1. Body mass index calculator

Write a function that calculates the BMI entered by the user, that is, whoever runs it will have to enter this data.

You can get more information about its calculation at: 🔗 Body Mass Index What is BMI and how is it calculated.

The function must classify the result into its respective categories.

Tip: Try validating the data beforehand, so that it sends a warning message if the data entered by the user is not in

the appropriate format or does not take reasonable values.

```
#Calculte BMI

def calculate_bmi(Weight,Height):

   bmi=Weight / (Height ** 2)



   if bmi< 18.5:

     category = "under Weight"

   elif 18.5 <= bmi <24.9:

      category =  "Normal Weight"

   elif 25 <= bmi < 29.9:

      category =  "Over Weight"

   else:

       category ="Obesity"


   return bmi,category
```

```
def get_userinput():
#Get the input from the user
    Weight = input("Enter Your Weight in KG")
    Height = input("Enter Your Height in Meters:")

# Check if input is numeric
    if not(Weight.replace('.','',1).isdigit() and Height.replace('.','',1).isdigit()):
        print("Enter Numeric value only")
        return None,None

    Weight= float(Weight)
    Height =float(Height)
#check the input is positive Values
    if Weight <=0 or Height <=0:
        print("Weight and Height must be greater than 0")
        return None,None

    return Weight,Height
# Main Program
Weight,Height = get_userinput()

if Weight is not None and Height is not None:
    bmi,category = calculate_bmi(Weight,Height)
    print(f"Your BMI is {bmi:.2f}.You are categorized as {category}.")
```

Enter Your Weight in KG 50

Enter Your Height in Meters: 1.53

Your BMI is 21.36.You are categorized as Normal Weight.

2. Temperature converter

There are several temperature units used in different contexts and regions. The most common are Celsius (°C), Fahrenheit (°F), and Kelvin (K).

There are also other units such as Rankine (°Ra) and Réaumur (°Re).

Select at least 2 converters, so that when entering a temperature, it returns at least two conversions,

so that they can be saved (remember that a print() can never be saved).

Tip: Try validating the data beforehand, so that it sends a warning message if the data entered by the user is not in the correct format.

(EXTRA): Think of a way to store all possible conversions in a single object (List? Dictionary? DataFrame?)

instead of writing many if elses based on the source temperature and the destination temperature.

```
def conversion_temp(celsius):

    #Convert Celsius to Fahrenheit and kelvin and return the values

    farenhit = (celsius * 9 / 5) + 32

    kelvin = celsius + 273.15

    return farenhit,kelvin


def get_userinput():

#get the input from user in celsius
```

```
    celsius = input("Enter the value in celsius:")

    if not celsius.replace('.','',1).lstrip('-').isdigit():
        print("Enter numeric values only")
        return None

    celsius = float(celsius)
    return celsius

#Main program
celsius =get_userinput()

if celsius is not None:
    farenhit,kelvin = conversion_temp(celsius)
    print(f"Fahrenheit:{farenhit:.2f},Kelvin:{kelvin:.2f}")
```

Enter the value in celsius: 50

Fahrenheit:122.00,Kelvin:323.15

3. Word counter for a text.

Write a function that, given a text, displays the number of times each word appears.

Try to handle all possible cases that cause the program not to work correctly.

(EXTRA): What is the average word length of the text you wrote? “Hello, how are you?” should return (4+3+2) / 3 = 3

```
import string
```

```
def clean_text(text):
# Convert text to lowercase and remove the punctuation
    text= text.lower()
    for p in string.punctuation:
        text = text.replace(p,"")
    return text

def count_words(words):
#count the frequency of each word in the list
    word_count= {}
    for word in words:
        word_count[word] = word_count.get(word,0) + 1
    return word_count
def average_wordlength(words):
#calculate average word length
    if not words:
        return 0
    return sum(len(w) for w in words) / len(words)

#Main program
text = input("Enter a text:")
cleaned_text = clean_text(text)
words = cleaned_text.split()

if not words:
```

```
    print("No words found")

else:
    word_count = count_words(words)
    print("Word frequencies:")
    for word,count in word_count.items():
        print(f"{word}:{count}")

    avg_len = average_wordlength(words)
    print(f"Average word length:{avg_len:.2f}")
```

Enter a text: hi how r u how r u

Word frequencies:

hi:1

how:2

r:2

u:2

Average word length:1.71

4.Reverse dictionary (with possibility of duplicates)

