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?

In [ ]:
```
#1. Define your fixed values:
wins = 3
draws = 1
losses = 0
```

In [ ]:
```
#2. Calculate the total points gained by Team London
london_points = 4*wins + 3*losses + 1*draws
```

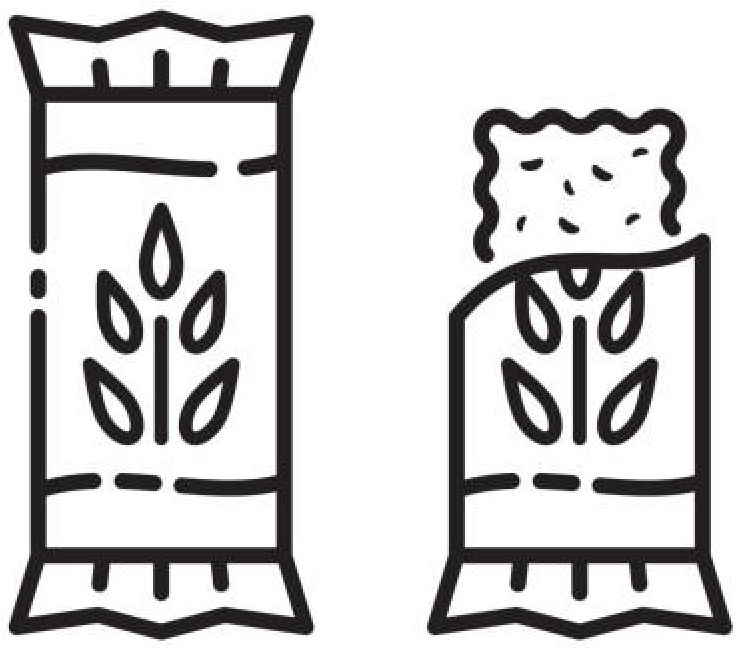
In [ ]:
```
# 3. Print the variable london_points
print(f"london_points: {london_points}")
```

```
london_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?



In [ ]:

```
# Ingredient weights
raisins_weight = 45   # in grams
almonds_weight = 65   # in grams
apricots_weight = 30   # in grams

# Total weight of the nutrition bar
total_weight = raisins_weight + almonds_weight + apricots_weight
```

In [ ]:

```
# Calculate the percentage of raisins and print the variable
raisins_percentage = (raisins_weight / total_weight) * 100
print(f"Raisins: {raisins_percentage:.2f}%")
```

```
Raisins: 32.14%
```

In [ ]:

```
# Calculate the percentage of almonds and print the variable
almonds_percentage = (almonds_weight / total_weight) * 100
print(f"Almonds: {almonds_percentage:.2f}%")
```

```
Almonds: 46.43%
```

In [ ]:

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

```
Apricots: 21.43%
```

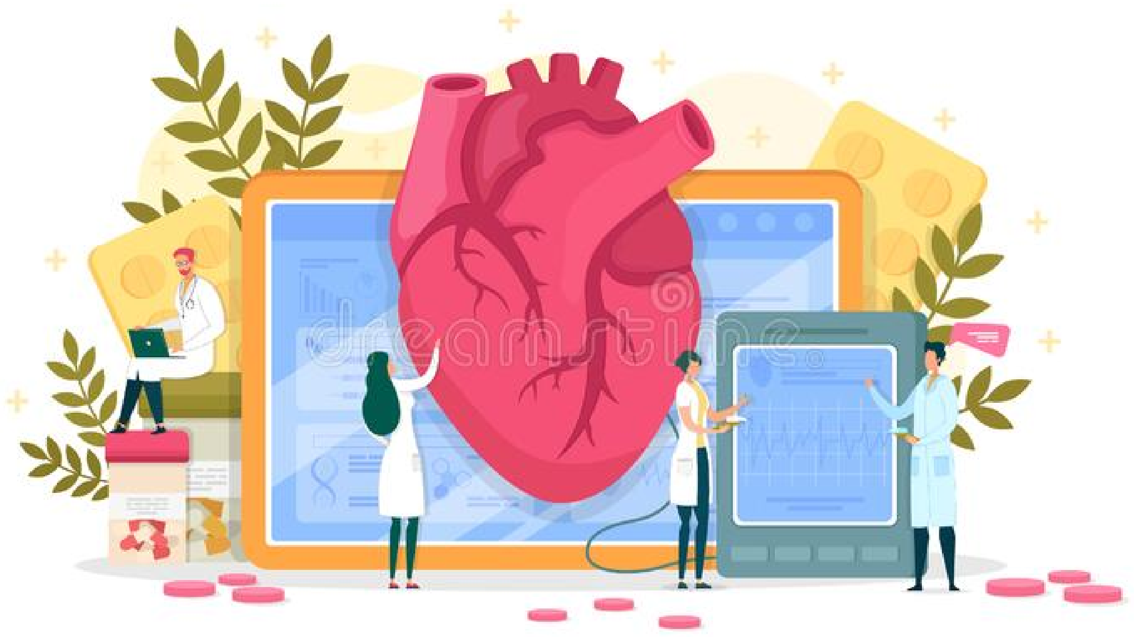
Measuring the Accuracy of a Disease Detection Model-Optional

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.



In [ ]:
```
# Store the number of heart disease patients
heart_disease= 45

# Store the number of diabetes patients
diabetes= 30

# Store the number of cancer patients
cancer= 25

# Store the number of your correct guesses of heart disease patients
heart_disease_correct= 35
```

```
# Store the number of your correct guesses of diabetes patients
diabetes_correct= 25

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

Calculate your accuracy in each of the three cases

```
In [ ]: # Print the heart disease detection accuracy
heart_disease_accuracy= (heart_disease_correct/heart_disease) *100
print(f"Heart Disease Detection Accuracy: {heart_disease_accuracy:.2f}%")
```

```
Heart Disease Detection Accuracy:  77.78%
```

```
In [ ]: # Print the diabetes detection accuracy
diabetes_accuracy= (diabetes_correct/diabetes) *100
print(f"Diabetes Disease Detection Accuracy: {diabetes_accuracy:.2f}%")
```

```
Diabetes Disease Detection Accuracy:  83.33%
```

```
In [ ]: # Print the cancer detection accuracy
cancer_accuracy= (cancer_correct/cancer) *100
print(f"Cancer Disease Detection Accuracy: {cancer_accuracy:.2f}%")
```

