Double-click (or enter) to edit

Double-click (or enter) to edit

Guidelines

1. About the Project:

The Introduction to Computer Programming project is designed to evaluate your understanding and practical application of fundamental programming concepts using Python. This assessment encompasses various topics, including data types, indexing, slicing, operators, in-built functions, statements, conditionals, loops, object-oriented programming, and exception handling. In addition, you will showcase your ability to create custom functions and tackle advanced looping concepts.

2. Skills Required:

To successfully complete this project assessment, you should possess the following skills and knowledge:

- Proficiency in Python programming language
- Understanding of data types, indexing, and slicing in Python
- Familiarity with operators, in-built functions, and methods
- Ability to work with statements, indentation, and conditionals
- Competence in implementing loops and iterations, including conditional and infinite looping
- Mastery of creating and utilizing custom functions in Python
- Knowledge of advanced looping concepts
- Understanding of object-oriented programming (OOPs) principles in Python
- Ability to handle exceptions in your code

3. Deliverables:

You are required to submit the following items for evaluation:

- Completed Python coding solutions for the given questions (coding assessment)
- Collab link containing your Python code for each question
- Video explanation for any five coding questions (demonstrating your understanding and approach)

4. Rubrics:

Your project assessment will be evaluated based on the following rubrics:

a. Coding Assessment:

- Correctness of the code solution
- Efficient use of coding constructs
- Clear and well-structured code organization

b. Collab Link:

- Submission of Python code through the provided Collab link
- Proper organization of code files and resources

c. Video Explanation:

- Clear articulation of solution approach
- Coverage of important concepts and logic
- Coherent and well-structured explanation

Please use these pointers as the basis for evaluating each section of the project assessment.

CodeStorm Challenge: Solving and Explaining Complex Coding Problems

Problem Statement:

The CodeStorm Challenge is an intense coding competition designed to test participants' programming skills, problem-solving abilities, and communication prowess. In this challenge, participants will be required to solve a total of 30 intricate coding questions that cover a wide range of algorithms, data structures, and programming paradigms. Additionally, participants are tasked with creating an explanatory video that delves into the solutions of any five chosen questions.

The challenge aims to evaluate participants on their coding efficiency, algorithmic thinking, and presentation skills. Participants will need to not only implement correct and optimized solutions to the coding problems but also effectively communicate their thought processes and strategies in the form of a video presentation.

Key Objectives:

- Coding Proficiency: Participants must demonstrate their ability to write clean, efficient, and bug-free code to solve complex programming challenges within a

time-constrained environment.

- Problem Solving: The challenge will encompass a variety of problem domains, including dynamic programming, graph algorithms, string manipulation, and more. Participants should showcase their prowess in breaking down problems, designing algorithms, and translating solutions into code.
- Communication Skills: In addition to coding skills, participants need to exhibit strong communication skills by creating a comprehensive video explaining the thought process, approach, and solution to five selected problems. Clarity, conciseness, and coherence in the video are essential.
- Time Management: Participants will need to efficiently manage their time throughout the challenge, allocating sufficient time to solve each question and ensuring they have ample time to create high-quality explanatory videos.
- Adaptability: As the challenge encompasses a diverse set of coding problems, participants must be adaptable in applying different algorithms and techniques to solve problems that may vary in difficulty and complexity.

Note:

1. Solve 30 coding problems out of the different problem statements given here.
2. Create a video explaining any 5 questions and include the video link before submitting.

Introduction to Python Programming

Score Board Operator

An operator working behind the scoreboard of a inter cohort AlmaBetter cricket tournament, is responsible for updating the scores and points of each team. However, the operator is currently facing a challenge. He has been tasked with updating the total number of points gained by Team London, but he does not possess the necessary programming skills to complete this task. According to the tournament's rules, teams are awarded the following points based on the outcome of a match:

- wins: 3 points
- draws: 1 point
- losses: 0 points

Team London has played **8 matches** in this tournament. They won **4 matches**, lost **3 matches** and **drew 1**. The operator is in need of assistance to calculate the total number of points earned by Team London. As a python expert adept with knowledge of integer, floats and boolean, you can help the operator by writing a solution for the following problem.

What would your approach be?


```
#1. Define your fixed values:
wins = 4
draws = 1
losses = 3
```

```
#2. Calculate the total points gained by Team London
win_point = wins * 3
draws_point = draws * 1
loss_point = losses * 0

total_point = win_point + draws_point + loss_point
```

```
# 3. Print the variable london_points
print("London Total Points :", total_point)
```

```
London Total Points : 13
```

Nutrition bar ingredient chart

Imagine you are a data analyst at a nutrition bar manufacturing company. Your department head approaches you with a question. The company produces a nutrition bar that contains **45g of raisins, 65g of almonds, and 30g of apricots**. The head of the manufacturing department wants you to create an ingredient percentage list for the nutrition bar using python.

Equipped with the knowledge of int, float and booleans in python, how would you approach this situation?

```
# 4. Calculate the percentage of raisins and print the variable
raisins= 45
almonds= 65
apricots=30

total_weight = raisins + almonds + apricots
```

```
raisins_percentage = (raisins / total_weight) * 100
print(f"Percentage of Raisins :{raisins_percentage:.2f}%")
```

```
Percentage of Raisins :32.14%
```

```
# 5. Calculate the percentage of almonds and print the variable

almonds_percentage = (almonds / total_weight) * 100
print(f"Percentage of Almonds :{almonds_percentage:.2f}%")
```

```
Percentage of Almonds :46.43%
```

```
# 6. Calculate the percentage of apricots and print the variable
apricots_percentage = (apricots / total_weight) * 100
print(f"Percentage of Apricots :{apricots_percentage:.2f}%")
```

```
Percentage of Apricots :21.43%
```

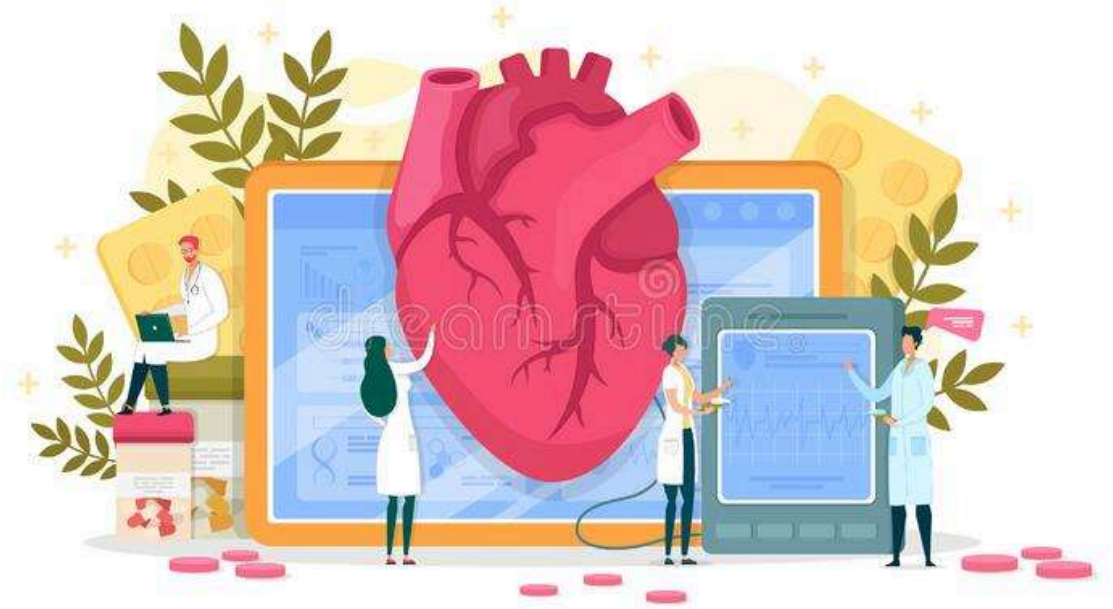
Measuring the Accuracy of a Disease Detection Model- Optional

If you're not impressed with these usecases yet, understand that you're just getting started with python. Going forward as a data scientist, you'll be using what you've learned to design machine learning models that just might even save lives!!

Let's say you have created a machine learning model to detect diseases in patients based on their symptoms and medical history. The model has been tested on a dataset of **100 patients**, including **30 with diabetes**, **45 with heart disease**, and **25 with cancer**. The model's performance was evaluated in three rounds. In the first round, the model correctly **detected diabetes in 25 out of the 30 patients** with the disease. In the second round, the model correctly detected **heart disease in 35 out of the 45 patients** with the disease. In the final round, the model correctly **detected cancer in 20 out of the 25 patients** with the disease.

The model's accuracy is defined as the percentage of correctly detected cases out of the total number of patient

Assess how accurate your model is across these 3 diseases.



```
# Store the number of heart disease patients
heart_patient = 30

# Store the number of diabetes patients
diabetes_patient = 45

# Store the number of cancer patients
cancer_patient = 25

# Store the number of your correct guesses of heart disease patients
correct_heart_patient = 25

# Store the number of your correct guesses of diabetes patients
correct_diabetes_patient = 35

# Store the number of your correct guesses of cancer patients
correct_cancer_patient = 20
```

Calculate your accuracy in each of the three cases

```
# 9. Print the heart disease detection accuracy
accuracy_heart_detection = (correct_heart_patient / heart_patient) * 100
print(f"Accuracy for heart disease : {accuracy_heart_detection:.2f}%")
```

```
Accuracy for heart disease : 83.33%
```

```
# 10. Print the diabetes detection accuracy

accuracy_diabetes_detection = (correct_diabetes_patient / diabetes_patient) * 1
print(f"Accuracy for diabetes disease : {accuracy_diabetes_detection:.2f}%")
```

```
Accuracy for diabetes disease : 77.78%
```

```
# 11. Print the cancer detection accuracy

accuracy_cancer_detection = (correct_cancer_patient / cancer_patient) * 100
print(f"Accuracy for cancer disease : {accuracy_cancer_detection:.2f}%")
```

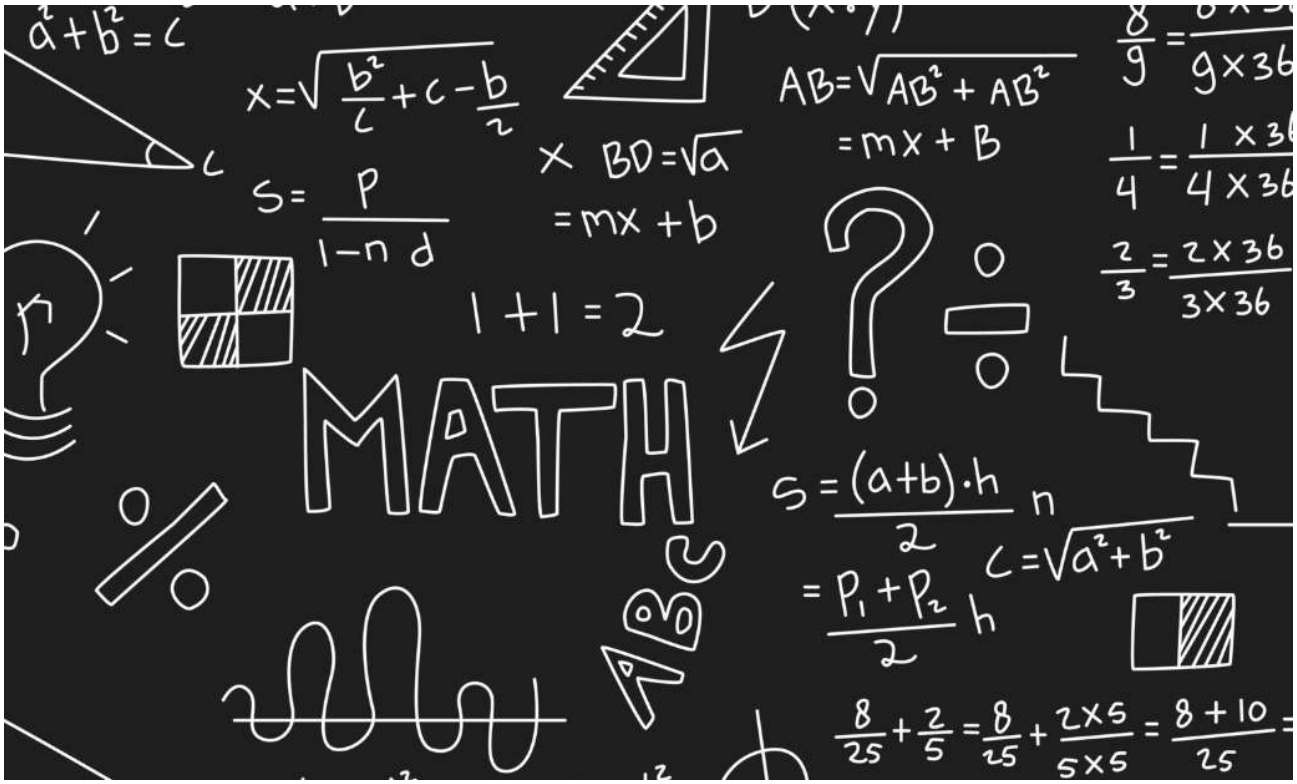
```
Accuracy for cancer disease : 80.00%
```

```
# 12. Print the overall accuracy of your model # Average of accuracy in 3 rounc
total_patient = heart_patient + diabetes_patient + cancer_patient
total_correct_patient = correct_heart_patient + correct_diabetes_patient + corr

accuracy_overall = (total_correct_patient / total_patient) * 100
print(f"Overall model accuracy: {accuracy_overall:.2f}%")
```

```
Overall model accuracy: 80.00%
```

Data Types in Python

Solving Algebra the Cool Way 😎



You must remember the algebraic equations and functions you came across during your school days. If you do, you also must remember how tedious it was to do solve them manually. 😫 Consider a quadratic function:

f(x)=x^2+3x-4

Find the value of f(x) at x =3 , x=-2, and x=4

Doing this manually like the old days would be a tedious job 😔. But now you have python at your disposal 😳 ! How would you approach this problem using what you have learned?

