ASSIGNMENT 2

Name : Karan Dave
Roll No : 06
Subject : Introduction to Python Programming

1) Display the difference in dates

```python
from datetime import datetime

def calculate_date_difference(start_date_str, end_date_str):
    # Define the date format
    date_format = "%Y-%m-%d"

    # Convert date strings to datetime objects
    start_date = datetime.strptime(start_date_str, date_format)
    end_date = datetime.strptime(end_date_str, date_format)

    # Calculate the difference
    difference = end_date - start_date

    # Print the results
    print(f"Start Date: {start_date_str}")
    print(f"End Date: {end_date_str}")
    print(f"¥nTime Difference:")
    print(f"Total Days: {difference.days}")
    print(f"Total Seconds: {difference.total_seconds()}")

# Example Usage
calculate_date_difference("2023-01-01", "2024-03-15")
```

Start Date: 2023-01-01

End Date: 2024-03-15

Time Difference:

Total Days: 439

Total Seconds: 37929600.0

2) Display time since epoch in hours and minutes

```python
import time

def epoch():
    # Get total seconds since the epoch
    seconds = time.time()

    # Calculate hours and remaining minutes
    hours = int(seconds / 3600)
    minutes = int((seconds % 3600) / 60)

    print(f"Total seconds since epoch: {seconds}")
    print(f"Time since epoch: {hours} hours and {minutes} minutes")

# Run the function
epoch()
```

Output

Total seconds since epoch: 1757875079.826191

Time since epoch: 488298 hours and 37 minutes

3) Display your age in years, months and days

```
from datetime import date, datetime

def get_age(birth_date_str):
    # Get today's date
    today = date.today()

    # Convert the birth date string to a date object
    birth_date = datetime.strptime(birth_date_str, "%d-%m-%Y").date()

    # Calculate the full years
    years = today.year - birth_date.year

    # Adjust years if the birthday hasn't occurred yet this year
    if (today.month, today.day) < (birth_date.month, birth_date.day):
        years -= 1

    # Calculate months and days
    months = today.month - birth_date.month
    days = today.day - birth_date.day

    # Adjust for negative months or days
    if days < 0:
        months -= 1
        last_month = today.replace(day=1) - date.resolution
        days += last_month.day

    if months < 0:
        months += 12

    print(f"Birth Date: {birth_date_str}")
    print(f"Current Date: {today}")
    print(f"¥nYour age is: {years} years, {months} months, and {days} days.")

# Your age is calculated here
get_age("18-02-2005")
```

Output:

Birth Date: 18-02-2005

Current Date: 2025-09-15

Your age is: 20 years, 6 months, and 28 days.

4) Display trigonometric table of sin, cos and tan

```
import math

def display_trigonometric_table():
  # Define a list of common angles in degrees
  angles_in_degrees = [0, 30, 45, 60, 90, 120, 135, 150, 180, 210, 225, 240,
270, 300, 315, 330, 360]

  # Print the table header
  print(f"{'Angle':<8}{'Sin':<10}{'Cos':<10}{'Tan':<10}")
  print("-" * 38)

  # Iterate through the angles to calculate and print values
  for angle in angles_in_degrees:
    # Convert degrees to radians for math functions
    angle_rad = math.radians(angle)

    # Calculate sine and cosine
    sin_val = math.sin(angle_rad)
    cos_val = math.cos(angle_rad)

    # Handle the special case for tan(90) and tan(270)
    if angle % 180 == 90:
        tan_val = "Undefined"
    else:
```

```
        tan_val = f"{math.tan(angle_rad):.4f}"

    print(f"{angle:<8}{sin_val:<10.4f}{cos_val:<10.4f}{tan_val:<10}")

# Run the function to display the table
display_trigonometric_table()
```

Output

Angle	Sin	Cos	Tan
0	0.0000	1.0000	0.0000
30	0.5000	0.8660	0.5774
45	0.7071	0.7071	1.0000
60	0.8660	0.5000	1.7321
90	1.0000	0.0000	Undefined
120	0.8660	-0.5000	-1.7321
135	0.7071	-0.7071	-1.0000
150	0.5000	-0.8660	-0.5774
180	0.0000	-1.0000	-0.0000
210	-0.5000	-0.8660	0.5774
225	-0.7071	-0.7071	1.0000
240	-0.8660	-0.5000	1.7321
270	-1.0000	-0.0000	Undefined
300	-0.8660	0.5000	-1.7321
315	-0.7071	0.7071	-1.0000
330	-0.5000	0.8660	-0.5774
360	-0.0000	1.0000	-0.0000