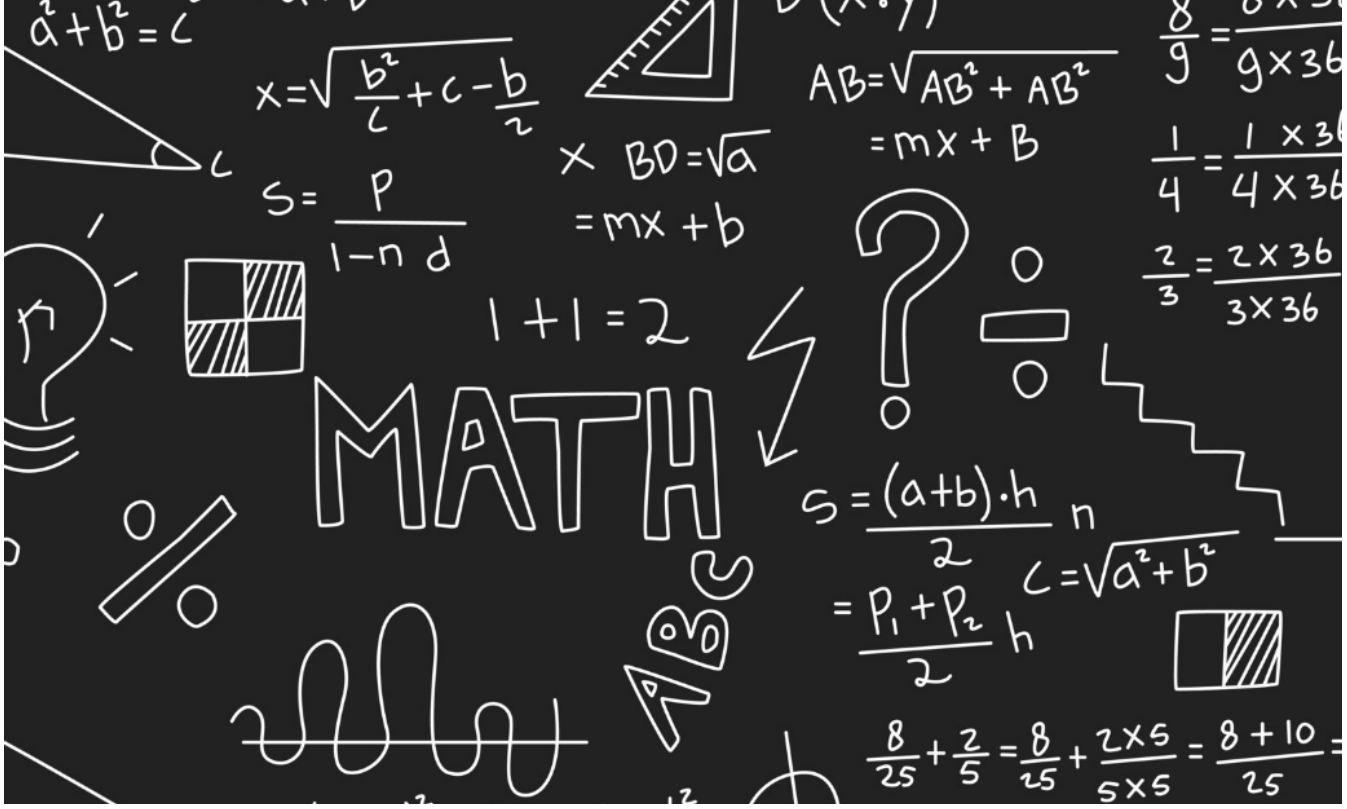
```
Cancer Disease Detection Accuracy:  80.00%
```

```
In [ ]: # Print the overall accuracy of your model
total_correct= heart_disease_correct + diabetes_correct + cancer_correct
total_patients= 100
overall_accuracy= (total_correct/ total_patients) * 100
print(f"Overall Disease Detection Accuracy: {overall_accuracy:.2f}%")
```

```
Overall Disease Detection 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?

In [ ]:
```python
# Calculate the value of the function f(x) at x = 3
x=3
func_evaluated_at_3 = x**2 + x*3 - 4
print(f"Function f(x) at x = 3 is {func_evaluated_at_3}")
```

```
Function f(x) at x = 3 is 14
```

In [ ]:
```python
# Calculate the value of the function f(x) at x = -2
x= -2
func_evaluated_at_minus2 = x**2 + x*3 - 4
print(f"Function f(x) at x = -2 is {func_evaluated_at_minus2}")
```

```
Function f(x) at x = -2 is -6
```

In [ ]:
```python
# Calculate the value of the function f(x) at x = 4
x= 4
func_evaluated_at_4 = x**2 + x*3 - 4
print(f"Function f(x) at x = 4 is {func_evaluated_at_4}")
```

```
Function f(x) at x = 4 is 24
```

In [ ]:
```python
#Check if func_evaluated_at_3 >= func_evaluated_at_minus2
print(f"func_evaluated_at_3 >= func_evaluated_at_minus2: {func_evaluated_at_3>=func_eval
```

```
func_evaluated_at_3 >= func_evaluated_at_minus2: True
```

In [ ]:
```python
#Check if value of f(x) at 3,-2 or 4 is equal to 0 or not
print("Function f(x) at 3 equal to 0: ",func_evaluated_at_3==0)
print("Function f(x) at -2 equal to 0: ",func_evaluated_at_minus2==0)
print("Function f(x) at 4 equal to 0: ",func_evaluated_at_4==0)
```

```
Function f(x) at 3 equal to 0:  False
Function f(x) at -2 equal to 0:  False
Function f(x) at 4 equal to 0:  False
```

Finding total A'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? ▯

In [ ]:
```
# Here's the list of books
books = "Python for Everyone, Learn Python the Hard Way, Python Crash Course , Starting
```

In [ ]:
```
# Store the count of books containing A/a in the title. Note this will be a case insensi
a_counts = books.count('a')

A_counts= books.count('A')

total_a_counts= a_counts + A_counts
```

In [ ]:
```
# Print the value
print(f"The total number of 'a's in the list of books is {total_a_counts}")
```

```
The total number of 'a's in the list of books is 7
```

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! ▯

In [ ]:
```
# List of student marks
marks = [67,87,75,46,89,97,68,98,87,88]
```

In [ ]:
```
# Calculate the mean of the marks. Use sum() and len() functions
mean_marks = sum(marks)/len(marks)
```

In [ ]:
```
# Print the value
print(f"Mean Marks: {mean_marks:.2f}")
```

```
Mean Marks: 80.20
```

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','A

□

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?

In [ ]:
```
#Here's John's list:
fruit_list=['Mango','Mango','Mango','Pineapple','Pineapple','Pineapple','Apple','Mango',
```

In [ ]:
```
# Step 2: Create an empty dictionary which will contain the unique fruit names
fruit_dict={}

# Set the values for each key separately

# First Key
fruit_dict['Mango']= fruit_list.count('Mango')

# Second Key
fruit_dict['Pineapple']= fruit_list.count('Pineapple')

# Third Key
fruit_dict['Apple']= fruit_list.count('Apple')
```

```
# Fourth Key
fruit_dict['Banana']= fruit_list.count('Banana')
```

In [ ]:
```
# Print the dictionary
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']

In [ ]:
```
# Here's your list of characters:

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

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

In [ ]:
```
# Extract the last 6 characters from the 'marvel_words' list in reverse order
new_marvel=marvel_words[:-7:-1]
print("Last 6 characters in the list:")

# Iterate through the elements in 'new_marvel' and print each element
for i in new_marvel:
  print(i)
```

```
Last 6 characters in the list:
Namor
Doctor Strange
Wolverine
Captain America
Black Widow
Thor
```

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.

```
In [ ]:
# Split 'marvel_words' into two teams based on index positions
team1 = marvel_words[::2]    # Characters at even index positions
team2 = marvel_words[1::2]   # Characters at odd index positions

# Print characters from each team
print("Characters located at even index position:")
for i in team1:
  print(i)
print("\n")
print("Characters located at odd index positions:")
for i in team2:
  print(i)
```

```
Characters located at even index position:
Avengers
Spider-Man
Hulk
Black Widow
Wolverine
Namor


Characters located at odd index positions:
X-Men
Iron Man
Thor
Captain America
Doctor Strange
```

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

```
In [ ]:
# Extract the third element ('Spider-Man') from 'marvel_words'
spider_man = marvel_words[2]

# Slice the first 6 characters of 'Spider-Man'
spidey_sliced = spider_man[0:6]

# Print the sliced characters
print(spidey_sliced)
```

```
Spider
```

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

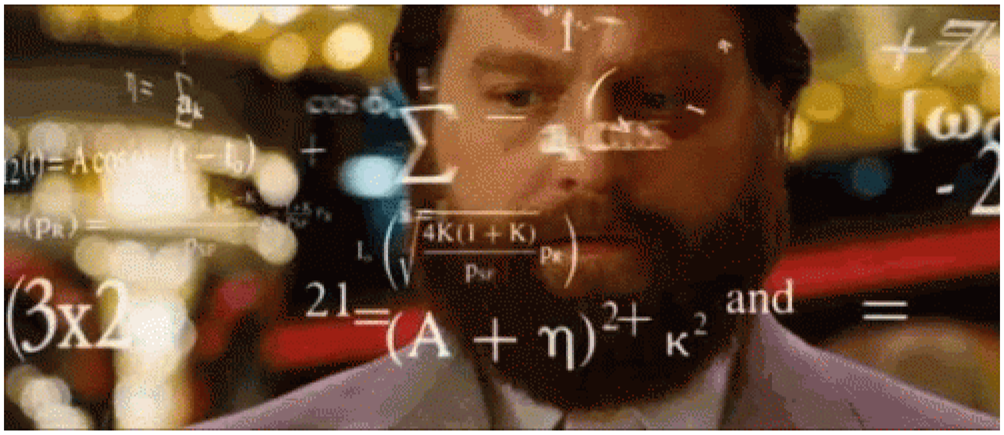
```
In [ ]:
# Extract the fourth element ('Iron Man') from 'marvel_words'
iron_man = marvel_words[3]

