Programming for Data Science 2020

Homework Assigment One

Please Remember: 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. But 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. Focuses in data loading and analysis using the core elements of Python, without fancy third-party libraries;
3. Focus in functional programming and will require you to define different functions and operate with the map() and filter();

Comment your code properly, which includes naming your variables in a meaningful manner. Badly documented code will be penalized.

Your submission will be graded according to the following guidelines:

1. Execution (does your program does what is asked from the exercise?)
2. Objectivity (are you using the adequate libraries? are you using a libraries not in the scope of exercise?)
3. Readibility of your code (is your code correctly documented? is variable naming adequate?)

For the Homework Assignment One you are only allowed to use the standard libraries from Python, the use of libraries developed by third-party organizations is not allowed. For instance, libraries such as numpy, scipy, pandas, scikit-learn, etc are not allowed. If you have questions it is better to check with teaching staff.

Note: When you are asked to answer a question, for example the highest number in a list, you must answer this question using code. If you just look at the list to find the highest number and write it in the cell you will not receive any points.

This assignment is Individual, students that are caught cheating will obtain a score of 0 points. The Homeworking Assignment One is worth 15% of your final grade.

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

1. Jupyter Notebook with the output of all the cells;
2. PDF print of your Jupyter Notebook;
3. All text or csv files exported as part of the exercises.

Submission is done through the respective Moodle activity. Deadline is October 2nd at 23:59. A penality of 1 point per day late will be applied to late deliveries.

Part 1 - Variable Declaration and Manipulation

Exercise I - of Lists and Random numbers

Q: Declare a variable X that stores a list of 100 integers randomly sampled between -25 and 25.
Note: You are not allowed to use Numpy or Scipy in this exercise.

In [1]:
```
def randomList(lowerBound,upperBound,n):
    """
        This functions generates a list of n random integers comprised between the upperBound and lowerBound values.
        It allows repetition of integers.
        Inputs:
            lowerBound:int -> minimum allowed value for the random numbers
            upperBound:int -> maximum allowed value for the random numbers
            n:int -> number of elements required in the output list
        Outputs:
            list:int[n] -> list of n random integers comprised between the upperBound and lowerBound values
    """
    import random
    return [ random.randint(lowerBound,upperBound) for i in range(n)]

# use the defined function to generate a list of 100 random integers chosen between -25 and 25, with repetion
minimum_integer_allowed = -25
maximum_integer_allowed = 25
size_of_list = 100
X = randomList(minimum_integer_allowed,maximum_integer_allowed,size_of_list)
```

Q: Write a program to tell you how many odd numbers are in the list X.

Check to make sure you have the same number of odd and even numbers, if not discard the list and generate a new one.

Extra: Can you automatize this pipeline to avoid having to run multiple cells multiple times by hand?

In [2]:
```
def CountOddNumbersInList(l_toCount):
    """
        This functions counts the number of odd numbers in a list of integers.
        Inputs:
            l_toCount:list -> list of integers to count the odd numbers
        Outputs:
            int -> Number of Odd Numbers in the list
    """
    # We classify a odd number if the division remainer of dividing the number by 2 is different than 0.
    # To achieve this we use the operand %.

    # The approach here is to use the filter method to create a new list
    # where the only allowed elements inside are odd numbers and then compute it's length
    OddNumbers = len(list(filter(lambda x: x%2 != 0 ,l_toCount)))

    return OddNumbers

def randomBalancedList(lowerBound,upperBound,n):
    """
        This functions generates a balanced list (equal number of odd and even integers) of n random integers
        comprised between the upperBound and lowerBound values.

        It allows repetition of integers.

        If, the list size is not odd, it will not generate a list since it can never be balanced.

        Inputs:
            lowerBound:int -> minimum allowed value for the random numbers
            upperBound:int -> maximum allowed value for the random numbers
            n:int -> number of elements required in the output list
        Outputs:
            list:int[n] -> list of n random integers comprised between the upperBound and lowerBound values
    """
    if n % 2 == 0: # checks if the list size is even
        X = randomList(lowerBound,upperBound,n) # creates the first version of the list
        while CountOddNumbersInList(X) != int(n/2): # check if odd numbers = half of the size of the list
            X = randomList(lowerBound,upperBound,n) # re-creates the list
        return X
    else:
        raise Exception("You can't create a balanced list with an odd size.")

# use the defined function to generate a list of 100 random integers chosen between -25 and 25, with repetion
minimum_integer_allowed = -25
maximum_integer_allowed = 25
size_of_list = 100
X = randomBalancedList(minimum_integer_allowed,maximum_integer_allowed,size_of_list)

print("We have {} odd numbers in the list.".format(CountOddNumbersInList(X)))
```

```
We have 50 odd numbers in the list.
```

Q: 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. Note: we consider that the number -2 contains one digit.

