Name : SK Shivaanee
Register number : 2310257
Section : S02

Assignment 9: Programs using Dictionaries

Aim: You will be able to apply computational thinking to devise solutions using Dictionaries.
● You code Python programs by defining the functions to handle Dictionaries.

Question 1: . Develop a program that simulates a simple banking system. The program must use a dictionary to store information about each account. (CO3, K3, 1.3.1,1.4.1, 2.1.2, 4.1.2) The dictionary should have the following keys: ● "account number" (a unique identifier for each account) ● "balance" (the current balance of the account) ● "transactions" (a list of all transactions that have occurred on this account) The program should allow the user: ● To create a new account by entering a unique account number and an initial balance of Rs.5000/. ● To deposit money into an existing account by entering the account number and the amount to be deposited. ● To withdraw money from an existing account by entering the account number and the amount to be withdrawn. ● To display the updated balance after each transaction (deposit/withdraw). ● To view their account balances at any time. ● To print out a statement of all transactions for a given account.

Code:

```
class SimpleBankingSystem:
    def __init__(self):
        self.accounts = {}

    def create_account(self, account_number, initial_balance=5000):
        if account_number not in self.accounts:
            self.accounts[account_number] = {'balance': initial_balance,
                'transactions': []}
            print(f"Account {account_number} created with an initial balance of
                Rs.{initial_balance}/-")
        else:
            print("Account already exists!")

    def update_balance(self, account_number, amount, transaction_type):
        if transaction_type == 'Deposit':
            self.accounts[account_number]['balance'] += amount
            self.accounts[account_number]['transactions'].append(('deposit',
                amount))
            print(f"Deposit of Rs.{amount}/- successful. Updated balance: Rs
                .{self.accounts[account_number]['balance']}/-")
        elif transaction_type == 'Withdraw':
            if self.accounts[account_number]['balance'] >= amount:
                self.accounts[account_number]['balance'] -= amount
                self.accounts[account_number]['transactions'].append
                    (('withdraw', amount))
                print(f"Withdrawal of Rs.{amount}/- successful. Updated balance
```

```
                    : Rs.{self.accounts[account_number]['balance']}/-")
            else:
                print("Amount NOT available - Maintain sufficient balance")

    def get_balance(self, account_number):
        return self.accounts.get(account_number, {}).get('balance', None)

    def view_transactions(self, account_number):
        transactions = self.accounts.get(account_number, {}).get('transactions'
            , [])
        if transactions:
            print(f"Transaction history for account {account_number}:")
            for transaction in transactions:
                print(f"{transaction[0]}: Rs.{transaction[1]}/-")
        else:
            print("No transactions found for this account.")

# Example usage
bank = SimpleBankingSystem()

# Create accounts
bank.create_account('1234')
bank.create_account('5678', initial_balance=100)
```

```
# Create accounts
bank.create_account('1234')
bank.create_account('5678', initial_balance=100)

# Deposit and withdraw
bank.update_balance('1234', 2000, 'Deposit')
bank.update_balance('5678', 50, 'Withdraw')

# View balances and transactions
print("Balance for account 1234:", bank.get_balance('1234'))
print("Balance for account 5678:", bank.get_balance('5678'))

bank.view_transactions('1234')
bank.view_transactions('5678')
```

Output:

```
Account 1234 created with an initial balance of Rs.5000/-
Account 5678 created with an initial balance of Rs.100/-
Deposit of Rs.2000/- successful. Updated balance: Rs.7000/-
Withdrawal of Rs.50/- successful. Updated balance: Rs.50/-
Balance for account 1234: 7000
Balance for account 5678: 50
Transaction history for account 1234:
deposit: Rs.2000/-
Transaction history for account 5678:
withdraw: Rs.50/-
```

Question 2: Create a nested dictionary to store information of 5 students as follows Eg: student = {Roll no1: { 'Name': 'Kumar', 'Sub1_mark': 75, 'Sub2_mark':76 'Percentage': 75.5}, {Roll no 2: { …}, … } a. Write a function by passing the student dictionary as a parameter to update the dictionary with additional values namely Total, Average and Grades and return the updated dictionary. b. Write a function to pass the updated student dictionary as a parameter and return the student who scored 1st rank based on the Average.