# Reverse the characters of 'Iron Man'
reversed_man = iron_man[::-1]

# Print the reversed characters
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?



In [ ]:
```
# Calculate the area of a square with sides measuring 5 cm
ar_square = 5**2

# Calculate the volume of a cube with sides measuring 12 cm
vol_cube = 12**3

# Find the quotient and remainder when 10 is divided by 4
quotient = 10//4
remainder = 10%4

# Print the results with corresponding descriptions
print(f"▯ Area of a square with sides of 5 cm: {ar_square} square cm")
print(f"▯ Volume of a cube with sides of 12 cm: {vol_cube} cubic cm")
print(f"▯ Quotient when dividing 10 by 4: {quotient}")
print(f"    Remainder when dividing 10 by 4: {remainder}")
```

```
▯ Area of a square with sides of 5 cm: 25 square cm
▯ Volume of a cube with sides of 12 cm: 1728 cubic cm
▯ Quotient when dividing 10 by 4: 2
    Remainder when dividing 10 by 4: 2
```

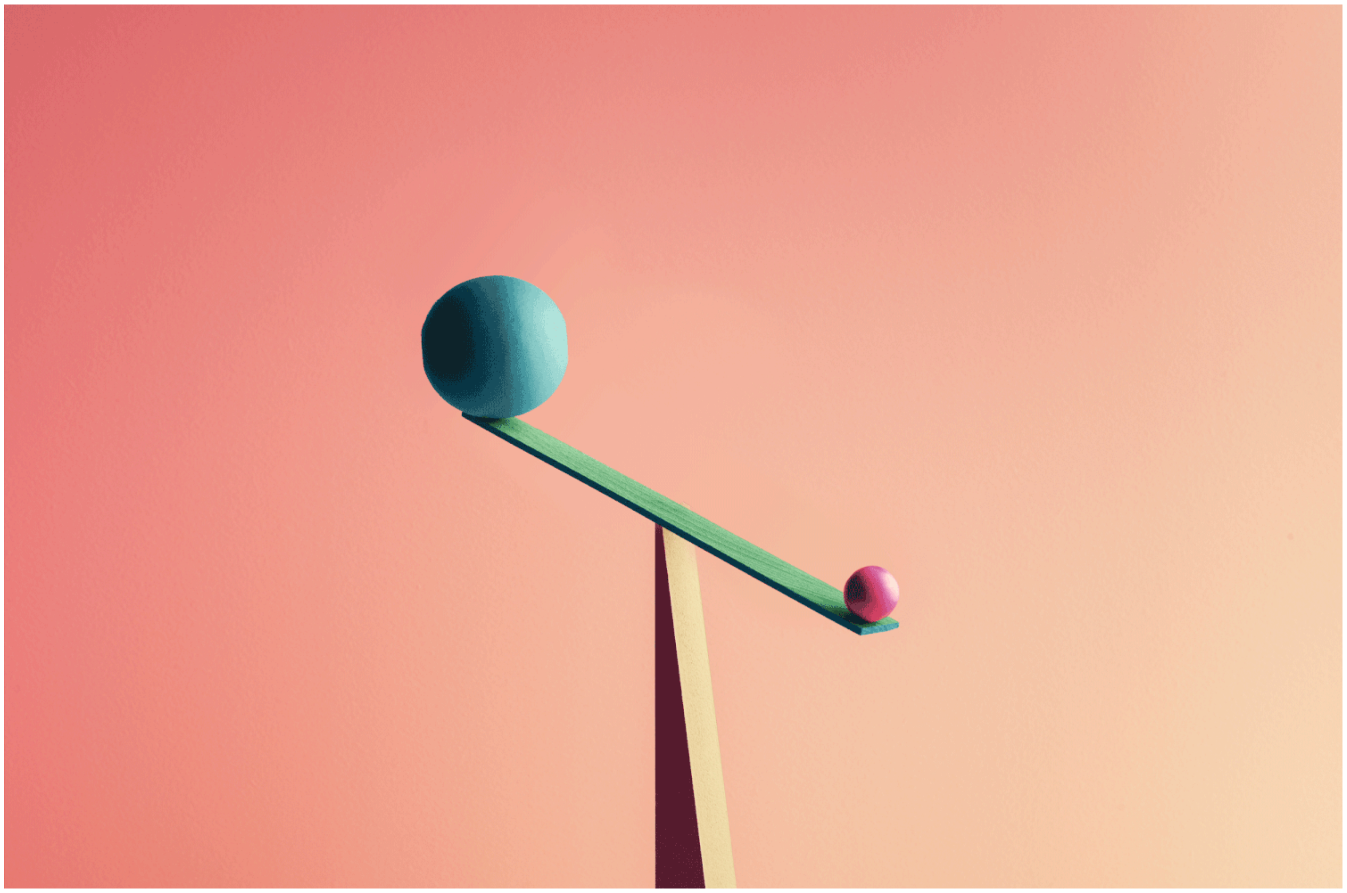
⍰ Let's practice some comparison operators! Here's a problem for you to solve:

▯ 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! ▯▯



In [ ]:

```
# Number of apples in the first and second baskets
basket1 = 12
basket2 = 8

# Check if the number of apples in the first basket is greater than the second basket
print("Is the number of apples in the first basket greater than the number of apples in

# Check if the number of apples in the second basket is less than or equal to the first
print("Is the number of apples in the second basket less than or equal to the number of

# Check if the number of apples in the first basket is equal to the second basket
print("Is the number of apples in the first basket equal to the number of apples in the

# Check if the number of apples in the second basket is not equal to the first basket
print("Is the number of apples in the second basket not equal to the number of apples in
```

```
Is the number of apples in the first basket greater than the number of apples in the sec
ond basket: True
Is the number of apples in the second basket less than or equal to the number of apples
in the first basket: True
Is the number of apples in the first basket equal to the number of apples in the second
basket: False
Is the number of apples in the second basket not equal to the number of apples in the fi
rst basket: 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!

In [ ]:
```python
# Define the weight of your bags
bag1_weight = 15
bag2_weight = 5

# Check if both bags weigh less than 13 kg
if bag1_weight < 13 and bag2_weight < 13:
    print("Enjoy your trip!")
```

```
else:
    print("Sorry, you cannot go through!")
```

```
Sorry, you cannot go through!
```

□ 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!

In [ ]:

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

# Check if at least one bag weighs less than 6 kg
if bag1_weight<6 or bag2_weight<6:
  print("Enjoy your trip!")
else:
  print("Sorry, You can not go through!")
```

```
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. □

In [ ]:
```python
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",
]
```

In [ ]:
```python
# Initialize the count
count = 0

# Iterate through each book and check if it contains the word "Python"
for book in books:
    if "Python" in book:
        count += 1

# Print the count
print("Number of books containing the word 'Python':", count)
```

```
Number of books containing the word 'Python': 2
```

In-Built Functions & Methods

Here is a string for you to work with:

```
In [ ]: # Here is the statement.

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

```
In [ ]: print(statement)
```

```
anas eagerly recommended 'the shawshank redemption' to his friends. He described the mov
ie as a captivating and emotional story of hope and friendship. His friends decided to w
atch it together the following weekend.
```

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

```
In [ ]: # 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 described the
movie as a captivating and emotional story of hope and friendship', 'His friends decided
to watch it together the following weekend.']
```

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

```
In [ ]: # The name 'anas' is spelled wrong.
        # Replace it with 'Anas'.

        statement= statement.replace('anas','Anas')  #Redefine the statement, with the correctio
        print(statement)
```

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

```
ie as a captivating and emotional story of hope and friendship. His friends decided to w
atch it together the following weekend.
```

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

In [ ]:
```python
# Make corrections here.
statement= statement.replace('the shawshank redemption','the shawshank redemption'.title
print(statement)
```

```
Anas eagerly recommended 'The Shawshank Redemption' to his friends. He described the mov
ie as a captivating and emotional story of hope and friendship. His friends decided to w
atch it together the following weekend.
```

1. 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.