In [3]:
```
def ExtractNumberOfDigitsOfANumber(Number):
    """
        This functions counts the number of digits of a provided Number. It only supports integers and floats.
        Inputs:
            Number:float/int -> integer to count number of digits
        Outputs:
            int -> Number of digits in the Number
    """
    if isinstance(Number,(float,int)):
        # If the number is a float/int
        # Logic:
        #  Compute the absolute value of the Number, to discard the - character.
        #  Convert the Number to a string
        #  Use the str.replace method to remove the . character (for the float possibility)
        #  Compute the length of the string
        return len(str(abs(Number)).replace('.',''))
    else:
        raise Exception("This function only supports integers or floats.")

# With the function defined, since the python lists index work on a offset basis,
# the 5th and 100th positions will have index 4 and 99 respectively.
print("The 5th element of the list is of value {} and has {} digits".format(X[4],
                                                                            ExtractNumberOfDigitsOfANumber(X[4])
                                                                           ))
print("The 100th element of the list is of value {} and has {} digits".format(X[99],
                                                                              ExtractNumberOfDigitsOfANumber(X[99])
                                                                             ))
```

```
The 5th element of the list is of value -22 and has 2 digits
The 100th element of the list is of value 11 and has 2 digits
```

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

In [4]:
```
def CheckIfListSumIsEven(l_toCheck):
    """
        This functions checks if the sum of the elements of the list is an even or an odd number.
        Inputs:
            l_toCheck:list -> List of Integers to sum and check the if it is even
        Outputs:
            boolean -> True for an even Number, False for an odd Number
    """
    # We classify an even number if the division remainer of dividing the number by 2 is equal than 0.
    # To achieve this we use the operand %.

    # Use the sum method to sum all the elements of the list and then return True if the division remainder is 0.
    return True if sum(l_toCheck) % 2 == 0 else False

if CheckIfListSumIsEven(X):
    print("The Total of the List is even.")
else:
    print("The Total of the List is odd.")
```

```
The Total of the List is even.
```

Q: What is the average of all the numbers in the list? What is the standard deviation?

In [5]:
```
def ComputeListAverage(l_toCompute):
    """
        This functions computes the average of the elements in the list.
        Inputs:
            l_toCompute:list -> List of integers/floats to average
        Outputs:
            float -> The average of all the elements in the list
    """
    # The average is defined by the sum of all the elements in the list,
    # divided by the number of elements in the list
    return sum(l_toCompute) / len(l_toCompute)

def ComputeListStandardDeviation(l_toCompute,mean=None):
    """
        This functions computes the Standard Deviation of the provided list.
        Inputs:
            l_toCompute:list -> List of integers/floats to average
            mean:float -> Optional, if not provided, the function will compute it. Otherwise,
                          it will use the provided value
        Outputs:
            float -> The standard deviation of the elements of the list
    """
    # if the Mean is None, it was not provided when the function was called and so,
    # we need to compute it using the ComputeListAverage function
    if mean==None:
        mean = ComputeListAverage(l_toCompute)

    # The variance is classified by the sum of the squared differences to the mean of all the elements in the list,
    # divided by the number of elements in the list
    variance = sum([((x - mean) ** 2) for x in l_toCompute]) / len(l_toCompute)

    # The standard variance is the squared root of the variance
    stddev = variance ** 0.5

    return stddev


Mean = ComputeListAverage(X)

stddv = ComputeListStandardDeviation(X,Mean)
# {:.3f} prints the float with 3 decimal characters
print("The list has an average of {:.3f} and a standard deviation of {:.3f}".format(Mean,stddv))
```

```
The list has an average of -0.180 and a standard deviation of 13.967
```

Q: Sort list X in descending order and store the result in variable Xsort.
Then replace each value in Xsort with index i as the sum of the values with index i-1 and i.
Note: Per definition Xsort[-1] = 0.