Code:

```python
def update_student_info(student_dict):
    updated_student_dict = {}
    for roll_no, details in student_dict.items():
        total_marks = details['Sub1_mark'] + details['Sub2_mark']
        average = total_marks / 2
        grades = get_grades(average)

        updated_details = {
            'Name': details['Name'],
            'Sub1_mark': details['Sub1_mark'],
            'Sub2_mark': details['Sub2_mark'],
            'Total': total_marks,
            'Average': average,
            'Grades': grades
        }

        updated_student_dict[roll_no] = updated_details

    return updated_student_dict

def get_grades(average):
    if average >= 90:
        return 'A'
    elif 80 <= average < 90:
        return 'B'
    elif 70 <= average < 80:
```

```
        return 'C'
    elif 60 <= average < 70:
        return 'D'
    else:
        return 'F'

def get_top_scorer(student_dict):
    top_scorer = max(student_dict.items(), key=lambda x: x[1]['Average'])
    return top_scorer[0], top_scorer[1]['Name']

# Example usage
students = {
    'Roll no 1': {'Name': 'Kumar', 'Sub1_mark': 75, 'Sub2_mark': 76},
    'Roll no 2': {'Name': 'John', 'Sub1_mark': 85, 'Sub2_mark': 92},
    'Roll no 3': {'Name': 'Alice', 'Sub1_mark': 90, 'Sub2_mark': 88},
    'Roll no 4': {'Name': 'Bob', 'Sub1_mark': 78, 'Sub2_mark': 80},
    'Roll no 5': {'Name': 'Eva', 'Sub1_mark': 92, 'Sub2_mark': 95},
}

updated_students = update_student_info(students)
print("Updated Student Dictionary:")
print(updated_students)

top_scorer_roll_no, top_scorer_name = get_top_scorer(updated_students)
print(f"\nTop Scorer: {top_scorer_name} (Roll no {top_scorer_roll_no})")
```

Output:

```
Updated Student Dictionary:
{'Roll no 1': {'Name': 'Kumar', 'Sub1_mark': 75, 'Sub2_mark': 76, 'Total': 151,
    'Average': 75.5, 'Grades': 'C'}, 'Roll no 2': {'Name': 'John', 'Sub1_mark': 85,
    'Sub2_mark': 92, 'Total': 177, 'Average': 88.5, 'Grades': 'B'}, 'Roll no 3':
    {'Name': 'Alice', 'Sub1_mark': 90, 'Sub2_mark': 88, 'Total': 178, 'Average': 89.0,
    'Grades': 'B'}, 'Roll no 4': {'Name': 'Bob', 'Sub1_mark': 78, 'Sub2_mark': 80,
    'Total': 158, 'Average': 79.0, 'Grades': 'C'}, 'Roll no 5': {'Name': 'Eva',
    'Sub1_mark': 92, 'Sub2_mark': 95, 'Total': 187, 'Average': 93.5, 'Grades': 'A'}}

Top Scorer: Eva (Roll no Roll no 5)
>
```

Additional practice problems:

Question 1: Consider a dictionary with product name as a key and quantity and unit price as values. Write a function to pass this dictionary as a parameter and return the total price of all the products.

Code:

```
def calculate_total_price(product_dict):
    total_price = 0

    for product, details in product_dict.items():
        quantity, unit_price = details
        total_price += quantity * unit_price

    return total_price

# Example usage
products = {
    'Product1': (5, 10.50),  # Quantity = 5, Unit Price = Rs. 10.50
    'Product2': (2, 5.75),   # Quantity = 2, Unit Price = Rs. 5.75
    'Product3': (8, 3.00),   # Quantity = 8, Unit Price = Rs. 3.00
}

total_price = calculate_total_price(products)
print(f"Total Price of all products: Rs. {total_price:.2f}/-")
```

Output:

```
Total Price of all products: Rs. 88.00/-
>
```

Question 2: Create a dictionary to maintain capitals of different countries. Key has to be the country name. Store the country names in the dictionaries in the sorted order. Also, delete a particular country from the dictionary.

Code:

```
def create_sorted_capitals_dict():
    capitals = {
        'India': 'New Delhi',
        'USA': 'Washington, D.C.',
        'Canada': 'Ottawa',
        'Australia': 'Canberra',
        'Japan': 'Tokyo',
    }

    sorted_capitals = dict(sorted(capitals.items()))  # Sort the dictionary by
        keys
    return sorted_capitals

def delete_country(capitals_dict, country_name):
    if country_name in capitals_dict:
        del capitals_dict[country_name]
        print(f"{country_name} deleted from the dictionary.")
    else:
        print(f"{country_name} not found in the dictionary.")

# Example usage
sorted_capitals_dict = create_sorted_capitals_dict()
print("Sorted Capitals Dictionary:")
print(sorted_capitals_dict)
```

```
country_to_delete = 'Canada'
delete_country(sorted_capitals_dict, country_to_delete)

print(f"\nAfter deleting {country_to_delete}:")
print(sorted_capitals_dict)
```

Output:

```
Sorted Capitals Dictionary:
{'Australia': 'Canberra', 'Canada': 'Ottawa', 'India': 'New Delhi', 'Japan': 'Tokyo',
    'USA': 'Washington, D.C.'}
Canada deleted from the dictionary.
After deleting Canada:{'Australia': 'Canberra', 'India': 'New Delhi', 'Japan': 'Tokyo'
    , 'USA': 'Washington, D.C.'}
```

Question 3: Given a sentence. Create a histogram of all the alphabets as a dictionary where alphabet is a key and number of occurrences of the alphabet are values.