In [ ]:
```python
# Take input of movie
new_movie=input("Which movie wold you recommend? ")

# Change the recommendation of movie to one, of your choice.
statement= statement.replace('The Shawshank Redemption',new_movie)
print(statement)
```

```
Which movie wold you recommend? Casper
Anas eagerly recommended 'Casper' to his friends. He described the movie as a captivatin
g and emotional story of hope and friendship. His friends decided to watch it together t
he following weekend.
```

□ 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 □.

In [ ]:
```python
# Start with an empty list
movie_collection = []

# Add some movies to the collection using the append() method
movie_collection.append("Casper")  # Use your favourite movie names
movie_collection.append("Spiderman")
movie_collection.append("The Dark Knight")
movie_collection.append("Titanic")
movie_collection.append("Star Wars")

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

# Remove a movie from the collection using the pop() method
# Remove the movie at index 3 (which is the fourth movie in the list)
movie_collection.pop(3)
print("After removing movie at index 3:", movie_collection)

# Count how many movies are in the collection using the len() function
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)
```

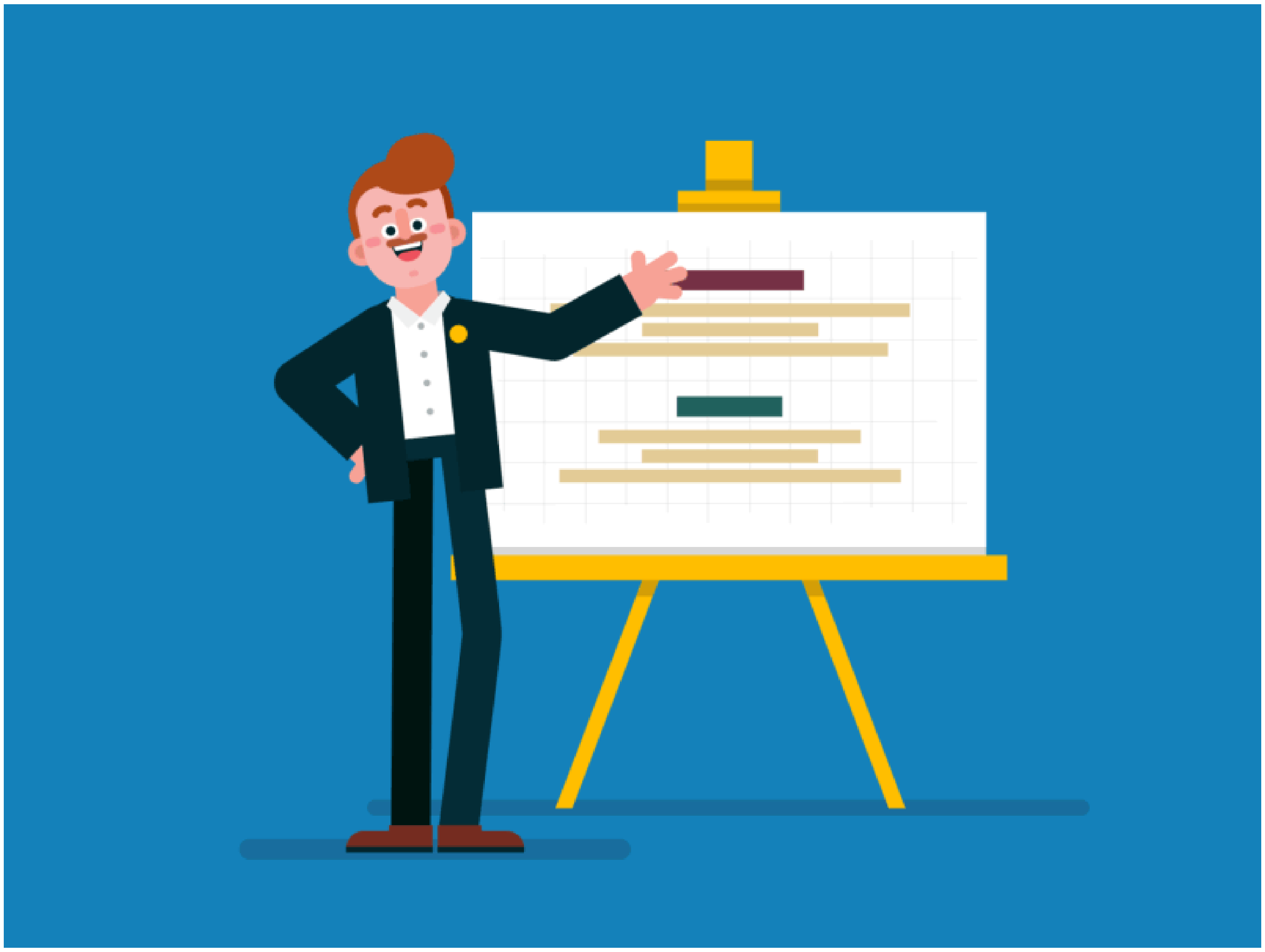
```
My Movie Collection: ['Casper', 'Spiderman', 'The Dark Knight', 'Titanic', 'Star Wars']
After removing movie at index 3: ['Casper', 'Spiderman', 'The Dark Knight', 'Star Wars']
Number of movies: 4
Reversed movie collection: ['Star Wars', 'The Dark Knight', 'Spiderman', 'Casper']
```

?? 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.



In [ ]:
```
student_tuple=('Adam','Alice','Ben','Bilal','Bharath')
```

In [ ]:
```
# Take input from the user and display the role number as the index of the input + 1
student = input("Input the student name to get the roll number - ").title()

# Check if the student is in the tuple before trying to get the index
if student in student_tuple:
    print(f"{student}'s roll number is: {student_tuple.index(student) + 1}")
else:
    print("Student not found in the tuple.")
```

```
Input the student name to get the roll number - Alice
Alice's roll number is: 2
```

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. ▯

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

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

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

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

# Find fruits unique to each basket
unique_basket1 = basket1 - basket2
unique_basket2 = basket2 - basket1
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: {'pineapple', 'apple', 'orange', 'grape', 'banana', 'mango'}
Common fruits in both baskets: {'banana', 'orange'}
Fruits unique to basket 1: {'apple', 'grape'}
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".

In [ ]:
```python
# Get the lead content of the packaged food material
lead_content = float(input("Please enter the lead content of the packaged food material

# Check if the lead content exceeds 5%
if lead_content > 5:
    print("This batch cannot be approved.")
else:
    print("This batch is approved.")
```