For example the list [1,2,3,4,5] would become [1,3,6,10,15]

In [6]:
```
# the list.sort method works by changing the list inplace, which we do not want.
#So we use the sorted method which is similar but returns another list
# using reverse=True to have the values in descending order
Xsort = sorted(X, reverse = True)

def AlgorithmImplementation(l):
    """
        This functions implements the following algorithm:
            for each index, assign to it the sum of the value of previous index + current index.
            The value of the first element is the first element.
            Example:
             AlgorithmImplementation([1,2,3,4,5]) = [1,3,6,10,15]
        Inputs:
            l:list -> List of integers/floats to apply the algorithm
        Outputs:
            list -> The list with the Algorithm applied
    """
    for i in range(len(l)):
        if i == 0: # if i== 0 the previous value is 0
            previousNumber = 0
        else: # Get the value of index (i-1).
            previousNumber = l[i-1]

        l[i] = previousNumber + l[i]
    return l

Xsort = AlgorithmImplementation(Xsort)
```

Exercise II - we have a gamer in the room

Q: 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.
What is the name of the client that bought the most videogames?

TIP: You will want to check the methods associated with string manipulation. See the links below:
https://python-reference.readthedocs.io/en/latest/docs/unicode/index.html (https://python-reference.readthedocs.io/en/latest/docs/unicode/index.html)
https://www.w3schools.com/python/python_strings.asp (https://www.w3schools.com/python/python_strings.asp)

In [7]:
```
purchases = {

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

}
```

In [8]:
```
clients = {

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

}
```

In [9]:
```
def ListNamesWithMoreThanNPurchases(d_purchases, d_clients, n):
    """
        This functions returns a list with the Names of the clients which have purchased more than n games.
        Note: This function assumes that the Client Codes in the purchases dictionary exist in the clients dictionary.
              If this is not true, it will return 'Unknown'.
        Inputs:
            d_purchases:dict -> Dictionary with the purchases made by the clients.
                                As keys, it requires the Client code.
                                As values, requires one string with all the purchases of the client comma separated.
            d_clients:dict -> Dictionary with the Client Names whom have purchased games.
                                As keys, it requires the Client code.
                                As values, requires one string with the Client Name.
            n:int -> Integer to compare against the amount of games purchased. If amount of games purchased > n,
                     it will export the name (if it exists in the clients, otherwise 'Unknown')
        Outputs:
            list -> The list of Client Names which have purchased more than n games
    """
    # Notes :
    #  1. The values of the purchase dictionary are strings with the all the games purchased separated by commas.
    #     To fix this, use the str.split method which returns a list with the string splitted by the requested character
    #     and count the number of elements. This will be the number of games purchased.
    #  2. Then, apply a filter function which will create a new list with all the keys in the d_purchases dictionary
    #     which meet the condition of the lambda function
    #  3. The lambda function here uses the previous knowledge of how to compute the number of games
    #     and checks if the number is greater than the requirement n
    ClientIdsWithMoreThanAPurchase = list(filter(lambda X: len(d_purchases[X].split(",")) > n, d_purchases))

    # for each client code (which are the keys in the customer dictionary) get the client name (which are the values)
    return [d_clients[id] if id in d_clients else 'Unknown' for id in ClientIdsWithMoreThanAPurchase ]

def CheckHighestBuyer(d_purchases,d_clients):
    """
        This functions returns the name of the person with the highest number of purchased games.
        Note: This function assumes that the Client Codes in the purchases dictionary exist in the clients dictionary.
              If this is not true, it will return 'Unknown'.
        Inputs:
            d_purchases:dict -> Dictionary with the purchases made by the clients.
                                As keys, it requires the Client code.
                                As values, requires one string with all the purchases of the client comma separated.
            d_clients:dict -> Dictionary with the Client Names whom have purchased games.
                                As keys, it requires the Client code.
                                As values, requires one string with the Client Name.
        Outputs:
            tup -> A tuple with the Client Code and Client Name (ClientCode,ClientName)
    """
    # To extract the client code of the client which has purchased more game, the max function is used
    # The object passed to the function is the d_purchases dictionary and they key used is number of games bough,
    # computed using the same approach as before
    HightBuyer_ClientCode = max(d_purchases, key = lambda x: len(d_purchases[x].split(",")))
    # With the client code (key in dictionary clients) get the client name (value of the key)
    HightBuyer_ClientName = d_clients[HightBuyer_ClientCode] if HightBuyer_ClientCode in d_clients else 'Unknown'

    # A tuple is returned with the Client code + client name
    return HightBuyer_ClientCode,HightBuyer_ClientName

# Calls the defined function with the dictionaries + the exercise requirement of purchased games, which is 1
l_Names = ListNamesWithMoreThanNPurchases(purchases,clients,1)
print("The following clients have purchased more than 1 game: {}".format(', '.join(l_Names)))

HB_ClientCode, HB_ClientName = CheckHighestBuyer(purchases,clients)
print("The person which purchased more games is {}, with Client Code  {}.".format(HB_ClientName,HB_ClientCode))
```

```
The following clients have purchased more than 1 game: Morty Smith, Mildred Ratched, Sheldon Cooper
The person which purchased more games is Sheldon Cooper, with Client Code  9823.
```

Part 2 - Data loading and analysis

Exercise I - Alice what do you have to say?