New Section

```
# 13. Define Method
def quadratic_function(x):
  return x**2 + 3*x - 4
```

```
# 14. Calculate the value of the function f(x) at x = 3
func_evaluated_at_3 = quadratic_function(3)
print(f"f(3) = {func_evaluated_at_3}")
```

```
f(3) = 14
```

```
# 15. Calculate the value of the function f(x) at x = -2
func_evaluated_at_minus2 = quadratic_function(-2)
print(f"f(-2) = {func_evaluated_at_minus2}")
```

```
f(-2) = -6
```

```
# 16. Calculate the value of the function f(x) at x = 4
func_evaluated_at_4 = quadratic_function(4)
print(f"f(4) = {func_evaluated_at_4}")
```

```
f(4) = 24
```

```
# 17. Check if func_evaluated_at_3 >= func_evaluated_at_minus2
def f(x):
  return x**2 + 3*x - 4

print([f"f({x}) is zero? {f(x) == 0}" for x in [3, -2, 4]])
```

```
['f(3) is zero? False', 'f(-2) is zero? False', 'f(4) is zero? False']
```

```
# 18. Check if value of f(x) at 3,-2 or 4 is equal to 0 or not
x_values = [3, -2, 4]

for x in x_values:
    f_x = quadratic_function(x)
    print(f"f({x}) = {f_x}. Is zero? {f_x == 0}")
```

```
f(3) = 14. Is zero? False
f(-2) = -6. Is zero? False
f(4) = 24. Is zero? False
```

Double-click (or enter) to edit

Finding total E's in the Bookshelves

In your pursuit of expanding your knowledge of Python, you come across a bookstore in search of books on the subject.

There, you meet a salesperson who is also a Python enthusiast!!

The salesperson presents you with a challenge - they have a list of five books related to Python and if you can write a program that finds the count of letter 'e', the store will give you a free book of your choice. 😀

Are you ready to accept the challenge? 💪


```
# 19.  Here's the list of books
books = "Python for Everyone, Learn Python the Hard Way, Python Crash Course ,
```

```
# 20. Store the count of books containing E/e in the title. Note this will be a
e_counts = books.count('e')

E_counts = books.count('E')

total_e_counts = e_counts + E_counts
```

```
#  21. Print the value
print(f"The total number of 'e's in the list of books is {total_e_counts}")
```

```
The total number of 'e's in the list of books is 8
```

Unleashing the Power of Mean in Exam Analysis

When analyzing exam scores, teachers have the power to unlock valuable insights into their students' performance. By using the mean and median, they can get a clear picture of how the class is doing as a whole and identify areas where improvement is needed.

The mean score is like a report card for the entire class, showing the average of all exam scores. It gives a good general idea of how the class is doing but can be influenced by a few students with extremely high or low scores.

By analysing the mean, teachers can get a complete picture of the class performance and make informed decisions to improve the learning experience for their students. So go ahead and use these powerful tools to take your students' exam scores to the next level! 💪

```
# 22. List of student marks

marks = [67,87,75,46,89,97,68,98,87,88]
```

```
# 23. Calculate the mean of the marks. Use sum() and len() functions
sum_marks = sum(marks)

count_marks = len(marks)

mean_marks = sum_marks / count_marks
```

```
# 24. Print the mean of the marks.
print(mean_marks)
```

```
80.2
```

John's Fruit Stand Sales Mix-up

You must remember from the previous lessons that dictionaries can be used to store values corresponding to a particular key. But how is this helpful to us?

Let's look at this scenario:

Jim owns a quaint little fruit stand that sells mangoes, bananas, apples, and pineapples. One day, with Jim being tied up, he tasked his son John to manage the shop for a day. Being new to the game, John recorded the sales data in an unconventional way. He wrote down the name of each fruit as many times as it was sold, creating a list like this:

['Mango','Mango','Mango','Pineapple','Pineapple','Pineapple','Apple','Mango','Banana','Apple','Banana','Apple','Pineapple','Apple','Apple','Pineapple','Pineapple'] 🤦

But, this method is not very efficient or easy to read.

Forunately you were there at the shop to grab some apples yourselves, and you saw what John was doing.

You can help him out to structure the data by using dictionaries. How?

```
# 25. Here's John's list:
fruit_list=['Mango','Mango','Mango','Pineapple','Pineapple','Pineapple','Apple'
```

```
# 26. Step 2: Create an empty dictionary which will contain the unique fruit na
fruit_dict={}

# Set the values for each key separately

# First Key
fruit_dict['Mango'] = 0

# Second Key
fruit_dict['Pineapple'] = 0

# Third Key
fruit_dict['Apple'] = 0

# Fourth Key
fruit_dict['Banana'] = 0

for fruit in fruit_list:
  fruit_dict[fruit] += 1
```

```
# 27. fruit_dict.

print(fruit_dict)
```

```
{'Mango': 4, 'Pineapple': 6, 'Apple': 5, 'Banana': 2}
```

Indexing & Slicing

Imagine that you are a superhero movie buff and you're trying to create a list of the most iconic Marvel characters. You have compiled a list of some of the most popular characters, but you want to narrow it down to the top 6.

Here's your list of characters:

marvel_words = ['Avengers', 'X-Men', 'Spider-Man', 'Iron Man', 'Hulk', 'Thor', 'Black Widow', 'Captain America', 'Wolverine', 'Doctor Strange', 'Namor']

```
# 28. Here's your list of characters:

marvel_words = ['Avengers', 'X-Men', 'Spider-Man', 'Iron Man', 'Hulk', 'Thor',
```

You want to narrow down the list to the last 6 characters in the list. These characters are:

- Namor
- Doctor Strange
- Wolverine
- Captain America
- Black Widow
- Thor

Write a code to slice these characters from the list

```
# 29. Replace x with appropriate values
new_marvel=marvel_words[5:]
print(new_marvel)
```

```
['Thor', 'Black Widow', 'Captain America', 'Wolverine', 'Doctor Strange', 'Namor
```

Create two teams of superheroes based on their index position: one team should consist of all the characters located at even index positions, while the other team should consist of all the characters located at odd index positions.

```
# 30. Add appropriate index and step size
team1 = marvel_words[1::2]
print(team1)
team2 = marvel_words[::2]
print(team2)
```

```
['X-Men', 'Iron Man', 'Thor', 'Captain America', 'Doctor Strange']
['Avengers', 'Spider-Man', 'Hulk', 'Black Widow', 'Wolverine', 'Namor']
```

3. Store the word 'Spider-Man' in a new variable, and slice the word 'Spider' from it.

```

```

```
#  31. Add appropriate index number
spider_man = marvel_words[2]
spidey_sliced = spider_man[0:6]
print(spidey_sliced)
```

```
Spider
```

4. Store the word 'Iron-Man' in a variable and use reverse indexing to reverse it.

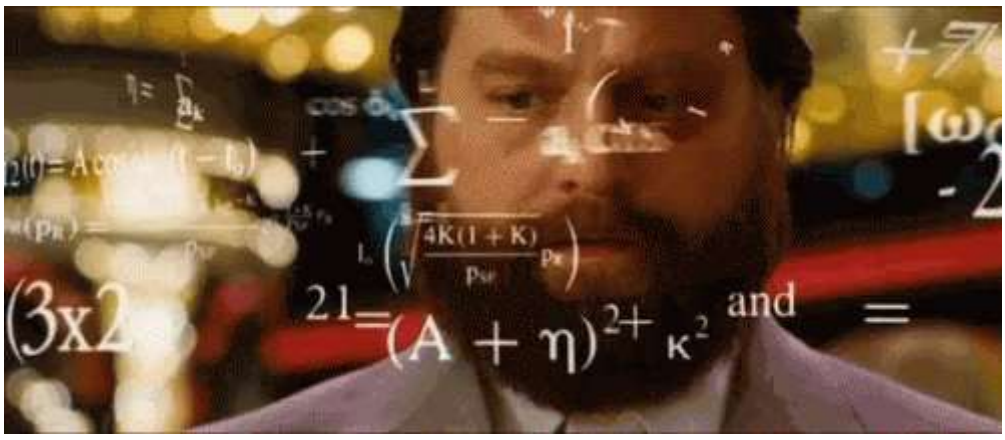
```
# 32. Your code here
iron_man = marvel_words[3]
reversed_man = iron_man[::-1]
print(reversed_man)
```

```
naM norI
```

Operators in Data Types

🎉 Let's do some math!

- 📐 First up, let's calculate the area of a square with sides that measure 5 cm. 🤔 How much space does this shape occupy?
- 🔍 Next, let's find the volume of a cube with sides that measure 12 cm. 📏 How much room does this shape take up?
- 🔢 Lastly, we'll find the quotient and remainder when we divide 10 by 4. 🧐 How many times does 4 fit into 10? What's left over?



```
# My code
# Calculations for the math problems
side_square = 5
ar_square = side_square * side_square         # Area of a square = side * side

side_cube = 12
vol_cube = side_cube * side_cube * side_cube  # Volume of a cube = side * side

dividend = 10
divisor = 4
quotient = dividend // divisor                 # Integer division (//) gives th
remainder = dividend % divisor                 # Modulo operator (%) gives the


# Print the results in the requested order
print(remainder)
print(quotient)
print(vol_cube)
print(ar_square)
```

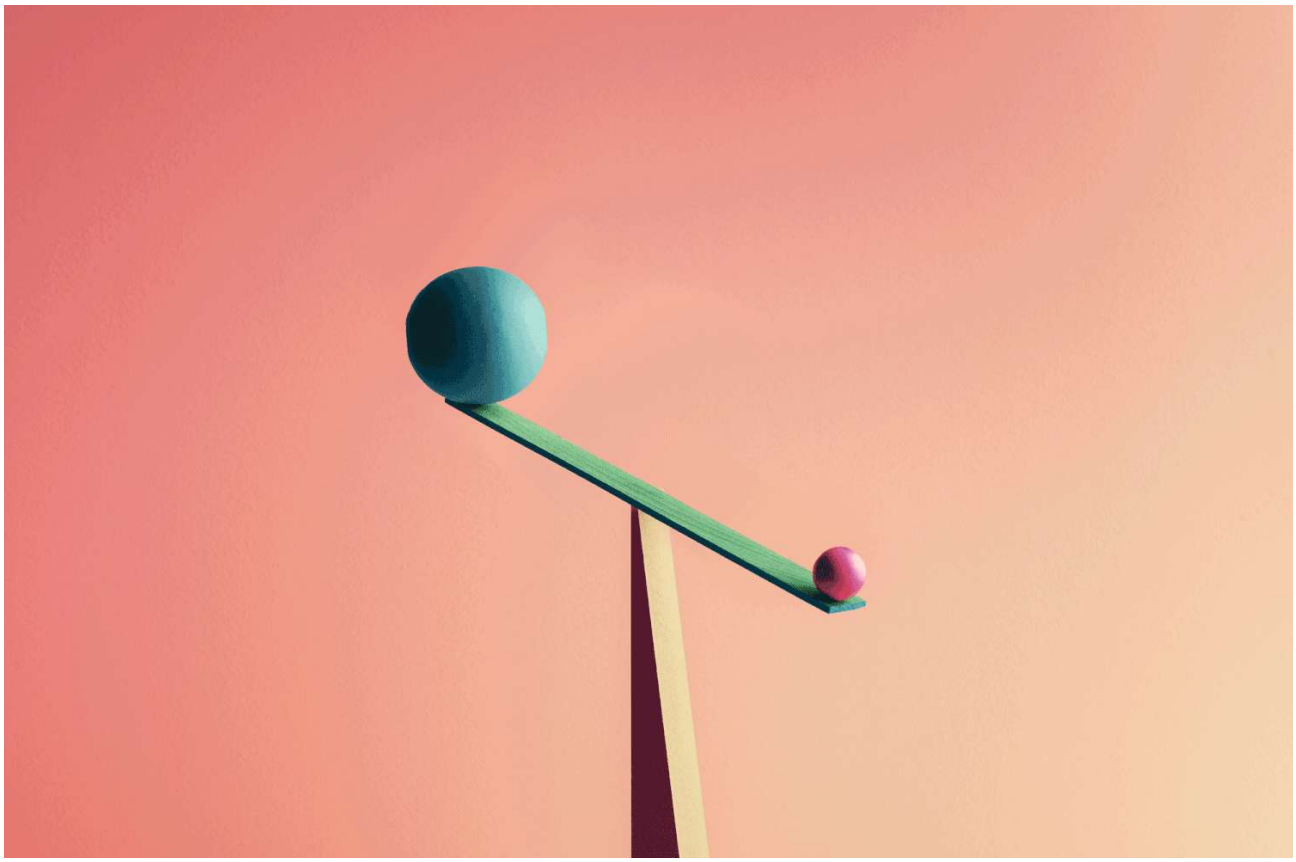
```
2
2
1728
25
```

📚 Let's practice some comparison operators! Here's a problem for you to solve:

🍎 The first basket contains 12 apples, and the second basket contains 8 apples. Use comparison operators to answer the following questions:

- Is the number of apples in the first basket greater than the number of apples in the second basket?
- Is the number of apples in the second basket less than or equal to the number of apples in the first basket?
- Is the number of apples in the first basket equal to the number of apples in the second basket?
- Is the number of apples in the second basket not equal to the number of apples in the first basket?

Use comparison operators (i.e. >, >=, <, <=, ==, !=) in your code to answer each question. Good luck! 🍎🍏



```
# My code here
# Initialize variables based on the problem description
basket1 = 12
basket2 = 8

# Question 1: Is the number of apples in the first basket greater than the numb
q1_result = basket1 > basket2
print(f"Is basket1 > basket2? {q1_result}")

# Question 2: Is the number of apples in the second basket less than or equal t
q2_result = basket2 <= basket1
print(f"Is basket2 <= basket1? {q2_result}")

# Question 3: Is the number of apples in the first basket equal to the number c
q3_result = basket1 == basket2
print(f"Is basket1 == basket2? {q3_result}")

# Question 4: Is the number of apples in the second basket not equal to the num
q4_result = basket2 != basket1
print(f"Is basket2 != basket1? {q4_result}")
```

```
Is basket1 > basket2? True
Is basket2 <= basket1? True
Is basket1 == basket2? False
Is basket2 != basket1? True
```

🌴 Are you ready for your trip to Goa? 🌊 You've got two bags with you, and you need to make sure they're light enough to board the flight. Let's use some logical operators to find out if you can make it on time!

🧳 One of your bags weighs 15 kg, while the other one weighs only 5 kg. 💪 You know you can pack a lot more in the heavier bag, but you have to be careful not to go over the weight limit.

👮 The airline has a strict policy that both of your bags must weigh less than 13 kg each. Can you make it on the flight? Let's use logical operators to find out!


```
# Define the weight of your bags
bag1_weight = 15
bag2_weight = 5

# Check if both bags weigh less than 13 kg
limit = 13

# Check if both conditions are True using 'and'
can_board = (bag1_weight < limit) and (bag2_weight < limit)

print(f"Can board: {can_board}") # Output will be False
```

```
Can board: False
```

```
print("Sorry, You can not go through!")
```

```
Sorry, You can not go through!
```

OR

```
print("Enjoy your trip!")
```

🧳 Uh oh, it looks like we have a problem with our bags! But don't worry, the crew has offered a solution. 😊

👮‍♀️ They've said that if at least one of our bags weighs less than 6 kg, we can still board the flight this time. 🛬

🤔 Are we ready for the trip or not? Let's use some logical operators to find out!

```
# Define the weight of your bags
bag1_weight = 15
bag2_weight = 5

# Define the new conditional limit
conditional_limit = 6

# Check if at least one bag weighs less than 6 kg
# We use the 'or' operator: the result is True if EITHER condition is True.
can_board_flight = (bag1_weight < conditional_limit) or (bag2_weight < conditic

print(f"Bag 1 weight: {bag1_weight} kg")
print(f"Bag 2 weight: {bag2_weight} kg")
print(f"Conditional boarding limit: {conditional_limit} kg")
print("-" * 35)

# Conditional printing based on the logical result
if can_board_flight:
    print("Enjoy your trip! 🌴✈️")
else:
    print("Sorry, You can not go through! 😔")
```

```
Bag 1 weight: 15 kg
Bag 2 weight: 5 kg
Conditional boarding limit: 6 kg
-----------------------------------
```

```
Enjoy your trip! 🌴✈️
```

```
print("Sorry, You can not go through!")
```

```
Sorry, You can not go through!
```

OR

```
print("Enjoy your trip!")
```

🤔 Did you notice how we made a decision on which print statement to use based on the result of our logical operators? 🤔

🚀 Later on, we're going to learn how to do this in a single step, making it even easier for you. So stay curious 😎

📚 Imagine that you're a book lover with a collection of lot of books. 🤓

🔎 You need to find out how many of your books contain the word "Python" in them. 🐍

```
books = [
    "Python for Data Science Handbook by Jake VanderPlas",
    "The Pragmatic Programmer by Andrew Hunt and David Thomas",
    "Python Machine Learning by Sebastian Raschka",
    "The Alchemist by Paulo Coelho",
]
```

```
# Initialize a counter variable
python_book_count = 0

# Check each book title using the "in" operator and update the count
# Check books[0]
if "Python" in books[0]:
    python_book_count += 1

# Check books[1]
if "Python" in books[1]:
    python_book_count += 1

# Check books[2]
if "Python" in books[2]:
    python_book_count += 1
```

```
# Check books[3]
if "Python" in books[3]:
    python_book_count += 1

print(f"Total number of books containing 'Python': **{python_book_count}**")
```

```
Total number of books containing 'Python': **2**
```

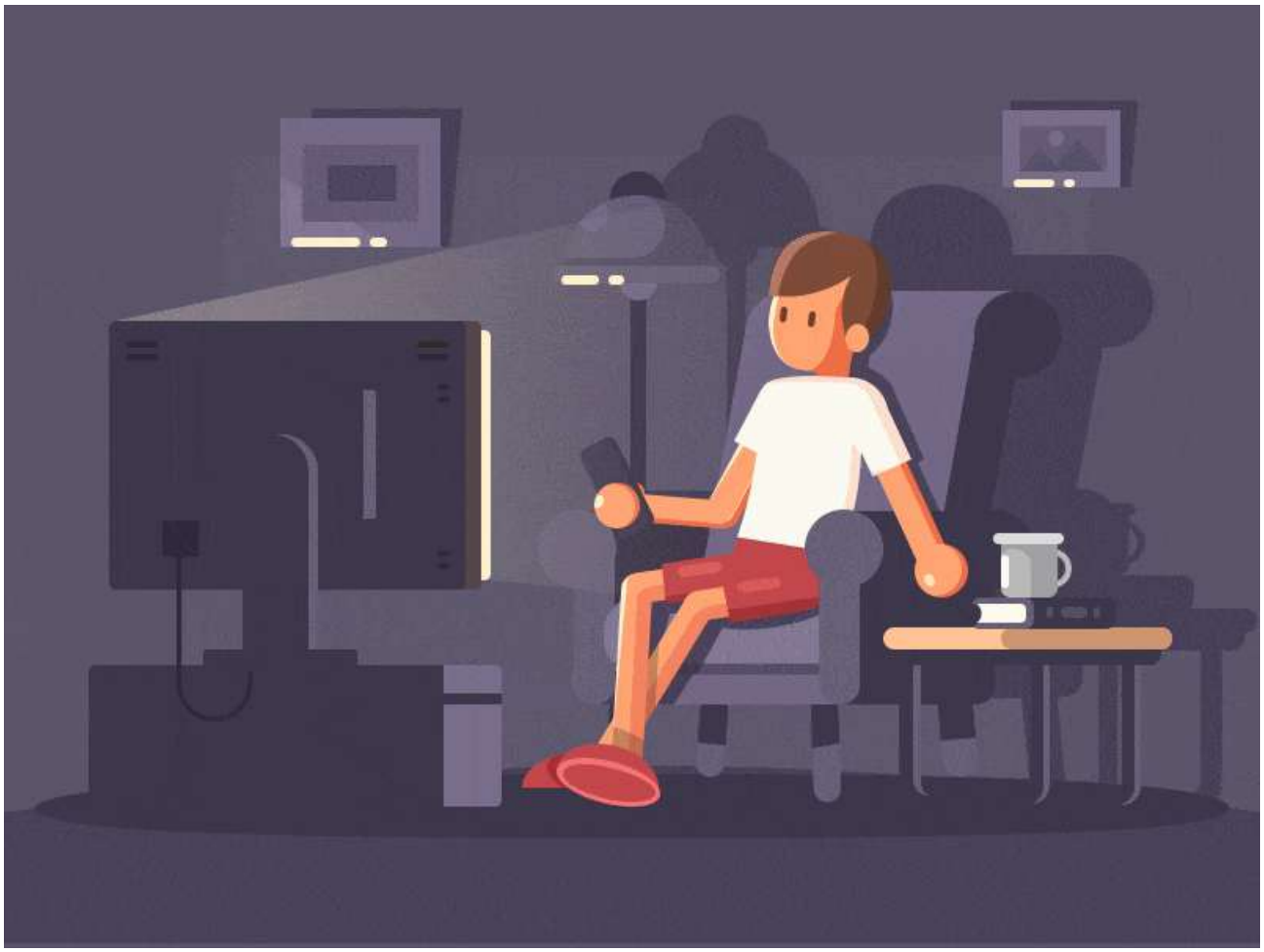
```
count=0     # run this only once
```

```
count=count+1
print(count)
```

```
1
```

In-Built Functions & Methods

Here is a string for you to work with:



```
# Here is the statement.

statement = "anas eagerly recommended 'the shawshank redemption' to his friends
```

```
print(statement)
```

```
anas eagerly recommended 'the shawshank redemption' to his friends. He described
```

1. Create a list containing the three sentences in the statement. Hint: Use split() method.

```
# Create a list containing the three sentences in the statement.

sentence_list= statement.split('.')
print(sentence_list)
```

```
["anas eagerly recommended 'the shawshank redemption' to his friends", ' He desc
```

2. The name 'anas' is spelled wrong. Replace it with 'Anas'.

```
# The name 'anas' is spelled wrong.
# Replace it with 'Anas'.

statement=  statement.replace('anas', 'Anas')
print(statement)
```

```
Anas eagerly recommended 'the shawshank redemption' to his friends. He described
```

3. The title of the book is not written in the correct cases as well. Make corrections there.

```
# Make corrections here.
statement = statement.replace('the shawshank redemption', 'The Shawshank Redemp
print(statement)
```