It turns out that the client has a very old survey stored in a dictionary and needs the results in reverse,

i.e., by swapping the keys and values. The values and keys in the original dictionary are unique;

if this is not the case, the function should print a warning message, along with a list of the values associated with the repeated key.

```
# Program to create a dictionary from user input and flip its keys and values
```

```
def create_dict():
    ini_dict = {}
    n = int(input("Enter number of items in the dictionary: "))

# Loop to take key-value pairs from the user
    for _ in range(n):
        key = input("Enter key: ")
        value = input("Enter value: ")
        ini_dict[key] = value  # Add the pair to the dictionary

    print("initial_dictionary", str(ini_dict))
    return ini_dict

def flip_dict(ini_dict):
    flipped = {}
# Loop through the original dictionary to flip keys and values

    for key, value in ini_dict.items():
        if value not in flipped:
            flipped[value] = [key]
        else:
            flipped[value].append(key)

    print("final_dictionary", str(flipped))
    return flipped
```

```
ini_dict = create_dict()

flipped = flip_dict(ini_dict)
```

```
Enter number of items in the dictionary:  3

Enter key:  a

Enter value:  banana

Enter key:  b

Enter value:  apple

Enter key:  c

Enter value:  banana

initial_dictionary {'a': 'banana', 'b': 'apple', 'c': 'banana'}

final_dictionary {'banana': ['a', 'c'], 'apple': ['b']}
```

Level 2

1. Counter and sorter of words in a text.

The client was happy with the word counter, but now he wants to read TXT files and calculate

the frequency of each word ordered within the usual dictionary entries according to the letter they begin with,

that is, the keys must go from A to Z and within A we must go from A to Z. For example, for the file

```
# Program to read a text file, count word frequencies, and organize them alphabetically (A–Z)

import string

from pprint import pprint  # For neat output formatting


def process_textfile():
```

```
text = open("C:/Users/sarav/Downloads/tu me quieres blanca.txt","r")
d=dict()
# Read each line from the text file
for line in text:
    line=line.strip()
    line=line.lower()
    line = line.translate(str.maketrans("", "", string.punctuation))# Remove punctuation
    words=line.split(" ")

# Count each word's occurrences
    for word in words:
        if word in d:
            d[word]=d[word]+1
        else:
            d[word]=1
text.close()

sorted_words = sorted(d.items())

nested_dict = {}

for word, count in sorted_words:
    if not word:
        continue    # Skip empty words if any
    first_letter = word[0].upper()
```

```
        if first_letter not in nested_dict:
            nested_dict[first_letter] = {}
        nested_dict[first_letter][word] = count

    # Sort each sub-dictionary alphabetically
    for letter in nested_dict:
        nested_dict[letter] = dict(sorted(nested_dict[letter].items()))

    print("\n\n=== Nested Dictionary (A–Z) ===")
    pprint(nested_dict)
#program Execution
process_textfile()
```

```
=== Nested Dictionary (A–Z) ===
{'A': {'a': 3,
       'agua': 1,
       'al': 2,
       'alba': 4,
       'alcobas': 1,
       'alimenta': 1,
       'alma': 1,
       'amarga': 1,
       'azucena': 1},
 'B': {'baco': 1,
       'banquete': 1,
       'bebe': 1,
```

'blanca': 3,
'boca': 1,
'bosques': 1,
'buen': 1},
'C': {'cabaã±as': 1,
'carnes': 2,
'casta': 3,
'cerrada': 1,
'con': 4,
'conservas': 1,
'copas': 1,
'corola': 1,
'corriste': 1,
'cuando': 2,
'cubierto': 1,
'cuerpo': 1,
'cuã¡les': 1},
'D': {'de': 8, 'dejaste': 1, 'del': 1, 'diga': 1, 'dios': 2, 'duerme': 1},
'E': {'el': 4,
'ellas': 1,
'en': 4,
'engaã±o': 1,
'enredada': 1,
'entonces': 1,
'escarcha': 1,
'espumas': 1,

'esqueleto': 1,
'estrago': 1},
'F': {'festejando': 1, 'filtrado': 1, 'frutos': 1},
'H': {'habla': 1,
'hacia': 1,
'haya': 1,
'hayas': 1,
'hermana': 1,
'hombre': 1,
'hubiste': 1,
'huye': 1},
'I': {'intacto': 1},
'J': {'jardines': 1},
'L': {'la': 3,
'labios': 1,
'las': 7,
'lo': 2,
'los': 4,
'luna': 1,
'lã©vate': 1,
'lã\xadmpiate': 1},
'M': {'mano': 1,
'manos': 1,
'margarita': 1,
'me': 9,
'mi': 1,