Important Make sure that the file alice.txt which was included with this homework is in the same folder where you have your notebook.

Q: Load the Alice text file into a variable called Alice.
Note1: You are not allowed to use third-party libraries like Pandas. Example code can be adapted from lab 2 or you can search the internet for help.
Note 2: The first two cells make sure that the alice.txt file was downloaded to the same folder as your notebook, now all you need to do is load it :) !

In [10]:
```
import sys
!conda install --yes --prefix {sys.prefix} requests
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

In [11]:
```
import requests
r = requests.get('https://www.dropbox.com/s/67pt7yaymxdw6up/alice.txt?dl=1')
open('alice.txt', 'wb').write(r.content)
```

Out[11]: 306

In [12]:
```
#write the code to load the dataset here:
# Opens the file in reading mode
with open('alice.txt','r') as f:
    # Logic:
    # 1. for each line read in the file, strip it's content (remove blank space before and after the first and last character,
    #    repectively)
    # 2. This will create a list of all the lines in the file
    # 3. Concatenate all the elements (lines of the file in this case) in the list with a blank space separating them,
    #    using the str.join method
    # 4. the s_Alice string will contain in a single string all the contents of the file.
    s_Alice = " ".join([line.strip() for line in f])
```

Q: 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: It is possible, and likely easier, to just use the inbuilt python methods available for string manipulation to complete this task. For those interrested one advanaced way to this is by following the links below for a discussion on regular expressions.

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

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

In [13]:
```
# Imported the string library (whis is part of the Python Standard Library), which contains the punctuation attribute.
# This attribute consists of a string containing all the possible ponctuation characters
# The same objective could be achieved by manually creating the string with the elements that we want to remove from another strin
g
# example, manual_ponctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
import string

# Creating a dictionary which will translate the ponctuation characters to None (effectively eliminating the ponctuation)
# The Keys of the dictionary are the unicode points of the characters, ex: character ! has unicode point 33
# so, chr(33) == '!' and ord('!') = 33
# The values of the dictionary are None
d_replacement = { ord(character):None for character in string.punctuation }

# Use str.translate method which uses the dict to perform replacements in the string
# Use str.lower() to put every character in lowercase
# Use str.split(" ") to split the string by " " character and create a list of words
wAlice = s_Alice.translate(d_replacement).lower().split(" ")
```

Using the list wAlice answer the following questions:
Q: How many words contains the text file Alice.txt?
Q: What is the longest and smallest word in the textfile? (Length here is measured in terms of the number of characters)
Q: Delete all the repeated words from wAlice. How many different words does the text contain? Hint: there is an easy way to do this in one line of code, but you can answer however you like.

In [14]:
```
NumberOfWordsInList = len(wAlice)
print("The file contains {} words.".format(NumberOfWordsInList))

print("The longest word in the file is {}.".format(max(wAlice, key = len))) # max length of all the elements in wAlice
print("The shortest word in the file is {}.".format(min(wAlice, key = len))) # min length of all the elements in wAlice

wAlice = set(wAlice) # sets do not have repeated elements, so converting to one eliminates all the duplicates
print("After deleting repeated words, we have {} words.".format(len(wAlice)))
```