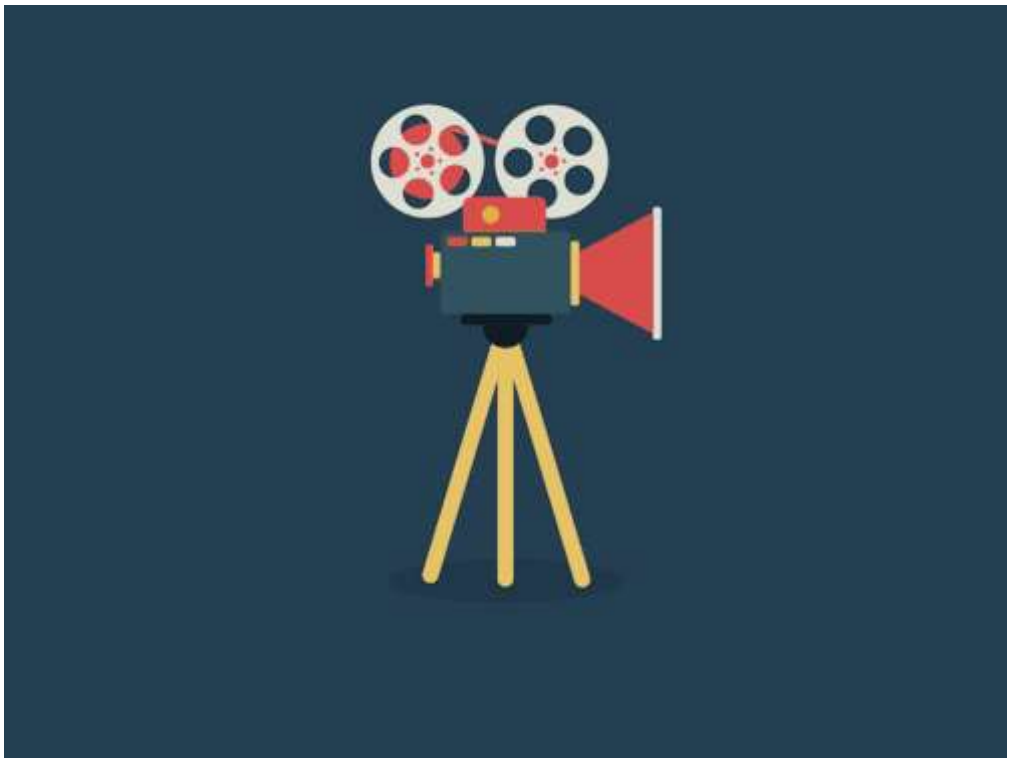
```
Anas eagerly recommended 'The Shawshank Redemption' to his friends. He described
```

4. You believe "The Shawshank Redemption" would be bit too gloomy for the lot. Take user input of which movie to recommend and replace "The Shawshank Redemption" from the statement with the new recommendation.

```
# Change the recommendation of movie to one, of your choice.
# Take input of movie

new_movie = input("Which movie wold you recommend? ")

# The replace() method is used to swap the old title with the new one.
statement = statement.replace('The Shawshank Redemption', new_movie)
print(statement)
```

```
Which movie wold you recommend? ramayan
Anas eagerly recommended 'ramayan' to his friends. He described the movie as a c
```

📽️ Suppose you have a collection of your favorite movies 🍿🎞️ and you want to keep track of them using a Python list 📜. You start with an empty list [] and keep adding new movies 🆕 to it. You also want to be able to remove movies ❌ that you have watched or no longer wish to keep in your collection. Finally, you want to be able to count 🔢 how many movies you have in your collection and reverse the order of the list 🔄.



```
# Start with an empty list
movie_collection = []

# Add some movies to the collection using the append() method
movie_collection.append("Pulp Fiction") # Use your favourite movie names
movie_collection.append("Inception")
movie_collection.append("Parasite")
movie_collection.append("Spirited Away")
movie_collection.append("Mad Max: Fury Road")

# Print the current collection of movies
print("My Movie Collection:", movie_collection)

# Remove a movie from the collection using the pop() method
# Code to remove index 3 here (Removes "Spirited Away")
movie_collection.pop(3)
print("After removing movie 4 (at index 3):", movie_collection)

# Count how many movies are in the collection using the len() function
# Note: len() counts the total number of items, which is the intended goal.
num_movies = len(movie_collection)
print("Number of movies:", num_movies)

# Reverse the order of the list using the reverse() method
movie_collection.reverse()

print("Reversed movie collection:", movie_collection)
```

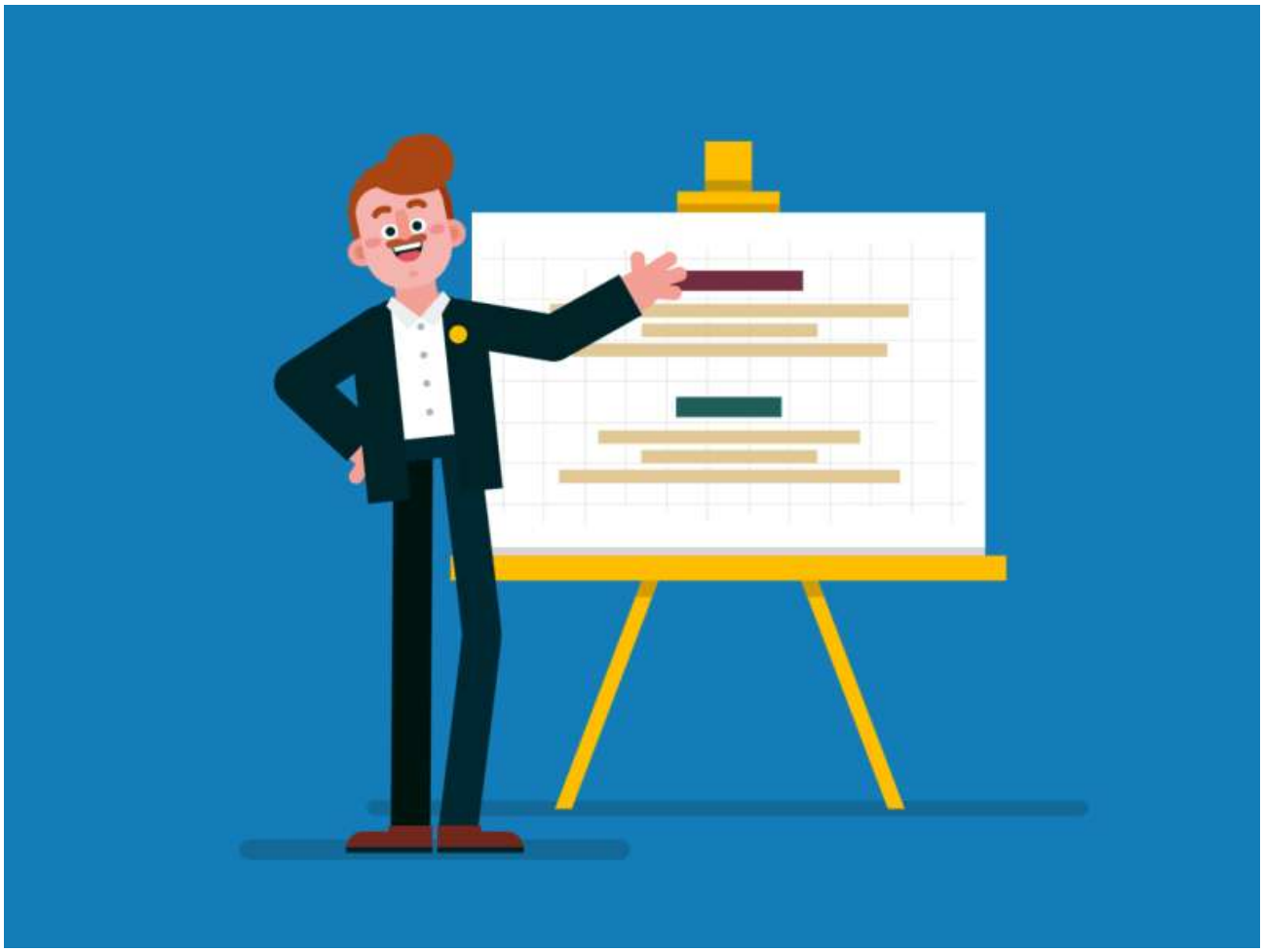
```
My Movie Collection: ['Pulp Fiction', 'Inception', 'Parasite', 'Spirited Away',
After removing movie 4 (at index 3): ['Pulp Fiction', 'Inception', 'Parasite', '
Number of movies: 4
Reversed movie collection: ['Mad Max: Fury Road', 'Parasite', 'Inception', 'Pulp
```

👩‍🏫 A teacher has stored the names of his students in a tuple as follows:

('Adam','Alice','Ben','Bilal','Bharath')

👨‍🎓 Their role numbers are defined as their index number + 1.

🖥️ Write a program which would take the name of the student as input and return their role number.



```
student_tupple=('Adam','Alice','Ben','Bilal','Bharath')
```

```
# Take input from user
student = input("Input the student name to get roll number - ")
```

```
    try:
        # 1. Use the index() method to find the position (index) of the student's r
        student_index = student_tupple.index(student)

        # 2. Calculate the roll number (index + 1)
        roll_number = student_index + 1

        # 3. Print the result
        print(student, "'s roll number is: ", roll_number)

    except ValueError:
        # Handle the case where the entered name is not in the tuple
        print(f"Error: The student '{student}' was not found in the class list.")
```

```
Input the student name to get roll number - Ben
Ben 's roll number is:  3
```

A store 🏪 sells different types of fruits 🍎🍌🍇🍓 in baskets 🧺. The storekeeper 👴 keeps track of the availability of each fruit type in separate sets. The storekeeper wants to know which fruits are available in both baskets, which fruits are unique to each basket, and the total number of fruits available. 🤔


```
# Define sets for fruits in each basket
basket1 = {'apple', 'banana', 'grape', 'orange'}
basket2 = {'banana', 'mango', 'pineapple', 'orange'}

# Find the total number of fruits available
all_unique_fruits = basket1 | basket2
total_fruits = len(all_unique_fruits)
print("Total number of fruits available:", total_fruits)

# Find fruits available in both baskets
both_baskets = basket1 | basket2   # Replace x with appropriate method
print("All the available fruits:", both_baskets)

# Find only the common fruits in both baskets
common_fruits = basket1 & basket2   # Replace x with appropriate method
print("Common fruits in both baskets:", common_fruits)

# Find fruits unique to each basket
unique_basket1 = basket1 - basket2    # Replace x with appropriate method
unique_basket2 = basket2 - basket1    # Replace x with appropriate method
print("Fruits unique to basket 1:", unique_basket1)
print("Fruits unique to basket 2:", unique_basket2)
```

```
Total number of fruits available: 6
All the available fruits: {'mango', 'grape', 'pineapple', 'orange', 'apple', 'ba
Common fruits in both baskets: {'orange', 'banana'}
Fruits unique to basket 1: {'grape', 'apple'}
Fruits unique to basket 2: {'pineapple', 'mango'}
```

Statements, Indentation & Conditionals

👉 Imagine that you are a food inspector who is responsible for inspecting the lead content of a packaged food material at a factory. If the content exceeds 5%, you should print the message "This batch cannot be approved".

```
# Get the lead content of the packaged food material
lead_content = float(input("Please enter the lead content of the packaged food

# Check if the lead content exceeds 5%
```

```
Please enter the lead content of the packaged food material in percentage: 3
```

```
if lead_content > 5:
  print("This batch cannot be approved")
```

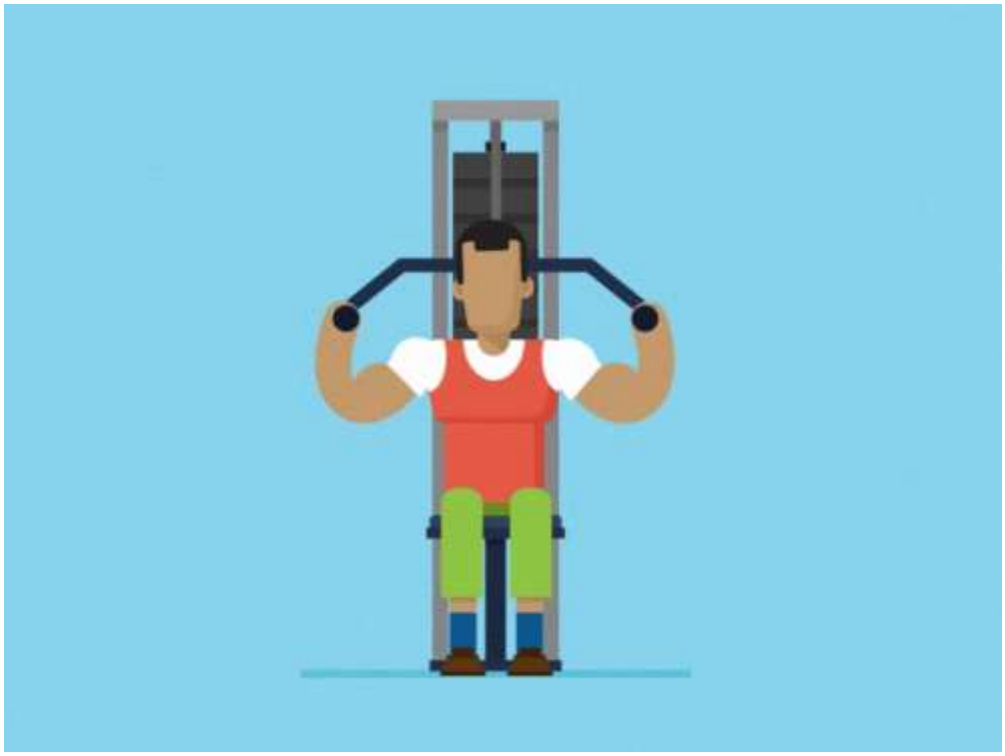
```
This batch cannot be approved
```

👉 Imagine: You're a bouncer at a nightclub and need to check if someone is old enough to enter. The legal age for entry is 18 years. If the person is less than 18 years print "Sorry You can not enter!".


```
age = int(input("enter the age"))
if age >= 18:
    print("you are eligible to vote")
# Check the age of the guest
if age < 18:
    print("Sorry You can not enter!")
else:
    print("Welcome! Enjoy your evening. 🎉") # Added an 'else' case for comple
```

```
enter the age12
Sorry You can not enter!
```