'mieles': 1,

'milagros': 1,

'mojada': 1,

'montaã±a': 1,

'morados': 1},

'N': {'negros': 1, 'ni': 2, 'no': 1, 'nã¡car': 1, 'nã\xadvea': 2},

'P': {'perdone': 2,

'perfume': 1,

'por': 2,

'pretendes': 3,

'pretã©ndeme': 3,

'puesto': 1,

'pã¡jaros': 1,

'pã¡mpanos': 1},

'Q': {'que': 6, 'quedã³': 1, 'quieres': 6},

'R': {'rayo': 1, 'raã\xadz': 1, 'renueva': 1, 'rocas': 1, 'rojo': 1},

'S': {'salitre': 1, 'se': 2, 'sea': 1, 'sean': 1, 'sobre': 2, 'sã©': 1},

'T': {'te': 3,

'tejidos': 1,

'tenue': 1,

'tierra': 1,

'toca': 1,

'todas': 2,

'todavã\xada': 1,

'tornadas': 1,

'tãº': 8},

'U': {'un': 1, 'una': 1},

'V': {'vestido': 1, 'vete': 1, 'vive': 1},

'Y': {'y': 5},

'Â': {'â¡me': 1}}

2. Data type conversion.

The client receives a list of data and needs to generate two lists,

the first where all the elements that could be converted to floats will be and the other where the elements that could not be converted are.

Example of the list that the client receives:

```
# Program to separate float-convertible values from non-float values in a list (including nested lists or tuples)
def conversion(data):
    float_list = []
    non_float_list = []

    for item in data:
        # If item is a list or tuple → check each element
        if isinstance(item, (list, tuple)):
            for sub_item in item:
                try:
                    float_list.append(float(sub_item))
                except (ValueError, TypeError):
                    non_float_list.append(sub_item)
        else:
```

```
        try:
            float_list.append(float(item))
        except (ValueError, TypeError):
            non_float_list.append(item)

    return (float_list, non_float_list)



#  Get user input at runtime
import ast
data = ast.literal_eval(input("Enter your list: "))

# Call function
result = conversion(data)

print(result)
```

Enter your list: ['1.3', 'one', '1e10', 'seven', '3-1/2', ('2', 1, 1.4, 'not-a-number'), [1, 2, '3', '3.4']]

([1.3, 10000000000.0, 2.0, 1.0, 1.4, 1.0, 2.0, 3.0, 3.4], ['one', 'seven', '3-1/2', 'not-a-number'])

Level 3

1. Password Generator

Explore the operation of the random module of the numpy library. 🔗 Random module in NumPy

At this point, the client has detected a problem with the passwords used by their employees.

Asdf1234, birthdays or similar. To solve this, they have commissioned us to create a Python function that generates more secure passwords.

The function must depend on the following parameters:

length (int): Password length

capital letters (bool = True): Whether capital letters should appear

lowercase (bool = True): Whether lowercase letters should appear

numbers (bool = True): Whether numbers should appear

signs (bool = False): If special characters should appear (,-$? or similar)

So, if we execute the function as follows:

```
create_password(10, True, True, True, True)
```

We should obtain an output (which we must be able to save) like this:

```
9Er,5Vn8P$
```

Make sure that all criteria are met, and that these passwords are truly random.

(EXTRA) Explore how we could make the function automatically copy the password to the computer's clipboard (as if we had selected it and done ctrl+copy).

```
import random

import string


def generate_password(min_length, upper=True, lower=True, numbers=True,
special_character=False):

    upper_case = string.ascii_uppercase
```

```
lower_case = string.ascii_lowercase
digits = string.digits
special = string.punctuation

characters = ""
if upper:
    characters += upper_case
if lower:
    characters += lower_case
if numbers:
    characters += digits
if special_character:
    characters += special

if not characters:
    raise ValueError("You must select at least one character type.")

pwd = ""

# Generate a random password of desired length
while len(pwd) < min_length:
    new_char = random.choice(characters)
    pwd += new_char

return pwd
```

```
# ---- USER INPUT ----
min_length = int(input("Enter the minimum length: "))
has_upper = input("Do you want uppercase letters? (y/n): ").lower() == 'y'
has_lower = input("Do you want lowercase letters? (y/n): ").lower() == 'y'
has_number = input("Do you want numbers? (y/n): ").lower() == 'y'
has_special = input("Do you want special characters? (y/n): ").lower() == 'y'

pwd = generate_password(min_length, has_upper, has_lower, has_number, has_special)
print("Generated password:", pwd)
```

```
Enter the minimum length:  13
Do you want uppercase letters? (y/n):  y
Do you want lowercase letters? (y/n):  y
Do you want numbers? (y/n):  y
Do you want special characters? (y/n):  y
Generated password: 9Sm-]Y:m@'$qb
```

2. Simple data processing

A coworker has asked us for a favor, taking advantage of the fact that she knows we are learning to program. She has a history of Catalan football matches in a file, where she has stored the names of the teams and the results. She needs us to process the data automatically, to extract the results she needs.