```
The file contains 57 words.
The longest word in the file is conversations.
The shortest word in the file is a.
After deleting repeated words, we have 41 words.
```

Exercise II - I Love Economics

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 [15]:
```
countries = ['PAN','ARG','NGA','KOR','SRB','BIH']
```

Consider the list country_data in the cell below. It is a list of tuples where each tuple has the following variables:

The first value is the 3-digit ISO code of a country.
The second value is the name of the country.
The third value is the continent in which the country is localized.
The fourth value is the population size of the country (in millions).

In [16]:
```
country_data = [
 ('arg', 'Argentina', 'SouthAmerica', 41.2238883972168),
 ('geo', 'Georgia', 'Asia', 4.231660842895509),
 ('kor', 'South Korea', 'Asia', 49.5528564453125),
 ('swz', 'Swaziland', 'Africa', 1.2028429508209229),
 ('cog', 'Republic of the Congo', 'Africa', 4.386693000793457),
 ('srb', 'Serbia', 'Europe', 7.291436195373535),
 ('pan', 'Panama', 'NorthAmerica', 3.643222093582153),
 ('ita', 'Italy', 'Europe', 59.72980880737305),
 ('dma', 'Dominica', 'NorthAmerica', 0.07143999636173247),
 ('nga', 'Nigeria', 'Africa', 158.57826232910156),
 ('bih', 'Bosnia and Herzegovina', 'Europe', 3.722084045410156)]
```

Q: Create a dictionary that allows us to map the 3-digit ISO code of a country to it's name.

In [17]:
```
def IsoCodeToCountryName_Hip1(l_country_data):
    """
        This functions returns a dictionary which maps 3-digit iso codes to a country name.
        Inputs:
            l_country_data:list -> List of country data where each element is a tuple where:
                                   index0 string : 3-digit iso code.
                                   index1 string : country name.
                                   index2 string : continent.
                                   index3 float  : population
        Outputs:
            dict -> key string : 3-digit code of the country
                    value string : country name
    """
    # index 0 = iso code
    # index 1 = country name
    # for each element in country_data, create dict with position 0 and 1 of the tuple
    return {data[0]:data[1] for data in l_country_data}

ISOtoCountryNameMap = IsoCodeToCountryName_Hip1(country_data)
ISOtoCountryNameMap
```

Out[17]:
```
{'arg': 'Argentina',
 'geo': 'Georgia',
 'kor': 'South Korea',
 'swz': 'Swaziland',
 'cog': 'Republic of the Congo',
 'srb': 'Serbia',
 'pan': 'Panama',
 'ita': 'Italy',
 'dma': 'Dominica',
 'nga': 'Nigeria',
 'bih': 'Bosnia and Herzegovina'}
```

Q: Using country_data, identify what is the most common Continent among the Nations in the list countries.

In [18]:
```
def CheckMostCommonContinent(l_countries,l_country_data):
    """
        This functions returns the name(s) of the most common continent(s) in a given list of country 3 digit iso codes.
        Inputs:
            l_countries:lsit -> List of 3-digit iso codes of countries to check.
            l_country_data:list -> List of country data where each element is a tuple where:
                                    index0 string : 3-digit iso code.
                                    index1 string : country name.
                                    index2 string : continent.
                                    index3 float  : population
        Outputs:
            tup -> A tuple with:
                    index0 integer : the ocorrences of the most common contrie(s)
                    index1 list : list of the continents with the highest occorrences
    """

    # first, we need to extract the list of continents (index 2 of the tuple) of the countries in the countries list
    Continents = [data[2] for data in country_data if data[0].upper() in countries]

    # compute the number of ocorrences of each continent
    CountContinents = {Continent:Continents.count(Continent) for Continent in Continents}

    # find the maximum number of occorences
    MaxOcorrences = max(CountContinents.values())

    # find the continent with the highest occorence
    Continents = [Continent for Continent, Occorrences in CountContinents.items() if Occorrences == MaxOcorrences]

    return MaxOcorrences, Continents

MaxOcorrences, Continents = CheckMostCommonContinent(countries,country_data)

print("With {} ocorrences, {} {} the most common continents on the list.".format(MaxOcorrences,
                                                                                 ' and '.join(Continents),
                                                                                 'is' if len(Continents) == 1 else 'are'
                                                                                 ))
```

```
With 2 ocorrences, Europe is the most common continents on the list.
```

Q: Using country_data, identify the most populated nation among the countries list.

In [19]:
```
def CheckMostPopulatedCountry(l_countries,l_country_data):
    """
        This functions returns the name(s) of the most common country/countries in a given list of country 3 digit iso codes.
        Inputs:
            l_countries:lsit -> List of 3-digit iso codes of countries to check.
            l_country_data:list -> List of country data where each element is a tuple where:
                                    index0 string : 3-digit iso code.
                                    index1 string : country name.
                                    index2 string : continent.
                                    index3 float  : population
        Outputs:
            tup -> A tuple with:
                    index0 integer : the population of the most populated country/countries
                    index1 list : list of the countries with the highest population
    """
    # create new dict with the countries in country_data which exist in countries
    DataForCountriesInList = list(filter(lambda x: x[0].upper() in l_countries ,l_country_data))