👉 Imagine you are a fitness instructor who needs to evaluate if a client has reached their weight loss goal. The target weight loss goal is 10 pounds. If the client has lost at least 10 pounds, you should print the message "Congratulations, you have reached your weight loss goal!" However, if the client has lost less than 10 pounds, you should print "Sorry, you have not reached your weight loss goal yet. Keep up the good work!"



```
# Get the client's weight loss
client_weight_loss = float(input("Please enter the client's weight loss (in pou

# Check if the client reached their weight loss goal
```

```
Please enter the client's weight loss (in pounds): 21
```

👉 Imagine: You're a gardener and need to check if a plant needs watering. If the soil moisture level is below 40%, print "Water the plant!", otherwise, print "The plant doesn't need watering."

```
# Take input of moisture level
moisture_level = int(input("Enter moisture level "))

#Check if moisture level is satisfactory or not
```

```
Enter moisture level 12
```

📦 📬 🚚 Creating a shipping cost calculator! ⚙️

You need to create a program that calculates the cost of shipping a package based on its weight and destination 📦 💰

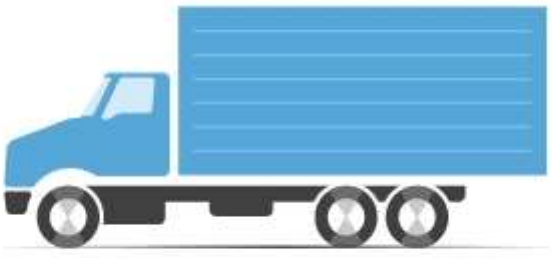
🌎 Domestic shipping within the country :

- Packages weighing up to 1 kg cost Rs.50 to ship 💸 Each additional kg costs Rs.10 extra 💰

🌎 International shipping to other countries 🌎:

- Packages weighing up to 1 kg cost Rs.500 to ship 💸

Each additional kg costs Rs.100 extra 💰 Ready to ship? 🚛 🚀



🚙 As a car dealership manager, you need to create a discount system for your customers based on the price of the car they want to buy. Here are the rules! 📝

🚘 If the car price is less than Rs. 10,00,000, there is no discount.

🎁 If the car price is between Rs. 10,00,000 and Rs. 20,00,000, there is a discount of 5%.

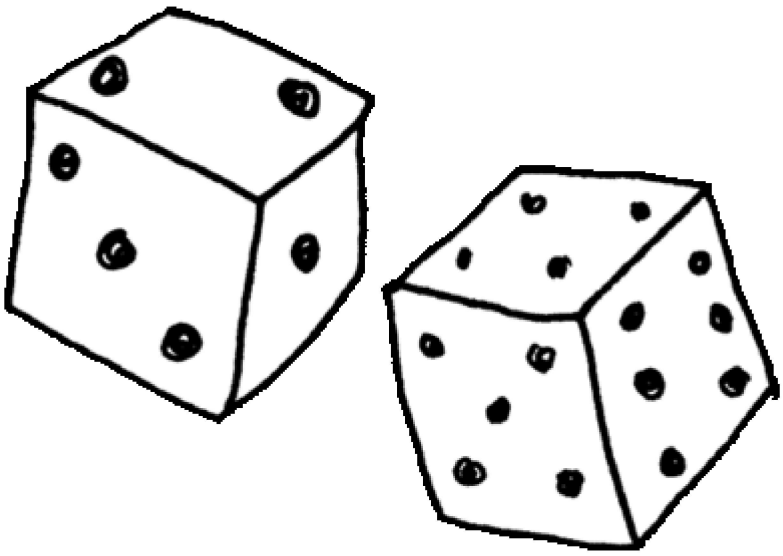
🎉 If the car price is between Rs. 20,00,000 and Rs. 30,00,000, there is a discount of 7.5%.

🥳 If the car price is greater than Rs. 30,00,000, there is a discount of 10%.

Are you ready to make your customers happy with these discounts? 🚙 💰

Loops & Iterations

You and your friends are playing a board game where you roll a dice and move your game piece accordingly. However, you want to know the average number rolled by each player to see who's really the luckiest. Write a program that takes a list of rolls as input for each player and uses a for loop to calculate the average roll for each player. The winner gets bragging rights for being the luckiest!



```
player_rolls = [
    [3, 5, 6, 2, 1],  # player 1's rolls
    [4, 4, 6, 3, 2],  # player 2's rolls
    [1, 5, 6, 6, 4]   # player 3's rolls
]
# list of a list
```

```
max_avg = 0
max_index = 0

# Use a for loop to iterate over each player's rolls
# 'index' will be 0, 1, 2, representing Player 1, Player 2, and Player 3
for index in range(len(player_rolls)):
    current_rolls = player_rolls[index]

    # Calculate the average roll: sum(rolls) / len(rolls)
    total_sum = sum(current_rolls)
    num_rolls = len(current_rolls)
    average_roll = total_sum / num_rolls

    # Print the player's average roll
    # Note: Player number is index + 1
    print(f"Player {index + 1}'s rolls: {current_rolls}")
```

```
        print(f"Player {index + 1}'s average roll: {average_roll:.2f}")

        # Check if this player has the new highest average
        if average_roll > max_avg:
            max_avg = average_roll
            max_index = index # Store the index of the luckiest player

        print("-" * 30) # Separator for readability

    # Determine and print the winner (the player with the highest average roll)
    luckiest_player = max_index + 1
    print(f"The luckiest player is Player {luckiest_player} with an average roll of
```

```
Player 1's rolls: [3, 5, 6, 2, 1]
Player 1's average roll: 3.40
------------------------------
Player 2's rolls: [4, 4, 6, 3, 2]
Player 2's average roll: 3.80
------------------------------
Player 3's rolls: [1, 5, 6, 6, 4]
Player 3's average roll: 4.40
------------------------------
The luckiest player is Player 3 with an average roll of 4.40!
```

Instructions:

1. Define a list of rolls for each player, where each list contains the numbers rolled by the player.
2. Use a for loop to iterate over each player's rolls in the list of player rolls.
3. For each player, calculate the average roll by summing up all the rolls and dividing by the number of rolls. You can use the formula sum(rolls) / len(rolls) to do this.
4. Print out each player's average roll using f-strings to display the player number and their average roll to two decimal places.

Write a Python program that analyzes a list of names and sorts them into two categories: those that start with a vowel and those that start with a consonant. To achieve this, you'll use a nested loop and some basic conditional statements.

```
names = ["Alice", "Bob", "Eve", "Charlie", "Ivy", "David", "Olivia", "Peter"]

VOWELS = {'a', 'e', 'i', 'o', 'u'}

# Names starting with a vowel
```

```
vowel_names = [
    name for name in names
    if name[0].lower() in VOWELS
]

# Names starting with a consonant
consonant_names = [
    name for name in names
    if name[0].lower() not in VOWELS
]

# Print results
print("Vowels:", vowel_names)
print("Consonants:", consonant_names)
```

```
Vowels: ['Alice', 'Eve', 'Ivy', 'Olivia']
Consonants: ['Bob', 'Charlie', 'David', 'Peter']
```

Instructions

1. Start by defining a list of names. You can call this list names and initialize it with a few names.
2. Define two empty lists to store the names that start with vowels and consonants, respectively. You can call these lists vowel_names and consonant_names.
3. Use a nested loop to iterate through each name in the names list. The outer loop should iterate through each name, while the inner loop should iterate through each character in the name.
4. Inside the inner loop, use an if statement to check if the first letter of the name is a vowel or a consonant. You can do this by checking if the first character of the name is in a list of vowels.
5. If the first letter is a vowel, append the name to the vowel_names list using the append() method.
6. If the first letter is a consonant, append the name to the consonant_names list using the append() method.
7. Once the loop has finished iterating through all the names, use the print() function to print the two lists of names. You can include a message indicating which list contains the names that start with vowels and which list contains the names that start with consonants

You are a teacher and you have a dictionary containing the grades of your students in a particular subject. The keys of the dictionary are the names of the students, and the values are their respective grades. You need to find the students who scored above 90% and print out their names.

```
# Define the grades dictionary
grades = {
    'Alice': 85,
    'Bob': 92,
    'Charlie': 88,
    'David': 95,
    'Emily': 78,
    'Frank': 91
}

print("--- Analyzing Student Grades ---")
print("Students who scored above 90% are:")
print("-" * 30)

# Loop through the dictionary using the .items() method to get both the student
for name, grade in grades.items():

    # Check if the grade is greater than 90.
    if grade > 90:
        # If it is, print out the name of the student along with their score.
        print(f"{name} (Score: {grade}%)")

print("-" * 30)
```

```
--- Analyzing Student Grades ---
Students who scored above 90% are:
------------------------------
Bob (Score: 92%)
David (Score: 95%)
Frank (Score: 91%)
------------------------------
```

Instruction

1. Define a dictionary to store the grades of the students. The keys of the dictionary should be the names of the students, and the values should be their respective grades.

2. Loop through the dictionary using a for loop and retrieve the grade of each student.
3. Check if the grade is greater than 90%. If it is, print out the name of the student.
4. Repeat steps 2-3 for all the students in the dictionary.
5. Once the loop is complete, the names of the students who scored above 90% will have been printed to the console.

You're given a nested list of numbers, and your task is to count the number of odd and even numbers in the list. To achieve this, you'll use a nested loop and some basic conditional statements.

List - numbers = [[2, 5, 11, 20, 8], [9, 4, 15, 28, 17], [1, 6, 21, 18, 3], [10, 13, 25, 33, 30], [14, 7, 16, 19, 22]]

Instructions

1. Inititate the odd and even count variables.
2. Iterate through the outer list first. ie, [2,5,11,20,8] would be the item in the first iteration.
3. Iterate through this resulting list using an inner for loop.
4. Check if the number is odd or even, and add the count to the counter variables initialised before.
5. Print the odd and even counts.

```
# List of numbers
numbers = [
    [2, 5, 11, 20, 8],
    [9, 4, 15, 28, 17],
    [1, 6, 21, 18, 3],
    [10, 13, 25, 33, 30],
    [14, 7, 16, 19, 22]
]

# My code here:

# Initiate the odd and even count variables
even_count = 0
odd_count = 0
```

```
print("Starting analysis of the nested list...")

# Iterate through the outer list (each inner list/row)
for row in numbers:
    print(f"Processing row: {row}")

    # Iterate through this resulting list using an inner for loop (each number)
    for num in row:
        # Check if the number is odd or even (using the modulo operator)
        # A number is even if the remainder when divided by 2 is 0.
        if num % 2 == 0:
            # Add the count to the even counter
            even_count += 1
        else:
            # Add the count to the odd counter
            odd_count += 1

print("\n--- Final Counts ---")
# Print the odd and even counts.
print(f"Total Even Numbers found: {even_count}")
print(f"Total Odd Numbers found: {odd_count}")
```

```
Starting analysis of the nested list...
Processing row: [2, 5, 11, 20, 8]
Processing row: [9, 4, 15, 28, 17]
Processing row: [1, 6, 21, 18, 3]
Processing row: [10, 13, 25, 33, 30]
Processing row: [14, 7, 16, 19, 22]

--- Final Counts ---
Total Even Numbers found: 12
Total Odd Numbers found: 13
```

Imagine you're organizing a chess tournament 🏆, where each player has to play against every other player. To achieve this, you can use nested loops in Python! Nested loops are simply loops within loops, and they come in handy when you need to perform tasks involving multiple levels of iteration.

