363 4'), 'isl': ('Iceland', 'Europe', '0.3203279972076416', '3772
9.01607603124'), 'ita': ('Italy', 'Europe', '59.72980880737305', '34727.6726883470
5'), 'ltu': ('Lithuania', 'Europe', '3.123802900314331', '18475.371145517096'), 'l
ux': ('Luxembourg', 'Europe', '0.5078889727592468', '57882.81020537233'), 'lva':
('Latvia', 'Europe', '2.1188480854034424', '16943.688322829523'), 'mda': ('Moldov
a', 'Europe', '4.0844807624816895', '4173.634753274572'), 'mkd': ('Macedonia', 'Eu
rope', '2.0707390308380127', '11265.643175619889'), 'mlt': ('Malta', 'Europe', '0.
41611000895500183', '22983.44992639125'), 'mne': ('Montenegro', 'Europe', '0.62428
49826812744', '14960.94855221103'), 'nld': ('Netherlands', 'Europe', '16.682916641
23535', '44004.14602477084 6'), 'nor': ('Norway', 'Europe', '4.885878086090088', '7
3262.68174170727'), 'pol': ('Poland', 'Europe', '38.32340240478515 6', '21006.02730
9294514'), 'prt': ('Portugal', 'Europe', '10.652320861816406', '25788.273965224
7'), 'rou': ('Romania', 'Europe', '20.44034767150879', '16775.609043478133'), 'ru
s': ('Russia', 'Asia', '143.15386962890625', '21754.067899615977'), 'srb': ('Serbi
a', 'Europe', '7.291436195373535', '12453.482551916944'), 'svk': ('Slovakia', 'Eur
ope', '5.404294013977051', '23061.12099058148'), 'svn': ('Slovenia', 'Europe', '2.
045167922973633', '25831.33128718208'), 'swe': ('Sweden', 'Europe', '9.39016819000
2441', '40421.906436576974'), 'ukr': ('Ukraine', 'Europe', '45.7924995422363 3', '8
713.259627419637'), 'abw': ('Aruba', 'NorthAmerica', '0.10166899859905243', '3705
9.34143869512'), 'aia': ('Anguilla', 'NorthAmerica', '0.013768999837338924', '2109
8.057152999143'), 'atg': ('Antigua and Barbuda', 'NorthAmerica', '0.09466099739074
707', '17162.16102781353'), 'bhs': ('Bahamas', 'NorthAmerica', '0.360832005739212 0
4', '29504.581346144332'), 'blz': ('Belize', 'NorthAmerica', '0.3216080069541931',
'7145.952547655193'), 'bmu': ('Bermuda', 'NorthAmerica', '0.06395599991083145', '5
1447.14748191088 4'), 'brb': ('Barbados', 'NorthAmerica', '0.27956900000572205', '1
3995.612896540895'), 'can': ('Canada', 'NorthAmerica', '34.16866683959961', '4026
9.03526728651'), 'cri': ('Costa Rica', 'NorthAmerica', '4.5452799797058105', '1210
6.99191566891'), 'cuw': ('CuraÃ§ao', 'NorthAmerica', '0.14760799705982208', '2433
7.816177887613'), 'cym': ('Cayman Islands', 'NorthAmerica', '0.05550700053572655',
'36670.678481483985'), 'dma': ('Dominica', 'NorthAmerica', '0.07143999636173248',
'9101.993729581038'), 'dom': ('Dominican Republic', 'NorthAmerica', '9.89798545837
4023', '11500.13134401987'), 'grd': ('Grenada', 'NorthAmerica', '0.104676999151706
7', '9791.291133092265'), 'gtm': ('Guatemala', 'NorthAmerica', '14.63041687011718
8', '6359.262684615136'), 'hnd': ('Honduras', 'NorthAmerica', '8.194778442382812',
'3789.9102248753848'), 'hti': ('Haiti', 'NorthAmerica', '9.999616622924805', '165
0.2697128700804'), 'jam': ('Jamaica', 'NorthAmerica', '2.8172099590301514', '6675.
176956450172'), 'kna': ('Saint Kitts and Nevis', 'NorthAmerica', '0.05144499987363
815', '17837.349758162076'), 'lca': ('Saint Lucia', 'NorthAmerica', '0.17258000373
840332', '9178.266990673521'), 'mex': ('Mexico', 'NorthAmerica', '117.318939208984
38', '14507.011923865603'), 'msr': ('Montserrat', 'NorthAmerica', '0.0049439999274
9095', '15385.002991478228'), 'nic': ('Nicaragua', 'NorthAmerica', '5.737722873687
744', '3992.7619078961748'), 'pan': ('Panama', 'NorthAmerica', '3.643222093582153
3', '15055.237591848218'), 'slv': ('El Salvador', 'NorthAmerica', '6.1646261215209
96', '6096.958357099287'), 'tca': ('Turks and Caicos Islands', 'NorthAmerica', '0.
030993999913334846', '7230.966074605446'), 'tto': ('Trinidad and Tobago', 'NorthAm
erica', '1.3280999660491943', '27510.236190041924'), 'usa': ('United States', 'Nor
thAmerica', '308.6413879394531', '49500.62628346238'), 'vct': ('Saint Vincent and
the Grenadines', 'NorthAmerica', '0.10931500047445297', '8095.21923068351'), 'vg
b': ('British Virgin Islands', 'NorthAmerica', '0.027224000543355942', '21183.2791
65646647'), 'aus': ('Australia', 'Oceania', '22.12006378173828', '44854.9000486665
7'), 'fji': ('Fiji', 'Oceania', '0.8599500060081482', '6902.0687720944225'), 'nz
l': ('New Zealand', 'Oceania', '4.370061874389648', '30867.05437525178'), 'arg':
('Argentina', 'SouthAmerica', '41.2238883972168', '15841.658208158999'), 'bol':
('Bolivia', 'SouthAmerica', '9.918242454528809', '4806.608983377068'), 'bra': ('Br
azil', 'SouthAmerica', '196.7962646484375', '13541.462561602328'), 'chl': ('Chil
e', 'SouthAmerica', '16.99335479736328', '18092.94007371081')}
```

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

In [39]:

```
def fixData(element, cData):
    for key, value in cData.items():
        if element[1].lower() == value[0].lower(): # checking for the correct infor
            return (key, value[0], value[1])
    element = (element[0].lower(), element[1], element[2]) # if a country from the
    return element


fixedList = []
for element in countries:
    fixedList.append(fixData(element, cData))

print("The fixed list is:", fixedList, "\n")

continentCounts = {}

for _, _, continent in fixedList: # Iterate through the fixedList and count contine
    if continent in continentCounts:
        continentCounts[continent] += 1
    else:
        continentCounts[continent] = 1

# Find the highest count of continents
maxCount = max(continentCounts.values())

# Find all continents with the highest count
mostCommonContinents = [continent for continent, count in continentCounts.items()

if len(mostCommonContinents) > 1:
    print("Most common continents:", mostCommonContinents)
    print("Number of times that each continent appears in the list 'countries':", r
else:
    print("Most common continent:", mostCommonContinents)
    print("Number of times that the continent appears in the list 'countries':", ma
```

```
The fixed list is: [('prt', 'Portugal', 'Europe'), ('isr', 'Israel', 'Asia'), ('co
l', 'Colombia', 'South America'), ('nev', 'Nevada', 'North America'), ('jpn', 'Jap
an', 'Asia'), ('rus', 'Russia', 'Asia'), ('dnk', 'Denmark', 'Europe'), ('nor', 'No
rway', 'Europe')]

Most common continents: ['Europe', 'Asia']
Number of times that each continent appears in the list 'countries': 3
```

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

In [40]:

```
# Create a list of only country names
countryNames = [country.lower() for _, country, _ in fixedList]

nation = ""
amount = 0.0

for key, value in cData.items():
    if value[0].lower() in countryNames: # checking for the correct information in
        if (float(value[2]) > amount):
            amount = float(value[2])
            nation = value[0]

print("The most populated nation in the list 'countries' is",nation, 'with', amount
```

```
The most populated nation in the list 'countries' is Russia with 143.1538696289062
5 million people.
```

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?

In [43]:

```
def removeNonExistant(element, cData):
    for key, value in cData.items():
        if element[1].lower() == value[0].lower(): # checking for the correct infor
            return (key, value[0], value[1])
    return False

fixedList = []
for element in countries:
    if removeNonExistant(element, cData): # removing countries with no GDP. They wo
        fixedList.append(removeNonExistant(element, cData))

countryNames = [country.lower() for _, country, _ in fixedList]

averageGdpCountries = 0.0

for key, value in cData.items():
    if value[0].lower() in countryNames: # fetching the GDP if matches cData entry
        averageGdpCountries += float(value[3])

averageGdpCountries = averageGdpCountries / len(countryNames) # calculating average

print("\nThe average GDP for contries inside countries list is:", averageGdpCountri

averageGdpCdata = 0.0

for key, value in cData.items():
    averageGdpCdata += float(value[3])
averageGdpCdata = averageGdpCdata / len(cData.values()) # calculating average GDP j

print("The average GDP for contries inside cData is:", averageGdpCdata, "\n")

print("We can conclude that, the GDP of the average person in a country inside 'cou
```

```
The average GDP for contries inside countries list is: 38242.63072863724

The average GDP for contries inside cData is: 18014.65454394603

We can conclude that, the GDP of the average person in a country inside 'countrie
s', is bigger than the average GDP of a person in a country inside 'cData'
```

Part 3 - Functions hurt nobody

Exercise I - I hate math

Consider the following equation:

\begin{equation} y = 6x^2 + 3x + 2 \end{equation}
17: Write a function called *f* that takes one argument, x, and returns y according to the equation above.

In [44]:

```
def f(x):
    return 6*x**2 + 3*x + 2
```

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

In [45]:

```
f(2)
```

Out[45]:

```
32
```

Consider the following sequence of numbers:

\begin{equation} x_0 = 0; ~~~ ~~~ ~~~ ~~ ~~~ ~~~ ~~~ ~~ ~~~ ~~~ ~~~ ~~ ~~~ ~~~ ~~ ~ \\ x_{n} = x_{n-1} - n; ~~~ \textit{ if} ~~~ x_{n-1}-n > 0 \\ x_{n} = x_{n-1} + n; ~~~ \textit{ otherwise}~~~ ~~~ ~~~ ~~ \\ \end{equation}
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)

In [46]:

```
def recamanSequence(n):
    if n == 0: # first condition, returning 0
        return 0

    list1 = [0]
    for i in range(1, n + 1):
        last = list1[i - 1]
        if (last - i) not in list1 and last - i > 0: # checking which condition we
            # element is already in the sequence, because In Recamán's sequence, ec
            list1.append(last - i) # this one is the first condition, x(n-1) - n >
        else:
            list1.append(last + i) # otherwise
    return list1[n]

recamanSequence(8)
```

Out[46]:

```
12
```

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.

In [47]:

```
def prime(num):
    flag = False
    if num == 1:
        print(num, "is not a prime number")
    elif num > 1:
        for i in range(2, num + 1): # check for factors, number can only be divide
            if (num % i) == 0:
                if flag:
                    return False
                flag = True # if factor is found, set flag to True

        return True
```

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

In [51]:

```
print("Is 15485863 prime?", prime(15485863))
print("Is 2 prime?", prime(2))
print("Is 4 prime?", prime(4))
```