    # get the max, using as key the index 3 of the tuple, which is the population
    HighestPopulation = max(DataForCountriesInList, key = lambda x: x[3])[3]

    # filter the list for the countries which have a population number as high as the highest
    CountriesWithHighestPopulation = list(filter(lambda x: x[3] == HighestPopulation,l_country_data))

    l_CountrieNamesWithHighestPopulation = [Country[1] for Country in CountriesWithHighestPopulation]

    return HighestPopulation, l_CountrieNamesWithHighestPopulation

HighestPopulation, HighestPopulatedCountries = CheckMostPopulatedCountry(countries,country_data)

print("With a population of {}, {} {} the most populated countries on the list.".format(HighestPopulation,
                                                                                        ' and '.join(HighestPopulatedCountries),
                                                                                        'is' if len(HighestPopulatedCountries) ==
1 else 'are'
                                                                                        ))
```

```
With a population of 158.57826232910156, Nigeria is the most populated countries on the list.
```

Part 3 - Functions hurt nobody

Exercise I - I hate math

Consider the following equation:

$$y = 6x^2 + 3x + 2$$

Q: write a function called f that takes one argument, x, and returns y according to the equation above. Call the function for x = 2 and print the answer.

In [20]:
```
def computeValueOfEquation(x):
    """
        This functions computes the value of the following function y = 6x^2 + 3x + 2
        Inputs:
            x:float -> integer/float which will be used in the calculation.
        Outputs:
            float -> the result of the application of the function
    """
    return 6*(x**2) + 3*(x) + 2

computeValueOfEquation(2)
```

Out[20]: 32

The following exercise was updated in version2

Consider the following sequence of numbers:

$$x_0 = 0;$$
$$x_n = x_{n-1} - n;\ if\ \ x_{n-1} - n > 0\ and\ not\ already\ in\ the\ sequence$$
$$x_n = x_{n-1} + n;\ otherwise$$

Q: 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 [21]:
```
def RecamanSequence(n):
    """
        This functions implements the Recaman sequence and returns the nth element of the sequence.
        Recaman Sequence Logic:
            1. X[0] = 0
            2. if ( X[n-1] - n > 0 ) and ( X[n-1] - n is not in the sequence ), X[n] = X[n-1] - n
            3. otherwise, X[n] = X[n-1] + n
        Inputs:
            n:integer -> the nth position that needs to be returned by the function.
        Outputs:
            int -> the value of the nth position of the Recaman's sequence
    """
    if n == 0:
        # X[0] = 0
        return 0
    else:
        # initialize list with the first member
        l = [0]

        # looping from start = 1 (since first position is already defined), end at n and with steps = 1
        for i in range(1, n, 1):
            # checks the condition of the Recaman logic, if the value of the sequence in the previous index - the current index
            # is greater than 0 and if the value compute doesn't exist in the sequence already
            value = l[i-1] - i
            if value > 0 and value not in l:
                # if the condition is True, append the value to the list
                l.append(value)
            else:
                # if the condition is False, computes the new value by summing the value of the sequence in the previous
                # index + the current index and appends it to the list
                l.append(l[i-1] + i)
        # Arriving at the end of the loop, our sequence is compute and we just need to return the last element of the list
        return l[-1]
```

In [22]:
```
# According to wikipedia, the 71th element of the Recaman sequence is 155, using our implementation we can see that it is correct
RecamanSequence(71)
```

Out[22]: 155

Exercise II - Role Playing with Python

Q: Create a function (called purchase_calculator) that will be used by a videogames store manager. This function recieves a list of number of codes of videogames, and a dictionary that has all the video games codes as keys and their cost as value.
This function should use the list and dictionary to return the total cost of the purchase.

In [23]:
```
def purchase_calculator(l_codes,d_cost):
    """
        This functions returns the total cost of a purchase by providing the list of item codes + the dictionary with the cost per
item.
        Inputs:
            l_codes:list -> List with the item codes of the purchase
            d_cost:dict -> keys: string with item codes
                           values: float with price per item code
        Outputs:
            float -> the total sum of the purchase
    """
    # Logic:
    # 1. Create a list with the price of each item code in the l_codes list
    # 2. Sum all elements of the new list
    return sum([d_cost[code] for code in l_codes])
```

Q: Create a function named random_nr that takes a number (e.g., m) as an argument and returns a random number from 1 to m.

In [24]:
```
def random_nr(m):
    """
        This functions returns one random number comprised in the interval [1,m].
        Inputs:
            m:integer -> Upperbound of the random generator
        Outputs:
            integer -> Random Integer
    """
    import random
    return random.randint(1,m)
```

Q: Create a function named key_returner that takes a number (n) and a dictionary (s) as arguments and returns n random keys from s.

In [25]:
```
def key_returner(n,s):
    """
        This functions returns a random n amount of keys from a dictionary.
        Inputs:
            n:integer -> Number of random keys to extract
            s:dict -> keys: string with item codes
                      values: string with the item names
        Outputs:
            list -> List of n random item codes
    """
    # It was not specified in the exercise 2 details:
    # 1. Object type of the return of the function. ASSUMPTION: List
    # 2. Can the item codes be duplicated? ASSUMPTION: Yes
    if n >= 0:
        import random
        # Create a list with the keys of the s dictionary
        Codes = list(s.keys())