In Python, you can nest 'for' loops, 'while' loops, or even a combination of both. Let's take a closer look at how nested loops work using our chess tournament scenario:

```
# Define the list of players
players = ['Alice', 'Bob', 'Charlie', 'Diana']
```

```
print("--- Chess Tournament Schedule ---")

# Iterate through the list in the first (outer) loop
for player_x in players:
    # Iterate through the same list in the inner loop
    for player_y in players:

        # Compare the elements. If they are not the same players...
        if player_x != player_y:
            # Print the match pairing
            print(f"Player {player_x} plays against Player {player_y}")

print("---------------------------------")
print(f"Total matches: {len(players) * (len(players) - 1)}")
```

```
--- Chess Tournament Schedule ---
Player Alice plays against Player Bob
Player Alice plays against Player Charlie
Player Alice plays against Player Diana
Player Bob plays against Player Alice
Player Bob plays against Player Charlie
Player Bob plays against Player Diana
Player Charlie plays against Player Alice
Player Charlie plays against Player Bob
Player Charlie plays against Player Diana
Player Diana plays against Player Alice
Player Diana plays against Player Bob
Player Diana plays against Player Charlie
---------------------------------
Total matches: 12
```

Instructions

1. Inititiate the list.
2. Iterate through the list in first loop.
3. Iterate throught the same list in inner loop.
4. Compare the element in the outer and inner loops. If they are not the same players, print "Player x plays against Player y".

Conditional & Infinite Looping

Building a Simple Calculator

In this activity, you will build a simple calculator using conditional looping in Python. The calculator will ask the user for two numbers and an operation (addition, subtraction,

multiplication, or division), and then perform the operation on the two numbers.

Instructions:

1. First, enter your first choice number.
2. Second, enter your second choice number.
3. Choose or enter the operation you want to perform.User Input will be "(addition, subtraction, multiplication, or division)".
4. Use a conditional loop to perform the appropriate operation you want to perform.
5. Finally, print the result of the operation.

```
# My code here
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

print("1.Addition 2.Substraction 3.Multiplication 4.Division ")
choice = int(input("Please Select: "))
while choice >= 1 and choice <= 4:
  if choice == 1:
    print(num1 + num2)
    break
  elif choice == 2:
    print(num1 + num2)
    break
  elif choice == 3:
    print(num1 * num2)
    break
  elif choice == 4:
    print(num1 / num2)
    break
else:
  print("Invalid Input")
```

```
Enter the first number: 25
Enter the second number: 62
1.Addition 2.Substraction 3.Multiplication 4.Division
Please Select: 3
1550.0
```

Building a Guessing Game

In this activity, you will build a simple guessing game using conditional looping in Python. The game will generate a random number between 1 and 100, and the user will have to guess the number within a certain number of tries. The program will provide hints to the user after each guess, telling them whether their guess is too high or too low.

What is the random module?

The random module is a built-in Python module that provides a suite of functions for generating random numbers and sequences. This module is often used in applications where randomization is important, such as games, simulations, and cryptography.

You do not need to worry much about this as you will be learning about these modules and functions in the upcoming classes.

Instructions:

1. First, import the "random" module to generate a random number for the game.
2. Next, enter the number of tries you want to have and set a counter to keep track of how many tries you are left with.

3. Now, use a while loop to allow yourself to keep guessing until you either guess the correct number or run out of tries.
4. Finally, if you run out of tries without guessing the correct number, print a message telling what the secret number was.

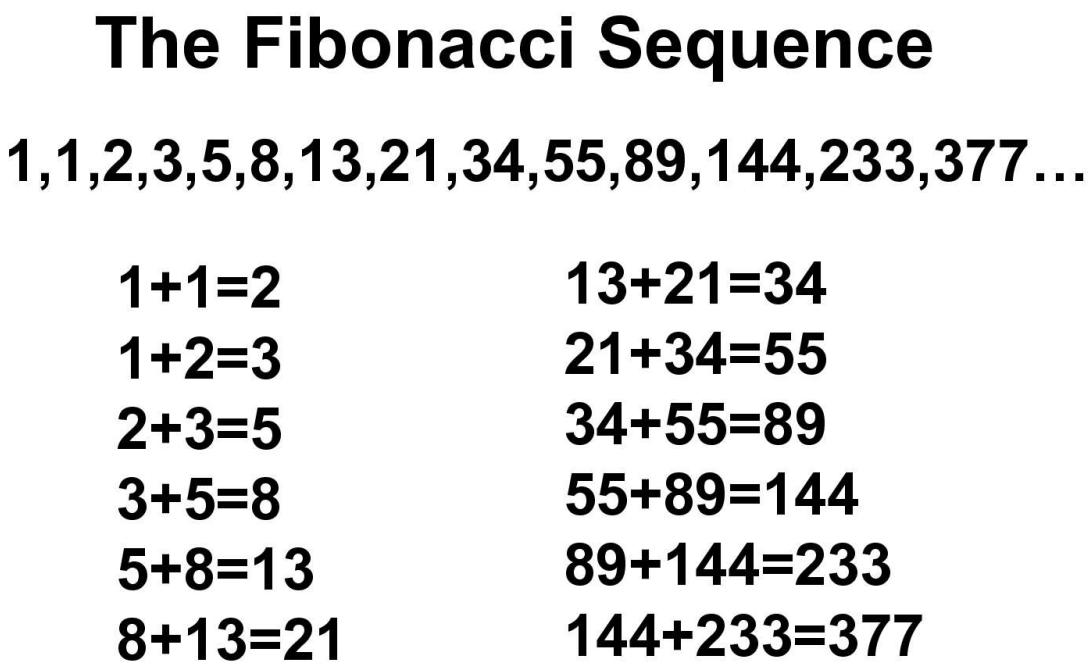
```
import random

# Generate a random number between 1 and 100
secret_number = random.randint(1, 100)

# My code here
tries = 7

print("Guess a number (1-100) in 7 tries.")

while tries > 0:
    try:
        guess = int(input(f"Tries left ({tries}): "))
    except:
        print("Invalid. Enter a number.")
        continue

    if guess == secret_number:
        print(f" Correct! The number was {secret_number}.")
        break
    elif guess < secret_number:
        print("Too low.")
    else:
        print("Too high.")

    tries -= 1
else:
    print(f"Game Over. The number was **{secret_number}**.")
```

```
Guess a number (1-100) in 7 tries.
Tries left (7): 52
Too high.
Tries left (6): 10
Too low.
Tries left (5): 50
Too high.
Tries left (4): 49
Too high.
Tries left (3): 30
Too high.
Tries left (2): 20
Too low.
Tries left (1): 2
Too low.
Game Over. The number was **25**.
```

Building a Fibonacci Sequence Generator

In this activity, you will build a program that generates a Fibonacci sequence using conditional looping in Python. The Fibonacci sequence is a series of numbers in which each number is the sum of the two preceding numbers. The sequence starts with 0 and 1, and the next number in the sequence is the sum of the previous two numbers.

Double-click (or enter) to edit




```
# My code here
terms = int(input("Enter how many terms you want in sequence: "))
count = 0

term1 = int(input("Enter the first term of the sequence: "))
term2 = int(input("Enter the second term of the sequence: "))

if terms < 0:
  print("Please enter a positive integer, the given number is not valid")
```

```
#if there is only one term, it will return n_1
elif terms == 1:
  print("The Fibonacci sequence of the numbers up to", terms, ":")
  print(term1)

#Then we will generate Fibonacci sequence of number
else:
  print("The fibonacci sequence of the numbers is:")
  while count < terms:
    print(term1)
    nth = term1 + term2

#At last, we will update values
    term1 = term2
    term2 = nth
    count = count+1
```

```
Enter how many terms you want in sequence: 5
Enter the first term of the sequence: 2
Enter the second term of the sequence: 2
The fibonacci sequence of the numbers is:
2
2
4
6
10
```

Instructions:

1. First, enter how many terms you want in the sequence.
2. Then, initialize the first two terms of the sequence.
3. Use a while loop to generate the remaining terms of the sequence.
4. Finally, print the sequence.

Counting the Digit

You have been given a task to write a Python program that counts the number of times a specific digit appears in a range of numbers. You need to use a for loop to iterate over the range of numbers, and a while loop to check each digit in the number.

Example:

- Enter start number: 100
- Enter end number: 150
- Enter digit to count: 5

- The digit 5 appears 6 times between 100 and 150.

```
# My code here
count = 0
start_num = int(input("Enter the starting number: "))
end_num = int(input("Enter the ending number: "))
digit_to_count = int(input("Enter the digit to count: "))

for i in range(start_num, end_num + 1):
    temp_num = i
    while temp_num > 0:
        # Check if the last digit matches the target digit
        if temp_num % 10 == digit_to_count:
            count += 1

        # Remove the last digit
        temp_num //= 10

print(f"The digit {digit_to_count} appears {count} times between {start_num} ar
```

```
Enter the starting number: 100
Enter the ending number: 150
Enter the digit to count: 5
The digit 5 appears 6 times between 100 and 150 (inclusive)
```

Instructions:

1. First, input two numbers as the range (start and end) between which the program should count the occurrence of a digit.
2. Then, input the digit that needs to be counted.
3. Initialize a counter variable to keep track of the number of occurrences of the digit.
4. Use a for loop to iterate over the range of numbers between the start and end values (inclusive).
5. Within the for loop, use a while loop to check each digit in the current number. If the digit matches the digit to be counted, increment the counter variable.
6. After the for loop completes, print the total count of the digit that was entered by the user.

Custom Functions in Python

Building a Calculator

In this activity, we will create a custom function in Python to build a calculator that can perform basic arithmetic operations such as addition, subtraction, multiplication, and division.

💻 Code:

```
# Define a function called calculator that takes three arguments: num1, num2, a
def calculator (num1, num2, operation):
  if operation == "+":
    return num1+num2
  elif operation == "-":
    return num1-num2
  elif operation == "*":
    return num1*num2
  elif operation== "/":
    return num1/num2
```

🧪 Test Code:

```
# Test the calculator function
calculator(16, 8, "/")
```

```
2.0
```

Instructions:

1. Define a function called calculator that takes three arguments: num1, num2, and operation.
2. In the function, use an if-else statement to determine the arithmetic operation to perform based on the value of operation.
3. If operation is '+', return the sum of num1 and num2.
4. If operation is '-', return the difference between num1 and num2.
5. If operation is '*', return the product of num1 and num2.
6. If operation is '/', return the quotient of num1 and num2.
7. If operation is not one of the four valid arithmetic operations, return the string "Invalid operation".
8. Test your function by calling it with different values for num1, num2, and operation.

Dice Rolling 🎲

Roll a dice and display the result. 🎲

What is the random module?

The random module is a built-in Python module that provides a suite of functions for generating random numbers and sequences. This module is often used in applications where randomization is important, such as games, simulations, and cryptography.

You do not need to worry much about this as you will be learning about these modules and functions in the upcoming classes.

Instructions:

1. The random module shoud be imported to generate a random number for the dice roll.
2. roll_dice() is defined as a custom function that generates a random number between 1 and 6.
3. The function returns the result of the dice roll.
4. The result variable is assigned to the return value of the roll_dice() function.
5. The result is printed using a formatted string.

Celsius to Fahrenheit Converter

Create a custom function that takes in a temperature in Celsius and returns the temperature in Fahrenheit.

For example: If the input temperature is 20 degrees Celsius, the output temperature should be 68 degrees Fahrenheit.

💻 Code:

```
# Define a function to convert celsius to fahrenheit
def celsius_to_fahrenheit(celsius):
    """
    Converts a temperature from Celsius to Fahrenheit using the formula:
    F = (C * 9/5) + 32

    Args:
        celsius (float or int): The temperature in degrees Celsius.

    Returns:
        float: The corresponding temperature in degrees Fahrenheit.
    """
    # Apply the conversion formula
```

```
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit
```

🧪 **Test Code:**

```
# Example usage
# Test case 1: Water freezing point (0°C)
temp_c1 = 0
temp_f1 = celsius_to_fahrenheit(temp_c1)
print(f"{temp_c1}°C is equal to {temp_f1}°F (Water Freezing Point)")

# Test case 2: Water boiling point (100°C)
temp_c2 = 100
temp_f2 = celsius_to_fahrenheit(temp_c2)
print(f"{temp_c2}°C is equal to {temp_f2}°F (Water Boiling Point)")

# Test case 3: Body temperature (37°C)
temp_c3 = 37
temp_f3 = celsius_to_fahrenheit(temp_c3)
# Using formatting to round the result to two decimal places
print(f"{temp_c3}°C is equal to {temp_f3:.2f}°F (Average Body Temperature)")
```

```
0°C is equal to 32.0°F (Water Freezing Point)
100°C is equal to 212.0°F (Water Boiling Point)
37°C is equal to 98.60°F (Average Body Temperature)
```

Output:

If the input temperature is 20 degrees Celsius, the output temperature should be 68 degrees Fahrenheit.

Instructions:

To create this function, you can use the formula to convert the temperature in Celsius to Fahrenheit. The formula for converting Celsius to Fahrenheit is F = (C * 9/5) + 32, where F is the temperature in Fahrenheit and C is the temperature in Celsius.