Code:

```
def create_alphabet_histogram(sentence):
    # Initialize an empty dictionary to store the histogram
    alphabet_histogram = {}

    # Iterate through each character in the sentence
    for char in sentence:
        # Check if the character is an alphabet
        if char.isalpha():
            # Convert the character to lowercase to make the histogram case
                -insensitive
            char = char.lower()

            # Update the histogram dictionary
            if char in alphabet_histogram:
                alphabet_histogram[char] += 1
            else:
                alphabet_histogram[char] = 1

    return alphabet_histogram

# Example usage
input_sentence = "This is a sample sentence with some alphabets."
histogram = create_alphabet_histogram(input_sentence)

# Display the histogram
```

```
print("Alphabet Histogram:")
for alphabet, count in sorted(histogram.items()):
    print(f"{alphabet}: {count}")
```

Output:

```
Alphabet Histogram:
a: 4
b: 1
c: 1
e: 6
h: 3
i: 3
l: 2
m: 2
n: 2
o: 1
p: 2
s: 6
t: 4
w: 1
```

Question 4: Consider two lists: movie_name and release_year. Construct a dictionary where movie name is the key and year of release as a value from the two lists. Write function to search for the movies released during the given year. (Hint: use reverse lookup).

Code:

```
def create_movie_dictionary(movie_names, release_years):
    movie_dict = dict(zip(movie_names, release_years))
    return movie_dict

def search_movies_by_year(movie_dict, search_year):
    matching_movies = [movie for movie, year in movie_dict.items() if year ==
        search_year]
    return matching_movies

# Example usage
movie_names = ["Movie1", "Movie2", "Movie3", "Movie4", "Movie5"]
release_years = [2005, 2010, 2005, 2015, 2010]

movies_dictionary = create_movie_dictionary(movie_names, release_years)

# Display the created dictionary
print("Movies Dictionary:")
for movie, year in movies_dictionary.items():
    print(f"{movie}: {year}")

# Search for movies released in a specific year
searched_year = 2010
matching_movies = search_movies_by_year(movies_dictionary, searched_year)

print(f"\nMovies released in {searched_year}:")
```

```
if matching_movies:
    for movie in matching_movies:
        print(movie)
else:
    print("No movies found for the given year.")
```

Output:

```
Movies Dictionary:
Movie1: 2005
Movie2: 2010
Movie3: 2005
Movie4: 2015
Movie5: 2010

Movies released in 2010:
Movie2
Movie5
```

Question 5: Consider a dictionary maintaining a friends network. i.e a person has a set of friends. friends={‘Kumar’:[‘Ramesh’,’Gopal’,’Vimal’], ‘Ramesh’:[‘Gopal’,’Vinay’,’Kumar’, ’Ragu’], ‘Vinay’:[..], ...} Find person names who are friends of more than one person

Code:

```
def find_mutual_friends(friends_dict):
    mutual_friends = {}

    # Count the occurrences of each friend in the network
    for person, friend_list in friends_dict.items():
        for friend in friend_list:
            if friend in mutual_friends:
                mutual_friends[friend] += 1
            else:
                mutual_friends[friend] = 1

    # Filter persons who are friends with more than one person
    result = [person for person, count in mutual_friends.items() if count > 1]

    return result

# Example usage
friends = {
    'Kumar': ['Ramesh', 'Gopal', 'Vimal'],
    'Ramesh': ['Gopal', 'Vinay', 'Kumar', 'Ragu'],
    'Vinay': ['Gopal', 'Ramesh'],
    'Gopal': ['Ramesh', 'Kumar', 'Vinay'],
    'Ragu': ['Ramesh'],
    'Vimal': ['Kumar']
}

mutual_friends_result = find_mutual_friends(friends)
print("Persons who are friends with more than one person:")
print(mutual_friends_result)
```

Output:

```
Persons who are friends with more than one person:
['Ramesh', 'Gopal', 'Vinay', 'Kumar']
```

Learning outcome:

1. Reading inputs / Printing the result
2. Using appropriate datatypes for the given input
3. Variable assignment
4. Converting the formula into python expressions

Result: Thus I learned to implement a simple problems in Python and solve the same using Strings.