        # Generate n elements and for each iteration, compute a random index in the interval of possible indexes
        ListOfRandomCodes = [ Codes[random.randint(0,len(Codes)-1)] for i in range(n)]

        return ListOfRandomCodes
    else:
        raise Exception("Number of elements needs to be greater or equal than 0.")
```

Q: Create a function named bool_gen that takes a number (p) as an argument and returns True with probability p, else it returns False.

NOTE: Your function needs to consider weather p is a number from 0 to 1 (and hence is the probability itself) or if the number is between 0 to 100 (and hence is in percentages and needs to be changed to a probability). If the value inserted is inferior to 0 or bigger than 100 the function should assume p to be 0.5

To clarify If p is 0.45 we will have a 45% probability to return True. If p is 55, it needs to be converted to 0.55 (having the same meaning as previously stated : we have a 55% probability of returning true

In [26]:
```
def bool_gen(p):
    """
        This functions returns True (boolean) with p probability.
        Notes:
            1. If p is in range [0,100] a convertion is applied to the range [0,1]
            2. p == 1 is considered part of the range [0,1]
            3. If p < 0 or p > 100, we assume p = 0.5
            4. If p is not convertible to float, we assume p = 0.5
        Inputs:
            p:float/int/string -> Probability of Success
        Outputs:
            boolean -> True with probability p, False with probability 1-p
    """
    import random

    # try to cast as float
    try:
        f_p = float(p)
    except:
        # Value provided is not convertible to a float, assume p=0.5
        f_p = 0.5

    # Deal with wrong probability numbers in both formats
    if f_p < 0.0 or f_p > 100.0:
        f_p  = 0.5

    # If the probability given is inside the interval ]1,100] we assume that it is the range of [0,100] and divide p by 100.0
    if f_p > 1.0:
        f_p = f_p / 100.0

    # generates a random number and check if it is value is lower than p (success, True) or not (failure, False)
    return random.random() < f_p
```

Q: Create a function named apply_discount that receives a number (v) and a boolean (b) as arguments. If b is true then the function should apply a discount of 15% to value, and return it. Otherwise it should return the value as is.

In [27]:
```
def apply_discount(v,p):
    """
        This functions applies a percentage reduction to a value if given p==True, else will return the value unchanged.
        Inputs:
            p:boolean -> Boolean which specifies if a discount will be applied or not
        Outputs:
            float -> Final value
    """
    if p:
        # if True, reduce 15% in the value
        return v - (v * 0.15)
    else:
        # if False, maintain the value
        return v
```

Q: Create a function called customerSim. This function takes as an input two dictionaries (stock and prices).
Then it should perform the following steps:

1. Call random_nr to generate a number between 1 and the maximum number of videogames available in the store's stock (this number will represent the number of videogames that will be purchased);
2. Call key_returner, passing the number of videogames that will be purchased obtained from 1. and the stock dictionary, in order to determine what are the codes of the videogames to be purchased;
3. Call apply_discount to each value in the customer order and combined with the boolGen, decide whether you apply a discount or not to each item;
4. Print the following message "Dear customer, your order of Y items has a total cost of X euros, and a 15% discount was applied to W items.", replacing X,Y, W with the respective values.
5. Return a tuple with the number of items ordered, total cost, and the total discount applied.

Run the function customerSim to simulate the orders of 10 clients.

In [28]:
```
#Write your function here. You have the stock and prices dictionaries below :)
def customerSim(stock,prices):
    MaxStock = len(stock)