```
Is 15485863 prime? True
Is 2 prime? True
Is 4 prime? False
```

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.

In [54]:

```
import random

def MontyHall(switch): # switch = True | 1 -> switch doors; switch = False | 0 -> c
    doors = [0, 0, 1]  # 0 = goat, 1 = car
    random.shuffle(doors) # shuffle the doors with random

    choice = random.randint(0, 2) # player choice (not asking for input since we wo

    # find all doors non selected with goats
    hostDoor = [i for i in range(3) if i != choice and doors[i] == 0]
    hostOpened = random.choice(hostDoor) # from the, at most, 2 doors with goats, s

    remainingDoor = [i for i in range(3) if i != choice and i != hostOpened][0] # I

    if switch: # if True, player switches to the remaining door, else he keeps the
        choice = remainingDoor

    return doors[choice] == 1 # Check if the player won the car. Return value to su
```

In [56]:

```
MontyHall(True)
```

Out[56]:

```
True
```

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.

In [32]:

```
# Simulate the game with and without switching, and calculate the win rates
numSimulations = 10000
```

```
winsWithSwitch = sum(MontyHall(True) for i in range(numSimulations))
winsWithoutSwitch = sum(MontyHall(False) for i in range(numSimulations))

winRateWithSwitch = winsWithSwitch / numSimulations
winRateWithoutSwitch = winsWithoutSwitch / numSimulations

print("Win rate with switching:", winRateWithSwitch)
print("Win rate without switching:", winRateWithoutSwitch)
```

```
Win rate with switching: 0.6659
Win rate without switching: 0.334
```

The results of our 10000 simulations appear to be in line with the theoretical odds of the Monty Hall problem. The theoretical odds suggest that if we switch doors, the chance of winning the car is 2/3, while if we stick with the initial choice, the chance is 1/3.

The simulation results indicate that the win rate with switching doors is approximately 0.6659, which is very close to the theoretical odds of 2/3. Likewise, the win rate without switching doors is approximately 0.334, which is close to the theoretical odds of 1/3.

In conclusion, based on our simulations, the results are consistent with the theoretical odds of the Monty Hall problem, reinforcing the idea that switching doors increases the likelihood of winning the car.

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.

In [63]:
```
A = [460,3347,3044,490,699,1258,1804,973,2223,3416,2879,1058,2915,2422,351,1543,10
```

In [64]:
```
A = list(map(lambda x: x - 1 if x % 2 == 1 else x + 1, A))
print(A)
```

```
[461, 3346, 3045, 491, 698, 1259, 1805, 972, 2222, 3417, 2878, 1059, 2914, 2423, 3
50, 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.

In [65]:
```
B = list(map(lambda x: True if x > 700 else False, A))
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.

In [66]:
```
from math import log10
L = list(map(lambda x: log10(x), A))
print(L)
```

```
[2.663700925389648, 3.5245259366263757, 3.4835872969688944, 2.6910814921229687, 2.
843855422623161, 3.1000257301078626, 3.256477206241677, 2.9876662649262746, 3.3467
44054604849, 3.5336449787987627, 3.4590907896005865, 3.024895960107485, 3.46448954
74339714, 3.384353414137506, 2.5440680443502757, 3.188084373714938, 3.009025742086
9104, 2.3201462861110542, 2.807535028068853, 2.8998205024270964, 3.52322604196570
1, 3.4122925093230463, 2.6720978579357175, 3.4186326873540653, 3.0318122713303706]
```

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

Note: You should use the function filter().

```
In [67]: print('There are', len(list(filter(lambda x: True if x > 1000 else False, A))), 'nu
```

```
There are 16 numbers in A 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.

```
In [68]: print('Prime numbers in A:',list(filter(lambda x: True if prime(x) else False, A))
```

```
Prime numbers in A: [461, 491, 1259, 2423, 1021]
```

Congratulations, it is done!

```
In [ ]:
```