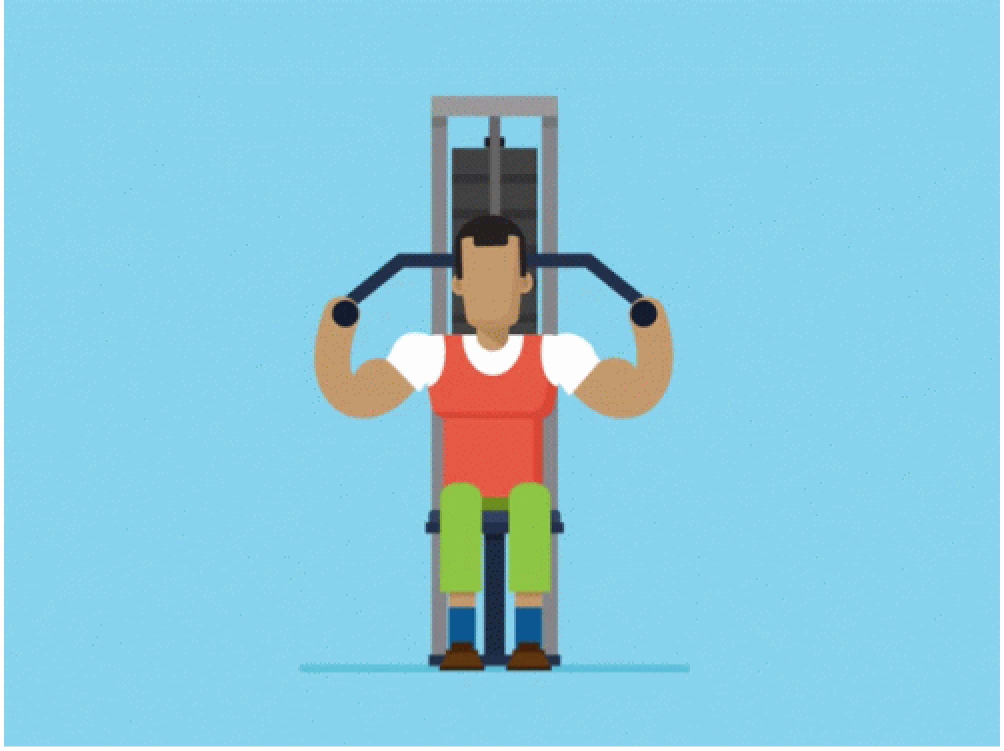
```
Please enter the lead content of the packaged food material in percentage: 9
This batch cannot be approved.
```

▯ 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!".

In [ ]:
```python
# Get the age of the guest
age = int(input("Enter the age: "))

# Check the age of the guest
if age < 18:
    print("Sorry, you cannot enter!")
else:
    print("Welcome! You are allowed to enter.")
```

```
Enter the age: 16
Sorry, you cannot 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!"



In [ ]:
```python
# Get the client's weight loss
client_weight_loss = float(input("Please enter the client's weight loss (in pounds): "))

# Check if the client reached their weight loss goal
if client_weight_loss >= 10:
    print("Congratulations, you have reached your weight loss goal!")
else:
    print("Sorry, you have not reached your weight loss goal yet. Keep up the good work!
```

```
Please enter the client's weight loss (in pounds): 6
Sorry, you have not reached your weight loss goal yet. Keep up the good work!
```

? 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."

In [ ]:
```python
# Take input of moisture level
moisture_level = float(input("Enter moisture level in percentage: "))

#Check if moisture level is satisfactory or not
if moisture_level< 40:
  print("Water the plant!")
else:
  print("The plant doesn't need watering.")
```

```
Enter moisture level in percentage: 30
Water the plant!
```

??? 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 □□

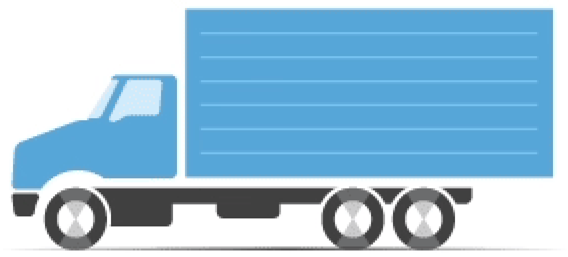
□ Domestic shipping within the country :

- 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? □□



In [ ]:
```python
# Get the package weight and destination from the user
weight = float(input("Please enter the weight of the package (in kg): "))
destination = input("Please enter the destination (domestic/international): ").lower()

# Initialize shipping cost with a default value
cost = 0

# Calculate the shipping cost based on the weight and destination

# If Domestic, check weight and calculate shipping charges accordingly.
if destination == 'domestic':
    if weight <= 1:
        cost = 50
    else:
        cost = 50 + (weight - 1) * 10

# If international, check weight and calculate shipping charges accordingly.
elif destination == 'international':
    if weight <= 1:
        cost = 500
    else:
        cost = 500 + (weight - 1) * 100

# Handle cases where the destination is neither domestic nor international
else:
    print("Invalid destination. Please enter either 'domestic' or 'international'.")

# Print the shipping cost to the user if a valid destination is provided
if cost > 0:
    print("The shipping cost for a package weighing", weight, "kg to", destination, "is
```

```
Please enter the weight of the package (in kg): 40
Please enter the destination (domestic/international): domestic
The shipping cost for a package weighing 40.0 kg to domestic is Rs. 440.0
```

□ 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%.

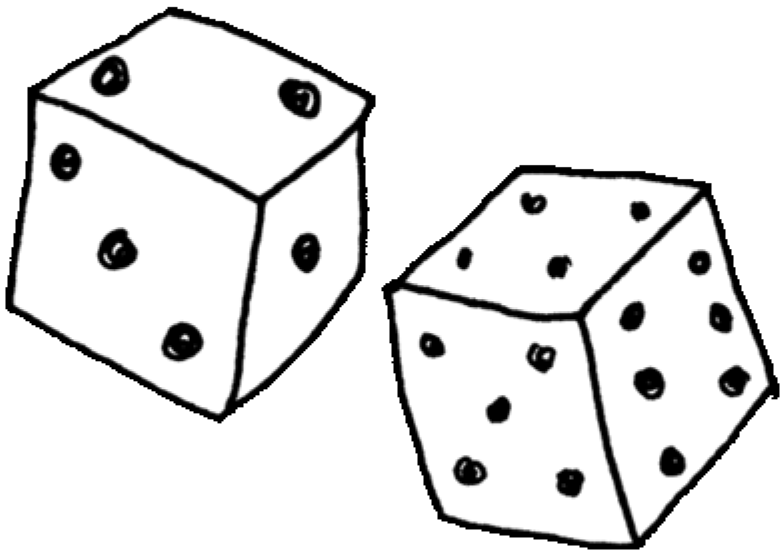
Are you ready to make your customers happy with these discounts? □□

In [ ]:
```python
# Ask the user for the car price
car_price = float(input("Enter the car price in Rs.: "))

# Check if the car price is less than Rs. 10,00,000
if car_price < 1000000:
    discounted_price = car_price
    print("No Discount")
# Check if the car price is between Rs. 10,00,000 and Rs. 20,00,000, with a 5% discount
elif car_price < 2000000:
    discounted_price = car_price * 0.95
    print("5% Discount")
# Check if the car price is between Rs. 20,00,000 and Rs. 30,00,000, with a 7.5% discoun
elif car_price < 3000000:
    discounted_price = car_price * 0.925
    print("7.5% Discount")
# If the car price is greater than Rs. 30,00,000, there is a 10% discount
else:
    discounted_price = car_price * 0.9
    print("10% Discount")

# Print the discounted price with a message to the user
print(f"The discounted price is Rs. {discounted_price:.2f}")
```

```
Enter the car price in Rs.: 1900000
5% Discount
The discounted price is Rs. 1805000.00
```

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!



In [ ]:
```
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
```

In [ ]:
```
# create an empty list to store each player's average roll
average_rolls = []

# loop through each player's rolls
for rolls in player_rolls:
    # calculate the average roll for the player using the sum() and len() functions
    avg_roll = sum(rolls) / len(rolls)

    # add the player's average roll to the list of average rolls
    average_rolls.append(avg_roll)

# loop through each player's average roll and print it out with the player number
for i, avg_roll in enumerate(average_rolls):
    print(f"Player {i+1}'s average roll: {avg_roll}")
```

```
Player 1's average roll: 3.4
Player 2's average roll: 3.8
Player 3's average roll: 4.4
```

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.

In [ ]:
```python
# List of names
names = ["Alice", "Bob", "Eve", "Charlie", "Ivy", "David", "Olivia", "Peter"]

# Initialize lists for names starting with vowels and consonants
vowel_names = []
consonant_names = []

# Iterate through each name in the list
for name in names:
    # Check if the first letter (case-insensitive) is a vowel
    if name[0].upper() in "AEIOU":
        vowel_names.append(name)
    else:
        consonant_names.append(name)

# Print lists of names starting with vowels and consonants
print("Names that start with vowels:", vowel_names)
print("Names that start with consonants:", consonant_names)
```

```
Names that start with vowels: ['Alice', 'Eve', 'Ivy', 'Olivia']
Names that start with 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.