Sum odd Numbers

In this activity, you will create a custom function in Python that takes a list of integers as an argument and returns the sum of all odd numbers in the list.

💻 Code:

```
# define sum_odd custom function
def sum_odd(lst):
    """
    Calculates the sum of all odd numbers within a list of integers.

    Args:
        lst (list): A list of integers.

    Returns:
        int: The sum of all odd numbers in the list.
    """
    # Inside the function, create a variable named result and set its value to
    result = 0

    # Use a for loop to iterate through each element in lst.
    for number in lst:
        # For each element, check if it is odd by using the modulus operator (%
        # If the result of number % 2 is NOT 0, the number is odd.
        if number % 2 != 0:
            # If the number is odd, add the element to result.
            result += number

    # After the loop, return result.
    return result
```

🧪 Test Code:

```
# test here
# Test case 1: Standard input from instructions
test_list1 = [1, 2, 3, 4, 5, 6]
output1 = sum_odd(test_list1)
print(f"Input: {test_list1}, Output: {output1}")  # Expected: 9 (1+3+5)

# Test case 2: List with only even numbers
test_list2 = [10, 20, 30]
output2 = sum_odd(test_list2)
print(f"Input: {test_list2}, Output: {output2}")  # Expected: 0

# Test case 3: List with only odd numbers
test_list3 = [11, 13, 15]
output3 = sum_odd(test_list3)
print(f"Input: {test_list3}, Output: {output3}")  # Expected: 39 (11+13+15)
```

```
Input: [1, 2, 3, 4, 5, 6], Output: 9
Input: [10, 20, 30], Output: 0
Input: [11, 13, 15], Output: 39
```

🤖 Output should be in format:

If the input is [1, 2, 3, 4, 5, 6], then the output should be 9

Instructions:

To create the sum_odd function, follow these steps:

1. Define a function named sum_odd that takes one argument, lst.
2. Inside the function, create a variable named result and set its value to 0.
3. Use a for loop to iterate through each element in lst.
4. For each element, check if it is odd by using the modulus operator (%). If the result not is 0, add the element to result.
5. After the loop, return result.

Advanced Looping Concepts

Lambda Functions

👩‍💻 You are working as a Python developer for a startup company that has just received a project from a client. The client wants a program that can perform certain operations on lists using lambda functions. Your manager has assigned you the task of finding the sum of squared odd numbers from the list.

👉 Hint: You have to use filter, map and reduce functions to get the task done.

```
# Import reduce from functools module
from functools import reduce

# Create a list of numbers from 1 to 10
numbers = list(range(1, 11))

# 1. Find the sum of squared odd numbers by chaining filter, map, and reduce:
# 2. filter(lambda x: x % 2 != 0, numbers): Filters for odd numbers.
# 3. map(lambda x: x ** 2, ...): Squares the resulting odd numbers.
# 4. reduce(lambda a, b: a + b, ...): Sums the final squared numbers.
final_sum = reduce(
    lambda a, b: a + b,
    map(
        lambda x: x ** 2,
        filter(lambda x: x % 2 != 0, numbers)
    )
)

# 5. Print the final result
print(f"Original List: {numbers}")
```

```
print(f"Sum of Squared Odd Numbers: {final_sum}")
```

```
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Sum of Squared Odd Numbers: 165
```

Instructions:

1. Begin by creating a list of numbers from 1 to 10.
2. Use a lambda function to filter out all even numbers from the list.
3. Use a lambda function to square all the remaining odd numbers in the filtered list.
4. Use a lambda function to find the sum of all the squared odd numbers in the list.
5. Print the final result.

The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in "functools" module. No need to worry if you haven't used this function before. You can get some additional information about the function here: https://www.geeksforgeeks.org/reduce-in-python/

Pizza Toppings

🍕 🍔 You have two lists, one containing pizza toppings and the other containing burger toppings. You need to create a new list that contains all the toppings from both the pizza and burger toppings lists, but only if they have more than 5 characters in their name.

Can you write the code to create a new list that contains all the toppings from both the pizza and burger toppings lists, but only if they have more than 5 characters in their name using list comprehension only?

👉 Hint: You can use a conditional statement to check the length of each topping's name.

```
# Here are the example lists of toppings
pizza_toppings = ['mushroom', 'olive', 'tomato', 'pepperoni', 'onion', 'garlic'
burger_toppings = ['lettuce', 'cheese', 'mayonnaise', 'bacon', 'pickle', 'avoca

# Use list comprehension to combine toppings and filter name with more than 5 c
filtered_toppings = [topping for topping in pizza_toppings + burger_toppings if

#Print the result
print("Filtered topping (more than 5 characters):", filtered_toppings)
```

```
Filtered topping (more than 5 characters): ['mushroom', 'tomato', 'pepperoni', '
```

Instructions:

1. Create two lists of toppings, one for pizza and one for burgers.
2. Use list comprehension to iterate over both the pizza and burger toppings lists, and combine them into a new list.
3. Use a conditional statement to check the length of each topping's name, and only include it in the new list if it has more than 5 characters.

Sales Data

🌎🛍️ Suppose you work for a company that sells products in different countries. You have been given two lists: one containing the names of the countries where the company sells its products, and the other containing the sales data for each country. Your task is to create a dictionary where the keys are the country names, and the values are the corresponding sales data. However, the sales data should only include values that are

greater than 1000. You are not allowed to use any loops, and must use list comprehension to solve the problem.

Can you write the code to create the desired dictionary using list comprehension?

👉 Hint: You can use zip function to create pairs of country and sales data from the two lists.


```
# Here are the two lists
countries = ["USA", "Canada", "Mexico", "Brazil", "UK", "France", "Germany", "C
sales = [2500, 300, 1200, 800, 500, 2000, 4000, 1000, 1500]

# Create the dictionary using a single Dictionary Comprehension.
# 1. zip(countries, sales): Creates pairs like ('USA', 2500), ('Canada', 300),
# 2. if sale > 1000: This is the filtering condition applied within the compreh
# 3. country: sale: Defines the key-value pair for the resulting dictionary.

high_value_sales = {
    country: sale
    for country, sale in zip(countries, sales)
    if sale > 1000
}

# Print the final dictionary
print("Countries with Sales > 1000:")
print(high_value_sales)
```

```
Countries with Sales > 1000:
{'USA': 2500, 'Mexico': 1200, 'France': 2000, 'Germany': 4000, 'India': 1500}
```

Instructions:

1. Use zip function to create a list of pairs, where each pair contains a country name and its corresponding sales data.
2. Use list comprehension to create a dictionary where the keys are the country names and the values are the sales data. However, only include sales data that are greater than 1000.

Stock Filtering

📊📈 Suppose you are working for a company that deals with financial data. You have been given a list of dictionaries, where each dictionary contains the following information about a stock: name, ticker symbol, price, and percentage change in price. Your task is to create a new list of dictionaries, where each dictionary contains only the name and price of the stock, but only for those stocks where the price is greater than 100 and the percentage change is positive. You must use list comprehension to solve the problem.

Can you write the code to create the desired list of dictionaries using list comprehension?

👉 Hint: You can use conditional statements to check if the price is greater than 100 and the percentage change is positive in the list comprehension.


```
# Here is an example of the list of dictionaries
stocks = [
    {'name': 'Apple Inc.', 'ticker': 'AAPL', 'price': 120.0, 'change': 0.05},
    {'name': 'Microsoft Corporation', 'ticker': 'MSFT', 'price': 95.0, 'change'
    {'name': 'Amazon.com, Inc.', 'ticker': 'AMZN', 'price': 250.0, 'change': 0.
    {'name': 'Alphabet Inc.', 'ticker': 'GOOGL', 'price': 110.0, 'change': 0.02
    {'name': 'Facebook, Inc.', 'ticker': 'FB', 'price': 80.0, 'change': 0.03}
```

```
]

# Write your code here

#Createa a list of dictionaries where there is only the name and price.
new_stocks= [{'name':stock['name'], 'price':stock['price']} for stock in stocks
new_stocks
```

```
[{'name': 'Apple Inc.', 'price': 120.0},
 {'name': 'Microsoft Corporation', 'price': 95.0},
 {'name': 'Amazon.com, Inc.', 'price': 250.0},
 {'name': 'Alphabet Inc.', 'price': 110.0},
 {'name': 'Facebook, Inc.', 'price': 80.0}]
```

```
#Create a list of dictionaries where the price is greater than 100 and the char
filtered_stocks = [stock for stock in stocks if stock['price'] > 100 and stock[

#Create a new list of dictionaries with only the name and price.
new_stocks_filtered = [{'name':stock['name'], 'price':stock['price']} for stock

# Print the new list of dictionaries
print("Filtered Stocks:")
print(new_stocks_filtered)
```

```
Filtered Stocks:
[{'name': 'Apple Inc.', 'price': 120.0}, {'name': 'Amazon.com, Inc.', 'price': 2
```

```
new_stocks
```

```
[{'name': 'Apple Inc.', 'price': 120.0},
 {'name': 'Microsoft Corporation', 'price': 95.0},
 {'name': 'Amazon.com, Inc.', 'price': 250.0},
 {'name': 'Alphabet Inc.', 'price': 110.0},
 {'name': 'Facebook, Inc.', 'price': 80.0}]
```

Instructions:

1. Use list comprehension to filter the stocks where the price is greater than 100 and the percentage change is positive.
2. Use list comprehension to create a new list of dictionaries where each dictionary contains only the name and price of the stock.

Duplicate Songs

🎵🎶 You have a list of songs that you like to listen to, but you notice that some of the songs have duplicate names. You want to create a new set that contains the unique names of all the songs and keep the songs whose name doesn't start with "S" and doesn't end with "n" in your list using set comprehension.

Can you write the code to create a new set that contains the unique names of all the songs in your list using set comprehension only?

👉 Hint: Set comprehension is similar to list comprehension, but with curly braces {} instead of square brackets [].

```
# Here's an example list of songs
songs = ['Bohemian Rhapsody', 'Stairway to Heaven', 'Bohemian Rhapsody', 'Hotel

# Use set comprehension to create a new set that contains only unique songs,
# excluding those that start with 'S' OR end with 'n'.
# The condition is: IF NOT (starts with 'S') AND NOT (ends with 'n')
# However, the simpler logic is to include the song IF it does NOT meet the exc
unique_filtered_songs = {
    song for song in songs
    if not (song.startswith('S') or song.endswith('n'))
}

# Print the final set
print(f"Original List of Songs: {songs}")
print("-" * 35)
print("Unique Filtered Songs (excluding 'S' start or 'n' end):")
print(unique_filtered_songs)
```

```
Original List of Songs: ['Bohemian Rhapsody', 'Stairway to Heaven', 'Bohemian Rh
-----------------------------------
Unique Filtered Songs (excluding 'S' start or 'n' end):
{'Bohemian Rhapsody', 'Hotel California'}
```

Instructions:

1. Create a list of songs that contains duplicate names.
2. Use set comprehension to iterate over the list of songs and create a new set that contains only the unique names of the songs.
3. Include a condition in the set comprehension to filter out songs whose name starts with "S" and ends with "n".

OOPs in Python

Building a Simple Calculator 🧮 🔢

In this activity, you will build a simple calculator program in Python using Object-Oriented Programming (OOP) concepts.

```
# Define a simple calculator class
class Calculator:
  # Method to add two numbers
  def add(self, num1, num2):
    return num1 + num2

  # Method to subtract two numbers
  def subtract(self, num1, num2):
    return num1 - num2

  # Method to multiply two numbers
  def multiply(self, num1, num2):
    return num1 * num2

   # Method to divide two numbers with error handling for division by zero
  def divide(self, num1, num2):
    if num2 == 0:
      raise ValueError("Cannot divide by zero")
    return num1 / num2

# Example usage
calc = Calculator()
```

```
# Test Case
print("Addition:", calc.add(10, 5))           #Output: 15
print("Subtraction:", calc.subtract(10, 5))   #Output: 5
print("Multiplication:", calc.multiply(10, 5))#Output: 50
print("Division:", calc.divide(10, 5))        #Output: 2.0
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

👉 Instructions:

1. Start by defining a class called Calculator.
2. Define four methods within the class: add, subtract, multiply, and divide.
3. Each method should take two arguments: num1 and num2.
4. The add method should return the sum of num1 and num2.

5. The subtract method should return the difference between num1 and num2.
6. The multiply method should return the product of num1 and num2.
7. The divide method should return the result of dividing num1 by num2. However, if num2 is zero, the method should raise a ValueError with the message "Cannot divide by zero".

🏀 Basketball Game Score Tracker 📈

Create a class that will keep track of the score of a basketball game. The class should have methods for updating the score and printing the current score.