Use the file "historic_partits.txt"

You need a program that returns:

The total number of goals scored by each team.

The name of the top-scoring team.

The name of the team with the most goals scored

The overall standings (each win: 3 pts, draw 1 pts, defeat 0 pts)

```
def process_match_results(filename):
    teams = {}

    with open(filename, "r", encoding="utf-8") as f:
        for line in f:
            line = line.strip()
            if not line:
                continue

        # Clean spacing
            parts = line.replace(" - ", "-").split("-")
            if len(parts) != 2:
                continue  # skip bad lines

            left, right = parts[0].strip(), parts[1].strip()

        # Extract team names and goals safely
            *team_a_parts, goals_a = left.split()
            goals_b, *team_b_parts = right.split()
```

```
    team_a = " ".join(team_a_parts)
    team_b = " ".join(team_b_parts)
    goals_a, goals_b = int(goals_a), int(goals_b)

# Initialize teams if not exist
    for team in (team_a, team_b):
        if team not in teams:
            teams[team] = {"W": 0, "D": 0, "L": 0, "GF": 0, "GA": 0, "Pts": 0}

# Update stats
    teams[team_a]["GF"] += goals_a
    teams[team_a]["GA"] += goals_b
    teams[team_b]["GF"] += goals_b
    teams[team_b]["GA"] += goals_a

    if goals_a > goals_b:
        teams[team_a]["W"] += 1
        teams[team_a]["Pts"] += 3
        teams[team_b]["L"] += 1
    elif goals_a < goals_b:
        teams[team_b]["W"] += 1
        teams[team_b]["Pts"] += 3
        teams[team_a]["L"] += 1
    else:
        teams[team_a]["D"] += 1
        teams[team_b]["D"] += 1
```

```
            teams[team_a]["Pts"] += 1
            teams[team_b]["Pts"] += 1

# Print header
    print(f"{'Team':<20} {'W':<3} {'D':<3} {'L':<3} {'GF':<4} {'GA':<4} {'Pts':<4}")
    print("-" * 45)

# Sort and print table
    for team, data in sorted(
        teams.items(),
        key=lambda x: (x[1]['Pts'], x[1]['GF'] - x[1]['GA'], x[1]['GF']),
        reverse=True
    ):
        print(f"{team:<20} {data['W']:<3} {data['D']:<3} {data['L']:<3} {data['GF']:<4} {data['GA']:<4}
{data['Pts']:<4}")

# Top scoring team
    max_goals = max(t["GF"] for t in teams.values())
    top_teams = [name for name, data in teams.items() if data["GF"] == max_goals]
    print("\nTop-scoring team(s):", ", ".join(top_teams), f"with {max_goals} goals")

filename = "C:/Users/sarav/Downloads/historic partits.txt"
process_match_results(filename)
Team                 W   D   L   GF   GA   Pts
---------------------------------------------
Girona FC            31  3   13  139  94   96
```

Llagostera	29	7	20	159	142	94
Sabadell	26	7	15	141	121	85
Cornellà	25	7	22	147	146	82
RCD Espanyol	23	11	21	131	144	80
Figueres	23	10	23	161	154	79
Lleida Esportiu	23	6	23	129	133	75
Terrassa	20	14	23	147	164	74
FC Barcelona	22	7	15	125	115	73
Vilafranca	20	12	25	157	172	72
Badalona	18	14	16	134	124	68
Reus Deportiu	20	8	16	119	111	68
Nàstic de Tarragona	20	8	27	130	148	68
Granollers	21	4	24	122	117	67
Olot	18	10	26	132	144	64
Sant Andreu	17	11	25	123	134	62
Manlleu	16	9	18	106	118	57
Prat	16	8	21	106	111	56
Cerdanyola	17	5	28	117	133	56
Europa	12	5	16	93	93	41

Top-scoring team(s): Figueres with 161 goals