In [5]:
```python
# Define the grades dictionary
grades = {
    'Alice': 85,
    'Bob': 92,
    'Charlie': 88,
    'David': 95,
    'Emily': 78,
    'Frank': 91
}

# Initialize a flag to check if any student scored above 90%
above_90 = False

# Print the names of the students who scored above 90%
print("Names of the students who scored above 90%:")
for student, score in grades.items():
    if score > 90:
        print(student)
        above_90 = True

# Check if no student scored above 90%
if not above_90:
    print("No student scored above 90%.")
```

```
Names of the students who scored above 90%:
Bob
David
Frank
```

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.

In [ ]:
```python
# 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]
]

# Initialize counters for even and odd numbers
even = 0
odd = 0

# Iterate through the nested list of numbers to count even and odd elements
for sublist in numbers:
    for number in sublist:
        # Check if the number is even
        if number % 2 == 0:
            even += 1
        else:
            odd += 1

# Print the counts of even and odd numbers
print(f"Even counts: {even} and odd counts: {odd}")
```

```
Even counts: 12 and odd counts: 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:

In [ ]:
```python
# List of players
players = ['Alice', 'Bob', 'Charlie', 'Diana']

# Iterate through each player's index in the players list
for idx_player1 in range(len(players)):
    # Iterate through the players after the current player to avoid duplicate pairings
    for idx_player2 in range(idx_player1 + 1, len(players)):
        # Print the pairing information
        print(f"Player {players[idx_player1]} plays against Player {players[idx_player2]
```

```
Player Alice plays against Player Bob
Player Alice plays against Player Charlie
Player Alice plays against Player Diana
Player Bob plays against Player Charlie
Player Bob plays against Player Diana
Player Charlie plays against Player Diana
```

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.

In [ ]:
```
# Ask the user for the first number
num1 = float(input("Enter the first number: "))

# Ask the user for the second number
num2 = float(input("Enter the second number: "))

# Ask the user for the operation they want to perform
operation = input("Enter the operation (addition, subtraction, multiplication, division)

# Perform the operation based on the user's input
if operation == "addition":
    result = num1 + num2
    print(f"The result of {num1} {operation} {num2} is {result}")
elif operation == "subtraction":
    result = num1 - num2
    print(f"The result of {num1} {operation} {num2} is {result}")
```

```
elif operation == "multiplication":
    result = num1 * num2
    print(f"The result of {num1} {operation} {num2} is {result}")
elif operation == "division":
    if num2 != 0:  # Check for division by zero
        result = num1 / num2
        print(f"The result of {num1} {operation} {num2} is {result}")
    else:
        print("Error: Division by zero is undefined.")
else:
    print("Invalid operation. Please enter addition, subtraction, multiplication, or div
```

```
Enter the first number: 12
Enter the second number: 3
Enter the operation (addition, subtraction, multiplication, division): division
The result of 12.0 division 3.0 is 4.0
```

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.

In [ ]:
```python
# Import the random module to generate a random number
import random

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

# Set the maximum number of tries
max_tries = 3
tries = 1

# Ask the user to guess the secret number within the specified number of tries
inp = int(input(f"Guess the secret number between 1 and 100. You will have {max_tries} t

# Use a while loop to give the user multiple chances to guess
while tries <= max_tries:
    # Check if the guess is too high
    if inp > secret_number:
        inp = int(input("Your guess was too high, try again: "))
    # Check if the guess is too low
    elif inp < secret_number:
        inp = int(input("Your guess was too low, try again: "))
    # Check if the guess is correct
    elif inp == secret_number:
        print("Congratulations! You guessed right! The secret number was", secret_number
        break

    # Increment the number of tries
    tries += 1

# If no correct guess is made within the allowed tries, print the correct secret number
if inp != secret_number:
    print("No tries left. The secret number was:", secret_number)
```

```
Guess the secret number between 1 and 100. You will have 3 tries: 15
Your guess was too low, try again: 68
Your guess was too low, try again: 98
Your guess was too high, try again: 76
No tries left. The secret number was: 81
```

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.

The Fibonacci Sequence

1,1,2,3,5,8,13,21,34,55,89,144,233,377…

1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21
13+21=34
21+34=55
34+55=89
55+89=144
89+144=233
144+233=377

In [ ]:

```python
# Take user input for the number of terms in the sequence
n = int(input("Enter how many terms you want in the sequence: "))

# Check if the user input is less than 2
if n < 3:
    print("Please enter a number greater than or equal to 3.")
else:
    # Initialize the first two terms of the sequence
    fibonacci_sequence = [0, 1]

    # Use a while loop to generate the remaining terms of the sequence
    while len(fibonacci_sequence) < n:
        fibonacci_sequence.append(fibonacci_sequence[-1] + fibonacci_sequence[-2])

    # Print the generated Fibonacci sequence
    print("Generated Fibonacci sequence:", fibonacci_sequence)
```

```
Enter how many terms you want in the sequence: 6
Generated Fibonacci sequence: [0, 1, 1, 2, 3, 5]
```

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.

In [ ]:
```python
# Input two numbers as the range
start = int(input("Enter the starting number of the range: "))
end = int(input("Enter the ending number of the range: "))

# Input the digit to be counted
digit_to_count = int(input("Enter the digit to be counted: "))

# Initialize counter variable
digit_count = 0

# Use a for loop to iterate over the range of numbers
for number in range(start, end + 1):
    # Use a while loop to check each digit
    temp_number = number
    while temp_number > 0:
        if temp_number % 10 == digit_to_count:
            digit_count += 1
        temp_number //= 10

# Print the total count of the digit
print(f"The digit {digit_to_count} occurs {digit_count} times in the specified range.")
```

```
Enter the starting number of the range: 100
Enter the ending number of the range: 150
Enter the digit to be counted: 5
The digit 5 occurs 6 times in the specified range.
```

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:**

In [ ]:
```
def calculator(num1, num2, operation):
    """
    Perform arithmetic operations on two numbers.

    Parameters:
    - num1 (float): The first number.
    - num2 (float): The second number.
    - operation (str): The arithmetic operation to perform. Valid operations are '+', '-

    Returns:
    - float or str: The result of the arithmetic operation. If the operation is invalid
    """
    # Convert operation to lowercase for case-insensitive comparison
    operation = operation.lower()

    # Error handling for division by zero
    if operation == "/" and num2 == 0:
        return "Error: Division by zero is undefined"

    # Perform the specified arithmetic operation
    if operation == "+":
        result = num1 + num2
    elif operation == "-":
        result = num1 - num2
    elif operation == "*":
        result = num1 * num2
    elif operation == "/":
        result = num1 / num2
    else:
        result = "Invalid operation"

    return result
```

□ **Test Code:**

In [ ]:
```
# Test the calculator function
print(calculator(5, 3, '+'))
print(calculator(5, 3, '-'))
print(calculator(5, 3, '*'))
print(calculator(5, 3, '/'))
print(calculator(5, 3, '%'))
```

```
8
2
15
1.6666666666666667
Invalid operation
```

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.

In [ ]:

```python
import random

def roll_dice():
    """
    Simulate rolling a six-sided die.

    Returns:
    - int: The rolled number.
    """
    return random.randint(1, 6)

# Call the function and print the rolled number
rolled_number = roll_dice()
print(f"The rolled number is {rolled_number}")
```

```
The rolled number is 3
```

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:

In [ ]:

```python
# Define a function to convert Celsius to Fahrenheit
def to_fahrenheit(celsius):
    """
    Convert Celsius to Fahrenheit.