```
# Create BasketballGame class
class BasketballGame:
    """
    A class to track the score of a basketball game between two teams.
    """
    def __init__(self):
        # Initialize the score for both teams to 0
        self.team_a_score = 0
        self.team_b_score = 0
        print("New Basketball Game started! Scores reset to 0-0.")

    def score_team_a(self, points):
        """Adds points to Team A's score."""
        if points > 0:
            self.team_a_score += points
            print(f"Team A scores {points} points.")
        else:
            print("Score update must be positive.")

    def score_team_b(self, points):
        """Adds points to Team B's score."""
```

```
            if points > 0:
                self.team_b_score += points
                print(f"Team B scores {points} points.")
            else:
                print("Score update must be positive.")

    def print_score(self):
        """Prints the current score for both teams."""
        print("\n*** CURRENT SCORE ***")
        print(f"Team A: {self.team_a_score}")
        print(f"Team B: {self.team_b_score}")
        print("**********************\n")
```

```
# Example Usage:
# Create an instance of the "BasketballGame" class
game = BasketballGame()

# Update the score for each team multiple times
game.print_score() # Initial score (0-0)

# Team A gets a 3-pointer
game.score_team_a(3)
# Team B gets a 2-point shot
game.score_team_b(2)
# Team B makes a free throw
game.score_team_b(1)
# Team A gets a fast break layup (2 points)
game.score_team_a(2)

# Print the current score
game.print_score()

# Team A gets another 3-pointer
game.score_team_a(3)

# Print the final score
game.print_score()
```

```
New Basketball Game started! Scores reset to 0-0.

*** CURRENT SCORE ***
Team A: 0
Team B: 0
**********************

Team A scores 3 points.
Team B scores 2 points.
Team B scores 1 points.
Team A scores 2 points.

*** CURRENT SCORE ***
```

```
Team A: 5
Team B: 3
********************

Team A scores 3 points.

*** CURRENT SCORE ***
Team A: 8
Team B: 3
********************
```

👉 **Instructions:**

1. Start by creating a class called "BasketballGame".
2. In the class constructor, initialize the score for both teams to 0.
3. Add methods to the class for updating the score for each team. These methods should take in a number of points and add that to the appropriate team's score.
4. Add a method to the class for printing the current score. This method should print out the score for both teams.
5. Test your code by creating an instance of the "BasketballGame" class, updating the score for each team, and printing the current score.

🎨 Paint Brush Class 🖌️

Create a Python class for a paint brush that has the following properties:

- Size (small, medium, or large)
- Color (red, blue, green, or yellow)
- Brand (Winsor & Newton, Liquitex, or Grumbacher)
- Type (round or flat)

```
# Define PaintBrush Class
class PaintBrush:
    """
    A class representing a paint brush with specific size, color, brand, and ty
    """
    def __init__(self, size, color, brand, brush_type):
        # Assign the values of the parameters to instance variables
        self.size = size
        self.color = color
        self.brand = brand
        self.type = brush_type
        print(f"A new {self.brand} {self.type} brush has been created.")

    def paint(self):
```

```
        """
        Simulates painting with the brush, printing out a message with its prop
        """
        print("\n--- Painting Simulation ---")
        print(f"Painting a vibrant {self.color} color.")
        print(f"Using a {self.size}-sized, {self.type} brush.")
        print(f"The quality comes from the {self.brand} brand.")
        print("---------------------------\n")
```

Now, create an instance of the PaintBrush class with the following properties:

- Size: medium
- Color: red
- Brand: Winsor & Newton
- Type: round
- Call the paint() method on the instance to simulate painting with the brush.

```
# Test Case
# Create an instance of the PaintBrush class with the required properties
brush_instance = PaintBrush(
    size="medium",
    color="red",
    brand="Winsor & Newton",
    brush_type="round"
)

# Call the paint() method on the instance to simulate painting
brush_instance.paint()
```

```
A new Winsor & Newton round brush has been created.

--- Painting Simulation ---
Painting a vibrant red color.
Using a medium-sized, round brush.
The quality comes from the Winsor & Newton brand.
---------------------------
```

👉 **Instructions:**

1. Start by defining the class using the keyword class, followed by the name of the class (in this case, "PaintBrush").
2. Within the class, define the **init** method to initialize the properties of the paint brush. The method should take four parameters: size, color, brand, and brush_type. Use the self keyword to refer to the instance of the class.

3. Inside the **init** method, assign the values of the parameters to instance variables using the self keyword.
4. Define a method named paint that prints out a message indicating the brush size, color, brand, and type.
5. Create an instance of the PaintBrush class by calling the class with the required parameters.
6. Call the paint() method on the instance to simulate painting with the brush.

Bank Account Management System 💻🏦

Create a Bank Account Management System using Object-Oriented Programming (OOPs) in Python. The system should allow users to create and manage bank accounts, deposit and withdraw funds, and view account details.

```
# Define a BankAccount class
class BankAccount:
    """
    Manages basic bank account operations including deposit, withdrawal,
    and viewing account details.
    """
    def __init__(self, name, account_number, balance=0.0):
        self.name = name
        self.account_number = account_number
        self.balance = balance
        print(f"Account created for {self.name}. Initial balance: ${self.balanc

    def deposit(self, amount):
        """Adds funds to the account balance."""
        if amount > 0:
```

```
            self.balance += amount
            print(f"Deposit successful. Amount: ${amount:.2f}. New balance: ${s
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        """Deducts funds from the account balance if sufficient funds are avail
        if amount > 0:
            if self.balance >= amount:
                self.balance -= amount
                print(f"Withdrawal successful. Amount: ${amount:.2f}. New balar
            else:
                print(f"Withdrawal failed. Insufficient funds. Current balance:
        else:
            print("Withdrawal amount must be positive.")

    def account_details(self):
        """Prints the current account information."""
        print("\n--- Account Details ---")
        print(f"Account Holder: {self.name}")
        print(f"Account Number: {self.account_number}")
        print(f"Current Balance: ${self.balance:.2f}")
        print("-----------------------\n")
```

```
# Test Case
my_account = BankAccount("Alex Johnson", "9876543210", 500.00)

my_account.account_details()

# Test Deposit
my_account.deposit(250.50)

# Test successful Withdrawal
my_account.withdraw(100.00)

# Test failed Withdrawal (Insufficient balance)
my_account.withdraw(700.00)

# View final details
my_account.account_details()
```

```
Account created for Alex Johnson. Initial balance: $500.00

--- Account Details ---
Account Holder: Alex Johnson
Account Number: 9876543210
Current Balance: $500.00
-----------------------
```

```
Deposit successful. Amount: $250.50. New balance: $750.50
Withdrawal successful. Amount: $100.00. New balance: $650.50
Withdrawal failed. Insufficient funds. Current balance: $650.50

--- Account Details ---
Account Holder: Alex Johnson
Account Number: 9876543210
Current Balance: $650.50
-----------------------
```

👉 **Instructions:**

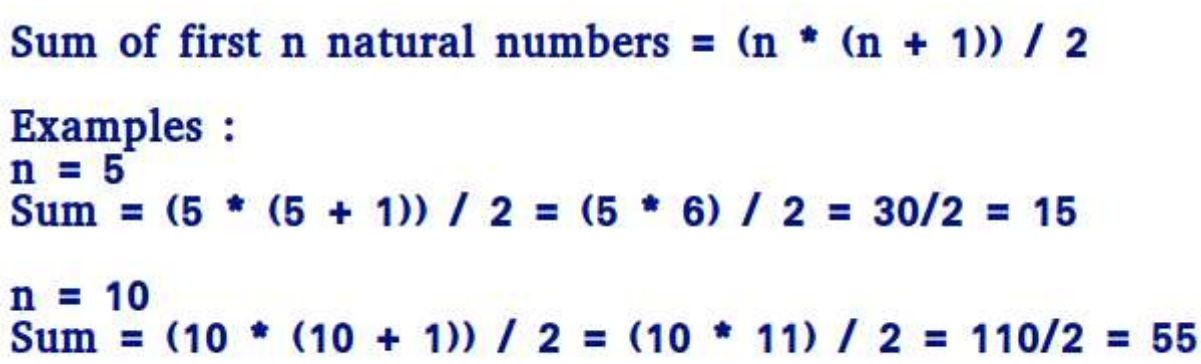
1. Define a BankAccount class with an init() method that takes the name, account_number, and balance as parameters and initializes the instance variables.
2. Define the deposit() method that takes the amount to be deposited as a parameter and adds it to the balance. Also, print a message with the new balance.
3. Define the withdraw() method that takes the amount to be withdrawn as a parameter and checks if the account has sufficient balance. If yes, deduct the amount from the balance and print a message with the new balance. If not, print a message with the current balance.
4. Define the account_details() method that prints the name, account number, and balance of the account.
5. Create a new instance of the BankAccount class with some initial balance.
6. Test the deposit(), withdraw(), and account_details() methods of the BankAccount class by calling them on the instance created in step 5 with some test values.

Exception Handling

Sum of first n Natural Numbers

💻 Suppose you are working on a project that requires you to write a Python program that calculates the sum of the first n natural numbers. You wrote a program to solve the problem, but the code contains several errors that prevent it from executing properly. ❌

Your task is to identify and rectify the errors in the code and write the correct program. 🛠️




```
# Code with errors
n = input("Enter a number: ")
sum = 0
for i in range(0, n+1):
  sum = sum + i
print("The sum of the first", n, "natural numbers is", sum
```

```
# 1. Convert the input string to an integer immediately. (Rectifying the Type E
n_input = input("Enter a number: ")
n = int(n_input)

total_sum = 0
# 2. Iterate from 1 up to and including n.
# range(start, stop) stops *before* 'stop', so we use n + 1.
# Also, using 'total_sum' instead of 'sum' is better practice to avoid overwrit
for i in range(1, n + 1):
    total_sum = total_sum + i

# Print the result using the original string input (n_input) for presentation.
print("The sum of the first", n_input, "natural numbers is", total_sum)
```

```
Enter a number: 20
The sum of the first 20 natural numbers is 210
```

👉 **Instructions:**

1. The input function returns a string, so we need to convert it to an integer using the int() function. 🔢
2. In the range() function, we need to provide an integer value for the upper limit. So, we need to convert the input to an integer as well. ⬆️ TOP
3. We need to add 1 to the upper limit in the range() function to include the last number. ➕

Average Temperature 🌡️🌡️

You have been given a task to develop a program that calculates the average temperature 🌡️ of a city 🏢 for a given week. You have written a program to solve the problem, but there seems to be some logical errors that are causing the program to output incorrect results. You are provided with two sets of weekly temperatures and need to find the average of each individual week.

Your task is to identify and rectify the logical errors in the code and write the correct program.

```
# Temperature data for the week 1 and week 2
temperatures_w1 = [32, 34, 31, 30, 29, 28, 33]
temperatures_w2 = [31, 34, 35, 28, 29]

# Helper function to calculate average concisely
def get_avg(temps):
    return sum(temps) / len(temps)

# Calculate and print results
average_w1 = get_avg(temperatures_w1)
average_w2 = get_avg(temperatures_w2)

print(f"The average temperature of week 1 is {average_w1:.2f} degrees Celsius."
print(f"The average temperature of week 2 is {average_w2:.2f} degrees Celsius."
```

```
The average temperature of week 1 is 31.00 degrees Celsius.
The average temperature of week 2 is 31.40 degrees Celsius.
```

```
# Temperature data for the week 1 and week 2
temperatures_w1 = [32, 34, 31, 30, 29, 28, 33]
temperatures_w2 = [31, 34, 35, 28, 29]

# Initialize sum variable
total_sum = 0

# Calculate the sum of all temperatures
for temperature in temperatures_w1:
  total_sum += temperature # Addb each temperature to total_sum

# Calculate the average temperature
average = total_sum / len(temperatures_w1) # Avoid hardcoding the length

# Output the result
print("The average temperature of the week 1 is", average, "degrees Celsius.")
```

```
The average temperature of the week 1 is 31.0 degrees Celsius.
```

👉 **Instructions:**

1. The given temperature list is in celsius and we need to find the average of each individual week.
2. While iterating through each temperature we need to initialize a variable to start from zero otherwise it can start from any garbage value.
3. Next, find the average by dividing the sum of the temperatures by the length of the temperatures list.
4. Print the average temperature of the each week.

Handle that Error! ❌❌

You are developing a Python program that requires user input. You want to make sure that your program doesn't crash due to invalid user input or unexpected errors 🛑. You have heard about exception handling in Python and want to implement it in your program to handle errors gracefully.

Your task is to implement exception handling in your program to handle invalid user input and unexpected errors.