

### **Qno: 01**

Write a program that simulates how websites ensure that everyone has a unique username. • Make a list of five or more usernames called `current_users`. • Make another list of five usernames called `new_users`. Make sure one or two of the new usernames are also in the `current_users` list. • Loop through the `new_users` list to see if each new username has already been used. If it has, print a message that the person will need to enter a new username. If a username has not been used, print a message saying that the username is available. • Make sure your comparison is case sensitive. If 'John' has been used, 'JOHN' should not be accepted. (To do this, you'll need to make a copy of `current_users` containing the lowercase versions of all existing users.)

### **Qno: 02**

Make a dictionary called `cities`. Use the names of three cities as keys in your dictionary. Create a dictionary of information about each city and include the country that the city is in, its approximate population, and one fact about that city. The keys for each city's dictionary should be something like `country`, `population`, and `fact`. Print the name of each city and all of the information you have stored about it.

### **Qno: 03**

A **college library** maintains a record of **book ID numbers** using a tuple in Python to keep the data secure and unchanged by default. First, the librarian enters the total number of books and inputs each book ID, and the system stores these values in a tuple and displays the original list along with their index positions. Since tuples are immutable, the program converts the tuple into a list to perform modifications. When new books arrive, a new book ID is **inserted at a specific index position** (for example, index 2), and the system displays the updated tuple along with the index where the insertion occurred. If some books are removed due to damage or outdated editions, a book ID is **deleted from a given index position** (such as index 4), and the system shows the updated tuple and the index from which the value was removed. Additionally, if an incorrect book ID is detected, the librarian **updates the book ID at a particular index** (for example, index 1), and the program displays the modified tuple while clearly mentioning the index where the updation took place. This approach helps the library manage book records efficiently while keeping track of all changes using index values.

### **Qno: 04**

A **school is organizing a fun quiz competition** and wants to select **students randomly** for different activities. The Python program uses the **random module** to make selections fair and unpredictable. First, the teacher enters the names of all participating students. The system uses `random.choice()` to select a **single student randomly** to answer a question. To randomize the order of students for games or presentations, the program uses `random.shuffle()` on the list of student names. For some mini-games, the system generates **random numbers** using `random.randint()` to simulate dice rolls or points scoring. This ensures the quiz and games are **exciting, fair, and unpredictable**, giving every student an equal chance to participate.

### **Qno: 05**

A college wants to manage and format its course information using a Python program. The system allows the user to enter the **course name** and **department name** and displays them in a readable format. The program applies the **title()** function to the course name so that the first letter of each word is capitalized, and the **capitalize()** function to the department name to ensure only the first letter is uppercase. If the course name contains a mistake, the program allows the user to enter an **old name** and a **new name**, and uses the **replace()** function to update the course name. Additionally, the program extracts specific parts of the course name using **string slicing with index values**: the first two characters (index 0 to 1) are extracted as the **course code**, characters from index 1 to 4 are extracted as a **verification part**, and the last three characters (using negative indexing, -3 to end) are extracted as the **course ID**. Finally, the program displays the updated course name along with the course code, course ID, and verification part.

### **Qno: 06**

A **school administration system** uses a Python program to manage **student records** using **file handling**. When a new academic session starts, the system creates a text file named `students.txt` to store student details such as roll number, name, and class. The administrator enters student information, and the program opens the file in **write mode** to save these details permanently. Whenever the administrator wants to view the stored records, the file is opened in **read mode**, and all student data is displayed on the screen. If new students are admitted later, the program opens the same file in **append mode** so that new records are added without deleting the existing data.

### **Qno: 07**

A group of friends wants to split the total bill. The program asks the user to enter the **total bill amount** and the **number of people**, separated by a comma. It then calculates how much each person should pay. The program handles errors like dividing by zero (if number of people is 0), missing commas, or invalid input. After successful calculation, it confirms that everything went well.

### **Qno: 08**

A **student marks system** ensures that the marks entered are valid (between 0 and 100). If the user enters invalid marks, the program stops immediately using **assert**