    Parameters:
    - celsius (float): Temperature in Celsius.
    Returns:
    - float: Temperature in Fahrenheit.
    """
    fahrenheit = (celsius * 9/5) + 32
    return fahrenheit
```

□ **Test Code:**

In [ ]:
```
# Example usage
celsius_temperature = 20
result_fahrenheit = to_fahrenheit(celsius_temperature)
print(f"{celsius_temperature} degrees Celsius is equal to {result_fahrenheit} degrees Fa
```

```
20 degrees Celsius is equal to 68.0 degrees Fahrenheit.
```

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:

In [ ]:
```
# define sum_odd custom function
def sum_odd(lst):
    result = 0
    # Use a loop to iterate through the numbers and sum those numbers which are odd
    for num in lst:
        if num % 2 != 0:
            result += num
    return result
```

□ **Test Code:**

In [ ]:
```
# test here
print(sum_odd([1, 2, 3, 4, 5, 6]))
```

```
9
```

□ 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.

In [7]:
```
# Import reduce from functools module
from functools import reduce

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

# Use filter to get only the odd numbers from the list
odd = list(filter(lambda x: x % 2 == 1, numbers))
print(f"Odd numbers in the list: {odd}")

# Use map to square each odd number in the filtered list
square = list(map(lambda y: y**2, odd))
print(f"Squares of all the odd numbers in the list: {square}")

# Use reduce to find the sum of squared odd numbers
sum_of_squares = reduce(lambda x, y: x + y, square)
print(f"Sum of all the squared odd numbers in the list: {sum_of_squares}")
```

```
Odd numbers in the list: [1, 3, 5, 7, 9]
Squares of all the odd numbers in the list: [1, 9, 25, 49, 81]
Sum of all the squared odd numbers in the list: 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.

In [ ]:
```
# Create two lists of toppings, one for pizza and one for burgers
pizza_toppings = ['mushroom', 'olive', 'tomato', 'pepperoni', 'onion', 'garlic']
burger_toppings = ['lettuce', 'cheese', 'mayonnaise', 'bacon', 'pickle', 'avocado']

# Use list comprehension to combine toppings from both lists
combined_toppings = [topping for topping in pizza_toppings + burger_toppings]

# Use a conditional statement to filter toppings with more than 5 characters
filtered_toppings = [topping for topping in combined_toppings if len(topping) > 5]

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

```
Filtered Toppings (more than 5 characters): ['mushroom', 'tomato', 'pepperoni', 'garli
c', 'lettuce', 'cheese', 'mayonnaise', 'pickle', 'avocado']
```

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.

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

# Use zip function to create a list of pairs (country, sales)
country_sales_pairs = list(zip(countries, sales))

# Use list comprehension to create a dictionary with sales greater than 1000
filtered_sales_dict = {country: sales for country, sales in country_sales_pairs if sales

# Print the result
print("Filtered Sales Dictionary (sales > 1000):", filtered_sales_dict)
```

```
Filtered Sales Dictionary (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.

In [ ]:
```python
# 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': -0.02},
    {'name': 'Amazon.com, Inc.', 'ticker': 'AMZN', 'price': 250.0, 'change': 0.1},
    {'name': 'Alphabet Inc.', 'ticker': 'GOOGL', 'price': 110.0, 'change': 0.02},
    {'name': 'Facebook, Inc.', 'ticker': 'FB', 'price': 80.0, 'change': 0.03}
]

# Use list comprehension to filter stocks with price > 100 and positive percentage chang
filtered_stocks = [stock for stock in stocks if stock['price'] > 100 and stock['change']

# Use list comprehension to create a new list of dictionaries with name and price
name_price_list = [{'name': stock['name'], 'price': stock['price']} for stock in stocks]

# Print the results
print("Name-Price List:", name_price_list)
```

```
Name-Price List: [{'name': 'Apple Inc.', 'price': 120.0}, {'name': 'Microsoft Corporatio
n', 'price': 95.0}, {'name': 'Amazon.com, Inc.', 'price': 250.0}, {'name': 'Alphabet In
c.', '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 [].

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

# Create a list of songs with duplicate names
duplicates = [song for song in songs if songs.count(song) > 1]

# Use set comprehension to create a set of unique names excluding those starting with "S
unique_names_set = {song for song in set(songs) if song[0] != 'S' and song[-1] != 'n'}

# Print the results
print("Set of Unique Names (excluding starting with 'S' and ending with 'n'):", unique_n
```

```
Set of Unique Names (excluding starting with 'S' and ending with 'n'): {'Hotel Californi
a', 'Bohemian Rhapsody'}
```

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.

In [ ]:
```
# Define Calculator Class
class Calculator:
    def add(self, num1, num2):
        # Method to add two numbers
        return num1 + num2

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

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

    def divide(self, num1, num2):
        # Method to divide num1 by num2
        if num2 == 0:
            # Check for division by zero
            raise ValueError("Cannot divide by zero")
        return num1 / num2
```

In [ ]:
```
# Create an instance of the Calculator class
```

```
calculator_instance = Calculator()

# Perform arithmetic operations using the Calculator instance
result_add = calculator_instance.add(5, 3)  # Add 5 and 3
result_subtract = calculator_instance.subtract(8, 2)  # Subtract 2 from 8
result_multiply = calculator_instance.multiply(4, 6)  # Multiply 4 and 6
result_divide = calculator_instance.divide(10, 2)  # Divide 10 by 2

# Print the results of each operation
print("Addition Result:", result_add)
print("Subtraction Result:", result_subtract)
print("Multiplication Result:", result_multiply)
print("Division Result:", result_divide)
```

```
Addition Result: 8
Subtraction Result: 6
Multiplication Result: 24
Division Result: 5.0
```

□ **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.

In [ ]:
```
# Create BasketballGame class
class BasketballGame:
    def __init__(self):
        # Initialize the score for both teams to 0
        self.team1_score = 0
        self.team2_score = 0

    def update_score_team1(self, points):
        # Update the score for Team 1
        self.team1_score += points

    def update_score_team2(self, points):
        # Update the score for Team 2
        self.team2_score += points

    def print_current_score(self):
        # Print the current score for both teams
        print(f"Team 1 Score: {self.team1_score}")
        print(f"Team 2 Score: {self.team2_score}")
```

In [ ]:
```
# Example usage
basketball_game = BasketballGame()

# Update the score for each team
basketball_game.update_score_team1(2)
basketball_game.update_score_team2(3)
basketball_game.update_score_team1(1)
basketball_game.update_score_team2(2)

# Print the current score
basketball_game.print_current_score()
```

```
Team 1 Score: 3
Team 2 Score: 5
```

□ **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)

In [ ]:
```
# Define PaintBrush Class
class PaintBrush:
    def __init__(self, size, color, brand, brush_type):
        # Initialize instance variables with provided values
        self.size = size
        self.color = color
        self.brand = brand
        self.brush_type = brush_type

    def paint(self):
        # Display a message indicating the painting action with brush details
        print("Painting with a {0} {1} {2} {3} brush".format(self.size, self.color, self
```

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.

In [ ]:
```
# Test Case
# Create an instance of the PaintBrush class
my_paintbrush = PaintBrush(size="medium", color="red", brand="Winsor & Newton", brush_ty

# Call the paint() method on the instance
my_paintbrush.paint()
```

```
Painting with a medium red Winsor & Newton round brush
```

□ **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.

```
In [ ]: # Define a BankAccount class
        class BankAccount:
            def __init__(self, name, account_number, balance):
                """
                Initialize a BankAccount instance.

                :param name: The name of the account holder.
                :param account_number: The account number associated with the account.
                :param balance: The initial balance of the account.
                """
                self.name = name
                self.account_number = account_number
                self.balance = balance

            def deposit(self, amount):
                """
                Deposit funds into the account.

                :param amount: The amount to be deposited.
                """
                self.balance += amount
                print(f"Deposited {amount} units. New balance: {self.balance}")

            def withdraw(self, amount):
                """
                Withdraw funds from the account.