    NumberOfVideoGamesToBuy = random_nr(MaxStock)

    CodesVideoGamesToBuy = key_returner(NumberOfVideoGamesToBuy,stock)

    # Exercise does not specify the value to use in the bool_gen function.
    # I checked with prof Flavio which said that assuming a fixed p was ok
    ProbabilityOfDiscount = 0.2

    # Generate a boolean list which will dictate if a discount is to be applied or not
    ApplyingDiscount = [bool_gen(ProbabilityOfDiscount) for i in range(NumberOfVideoGamesToBuy)]

    # Compute the total number of discounts by filtering the list for the True values and counting the resulting list
    TotalDiscounts = len(list(filter(lambda x : x, ApplyingDiscount)))

    # Zip the 2 lists: Item codes to buy and, if for each item code, will be applied discount or not
    ValuesToCharge = [ apply_discount(prices[Code],Discount) for Code, Discount in zip(CodesVideoGamesToBuy,ApplyingDiscount) ]
    TotalValue = sum(ValuesToCharge) # Sum the Values to charge to get the total price of the transaction

    # Compute the price without discounting
    TotalValueWithoutDiscount = purchase_calculator(CodesVideoGamesToBuy,prices)

    TotalDiscountApplied = TotalValueWithoutDiscount - TotalValue

    # Printing cost rounded to the second decimal case, which replicates what happens in the retailer world
    print("Dear customer, your order of {} items has a total cost of {:.2f} euros, and a 15% discount was applied to {} items.".fo
rmat(NumberOfVideoGamesToBuy,

TotalValue,

TotalDiscounts))
    return (NumberOfVideoGamesToBuy,TotalValue,TotalDiscountApplied)
```

In [29]:
```
stock = {

    "234":"God of War",
    "956":"Call of Duty: Modern Warfare 2",
    "365":"Final Fantasy IX",
    "827":"BioShock Infinite",
    "106":"World of Goo",
    "465":"Metal Gear Solid V: The Phantom Pain",
    "622":"Portal 2",
    "266":"The Last of Us",
    "534":"The Legend of Zelda: Majora's Mask",
    "884":"Halo 2",
    "472":"Red Dead Redemption",
    "123":"Grand Theft Auto: San Andreas"

}

prices = {

    "234":49.99,
    "956":35.0,
    "365":68.99,
    "827":20.99,
    "106":2.50,
    "465":49.99,
    "622":19.99,
    "266":19.99,
    "534":5.99,
    "884":9.99,
    "472":19.99,
    "123":24.99

}
```

In [30]:
```
#Call your function here :)
for i in range(10):
    customerSim(stock,prices)
```

```
Dear customer, your order of 9 items has a total cost of 214.42 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 5 items has a total cost of 82.96 euros, and a 15% discount was applied to 2 items.
Dear customer, your order of 12 items has a total cost of 349.40 euros, and a 15% discount was applied to 2 items.
Dear customer, your order of 4 items has a total cost of 105.72 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 3 items has a total cost of 82.13 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 4 items has a total cost of 100.46 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 5 items has a total cost of 122.08 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 3 items has a total cost of 69.48 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 7 items has a total cost of 182.93 euros, and a 15% discount was applied to 1 items.
Dear customer, your order of 10 items has a total cost of 217.80 euros, and a 15% discount was applied to 3 items.
```

Exercise II

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

In [31]:
```
A = [460,3347,3044,490,699,1258,1804,973,2223,3416,2879,1058,2915,2422,351,1543,1020,208,643,795,3337,2585,471,2623,1077]
```

In [32]:
```
A = list(map(lambda x: x+1 if x%2==0 else x-1,A))
```

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

In [33]:
```
B = list(map(lambda x: x > 700,A))
```

Q: Create a list L that contains the Logarithm of base 10 of each value in A.
Note: You should use the module math to compute the logarithm.

In [34]:
```
import math

L = list(map(lambda x: math.log10(x), A))
```

Q: How many numbers in A are greater than 1000?
Note: You should use the function filter()

In [35]:
```
# Filter the list for the elements which meet the criteria of being larger than 1000
# Count the size of the resulting list
GreaterThan1000 = len(list(filter(lambda x: x>1000,A)))
print("There are {} numbers greater than 1000.".format(GreaterThan1000))
```

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

Congratulations, it is done!
Don't forget to post any questions in the Forum or Microsoft Teams.