                :param amount: The amount to be withdrawn.
                """
                if amount <= self.balance:
                    self.balance -= amount
                    print(f"Withdrawn {amount} units. New balance: {self.balance}")
                else:
                    print(f"Insufficient funds. Current balance: {self.balance}")

            def account_details(self):
                """
                Print account details.
                """
                print(f"Account details:\nName: {self.name}\nAccount Number: {self.account_numbe
```

```
In [ ]: # Test Case
        # Create an instance of the BankAccount class
        my_account = BankAccount(name="John Doe", account_number="123456789", balance=1000)

        # Test deposit(), withdraw(), and account_details() methods
        my_account.deposit(500)
```

```python
my_account.withdraw(200)
my_account.account_details()
```

```
Deposited 500 units. New balance: 1500
Withdrawn 200 units. New balance: 1300
Account details:
Name: John Doe
Account Number: 123456789
Balance: 1300
```

□ **Instructions:**

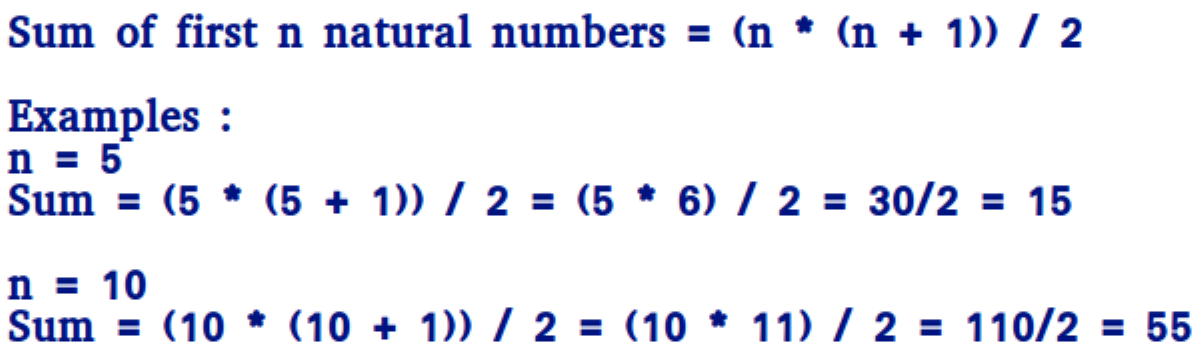
1. Define a BankAccount class with an init() method that takes the name, account_number, and balance as parameters and initializes the instance variables.
2. Define the deposit() method that takes the amount to be deposited as a parameter and adds it to the balance. Also, print a message with the new 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. □



Sum of first n natural numbers = (n * (n + 1)) / 2

Examples :
n = 5
Sum = (5 * (5 + 1)) / 2 = (5 * 6) / 2 = 30/2 = 15

n = 10
Sum = (10 * (10 + 1)) / 2 = (10 * 11) / 2 = 110/2 = 55

In [ ]:

```python
# 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
```

In [ ]:
```
# Write your code here
# Corrected code
n = int(input("Enter a number: "))  # Convert input to an integer
sum = 0
for i in range(1, n+1):  # Change the range starting point to 1
    sum = sum + i
print("The sum of the first", n, "natural numbers is", sum)
```

```
Enter a number: 5
The sum of the first 5 natural numbers is 15
```

▯ 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. ▯
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.

In [ ]:
```
# 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]

# Calculate the sum of all temperatures
for temperature in temperatures_w1:
  sum = sum + temperature

# Calculate the average temperature
average = sum / 7

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

In [ ]:
```
# Write your code here
# 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]

# Calculate the sum and average of temperatures for week 1
sum_w1 = 0
for temperature in temperatures_w1:
    sum_w1 = sum_w1 + temperature

average_w1 = sum_w1 / len(temperatures_w1)

# Output the result for week 1
print("The average temperature of week 1 is", average_w1, "degrees Celsius.")
```

```
# Calculate the sum and average of temperatures for week 2
sum_w2 = 0
for temperature in temperatures_w2:
    sum_w2 = sum_w2 + temperature

average_w2 = sum_w2 / len(temperatures_w2)

# Output the result for week 2
print("The average temperature of week 2 is", average_w2, "degrees Celsius.")
```

```
The average temperature of week 1 is 31.0 degrees Celsius.
The average temperature of week 2 is 31.4 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.

In [ ]:
```
# Code without exception handling
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

result = num1 / num2

print("The result is: ", result)
```

In [ ]:
```
# Write your code here
try:
    # Attempt to get two integers from the user
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))

    # Perform the division operation
    result = num1 / num2

    # Display the result
    print("The result is:", result)

# Handle the case where the user enters a non-integer value
except ValueError as ve:
    print(f"Error: {ve}. Please enter valid integers.")

# Handle the case where the user attempts to divide by zero
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")

# Handle any other unexpected errors
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
Enter the first number: 10
Enter the second number: 0
Error: Division by zero is not allowed.
```

□ **Instructions:**

1. Add a try-except block to handle any errors that may occur during runtime.
2. In the try block, add the code that may raise an exception.
3. In the except block, add the code to handle the exception.

What's the time? ??

Create a class Time which takes two inputs: hours and minutes to instantiate.

- Construct a method DisplayTime() which displays the time in AM/PM formats.
  - For example if the input is 14 hours and 45 mins, then this method will print "The time is 2:45 PM".
  - If the inputted hours exceeds 23 then print the message "The input hours should be less than 24" and if the inputted minutes exceeds 59 then print the message "The input minutes should be less than 60."
  - Also if the input is 12 hours 30 minutes, then the displayed time would be 12:30 PM
- Construct a method DisplayRatio() which should display the ratio of minutes to hours.
  - For example, (8 hours and 16 mins) should display 2. Use try, except block to account for ZeroDivisionError.

In [10]:
```python
# Write your code here
# Define the Time class
class Time:
    # Constructor to initialize hours and minutes
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes

    # Method to display time in AM/PM format
    def DisplayTime(self):
        try:
            # Check if input values are within valid range
            if self.hours >= 24 or self.minutes >= 60:
                raise ValueError("Invalid input. Hours should be less than 24 and minute

            # Convert to 12-hour format
            if self.hours >= 12:
                period = "PM"
                if self.hours > 12:
                    self.hours -= 12
            else:
                period = "AM"
                if self.hours == 0:
                    self.hours = 12

            # Display the formatted time
            print(f"The time is {self.hours}:{self.minutes:02d} {period}")

        except ValueError as ve:
            print(f"Error: {ve}")

    # Method to display the ratio of minutes to hours
    def DisplayRatio(self):
        try:
            # Check for division by zero
            if self.hours == 0:
                raise ZeroDivisionError("Cannot divide by zero. Hours should be greater

            # Calculate and display the ratio
            ratio = self.minutes / self.hours
            print(f"The ratio of minutes to hours is {ratio:.2f}")

        except ZeroDivisionError as zde:
            print(f"Error: {zde}")
```

□ **Test Code:**

In [12]:

```python
# Check for few sample inputs of hours and mins
hour_min_list = [(23, 45), (34, 50), (12, 34), (14, 67), (19, 20), (2, 15), (0, 10), (12

# Instantiate and test the Time class
for hours, minutes in hour_min_list:
    time_instance = Time(hours, minutes)
    time_instance.DisplayTime()
    time_instance.DisplayRatio()
    print("=" * 30)
```

```
The time is 11:45 PM
The ratio of minutes to hours is 4.09
==============================
Error: Invalid input. Hours should be less than 24 and minutes should be less than 60.
The ratio of minutes to hours is 1.47
==============================
The time is 12:34 PM
The ratio of minutes to hours is 2.83
==============================
Error: Invalid input. Hours should be less than 24 and minutes should be less than 60.
The ratio of minutes to hours is 4.79
==============================
The time is 7:20 PM
The ratio of minutes to hours is 2.86
==============================
The time is 2:15 AM
The ratio of minutes to hours is 7.50
==============================
The time is 12:10 AM
The ratio of minutes to hours is 0.83
==============================
The time is 12:30 PM
The ratio of minutes to hours is 2.50
==============================
The time is 8:16 AM
The ratio of minutes to hours is 2.00
==============================
```

□ Instructions:

1. Create a class Time which takes two inputs: hours and minutes.
2. Instantiate the values using a function.
3. Then, define two functions for displaying the time in the format explained in the question and for displaying the ratio of hours to minutes.
4. Use try-except block for displaying the ratio to account for Zero Division Error.