5) Generate 10 random numbers

```python
import random

def generate_random_numbers(count):
  """Generates a specified number of random integers."""

  print(f"Generating {count} random numbers:")
  for i in range(count):
      # Generate a random integer between 1 and 100 (inclusive)
      random_number = random.randint(1, 100)
      print(random_number)

# Call the function to generate 10 random numbers
generate_random_numbers(10)
```

Generating 10 random numbers:

42

41

77

97

3

57

41

5

19

50

6) Authentication: Ask username, password and compare

```
# Hardcoded credentials for comparison
correct_username = "mca_student"
correct_password = "password123"

# Prompt the user for input
username = input("Enter username: ")
password = input("Enter password: ")

# Compare the entered credentials
if username == correct_username and password == correct_password:
  print("Login successful! Welcome.")
else:
  print("Login failed. Incorrect username or password.")
```

Output

Enter username: mca_student
Enter password: password123
Login successful! Welcome.

7) Authentication: Ask username, password and compare with encryption

```
import hashlib

def hash_password(password):
  # This function turns a password into a unique, unreadable code.
  return hashlib.sha256(password.encode()).hexdigest()

# Define the correct username and password in plain text
correct_username = "mca_student"
correct_password = "password123"

# The program now hashes the correct password internally.
# You don't have to copy and paste the long code yourself.
```

```
correct_hashed_password = hash_password(correct_password)

# Get username and password from the user
username = input("Enter username: ")
password = input("Enter password: ")

# Turn the user's password into a code
user_hashed_password = hash_password(password)

# Compare the username and the two password codes
if username == correct_username and user_hashed_password ==
correct_hashed_password:
  print("Login successful! Welcome.")
else:
  print("Login failed.  Incorrect username or password.")
```

Output:

Enter username: mca_student
Enter password: password123
Login successful! Welcome.

8) Authentication: Ask username, password and compare with hashing

```
import hashlib

def hash_password(password):
  """Hashes a password using SHA-256 for secure storage."""
  # Hashes the password by converting it to bytes and applying the SHA-256
algorithm
  hashed_password = hashlib.sha256(password.encode()).hexdigest()
  return hashed_password

# Define a username and password to authenticate against
correct_username = "mca_student"
correct_password = "password123"
```

```
# Hash the correct password once at the start of the program
correct_hashed_password = hash_password(correct_password)

# Get username and password input from the user
username = input("Enter username: ")
password = input("Enter password: ")

# Hash the password provided by the user
user_hashed_password = hash_password(password)

# Compare the entered username and the hashed passwords
if username == correct_username and user_hashed_password ==
correct_hashed_password:
  print("Login successful! Welcome.")
else:
  print("Login failed. Incorrect username or password.")
```

Output:

Enter username: mca_student
Enter password: password123
Login successful! Welcome.

9) Convert string "Hello$World" into Base64

```
import base64

# The original string
original_string = "Hello$World"

# Convert the string to bytes, as Base64 works with binary data
bytes_to_encode = original_string.encode('utf-8')

# Encode the bytes into Base64
base64_bytes = base64.b64encode(bytes_to_encode)

# Convert the Base64 bytes back to a string for display
base64_string = base64_bytes.decode('utf-8')
```

```
print(f"Original String: {original_string}")
print(f"Base64 Encoded: {base64_string}")
```

Output:

Original String: Hello$World
Base64 Encoded: SGVsbG8kV29ybGQ=

10) Code for String Manipulation

Exercise 1A: Create a string made of the first, middle and last character

Write a program to create a new string made of an input string's first, middle, and last character.

Given:

```
str1 = "James"
```

Expected Output:

```
Jms
```

```
str1 = "James"
new_string = str1[0] + str1[len(str1) // 2] + str1[-1]
print(new_string)
```

Output:

Jms

Exercise 1B: Create a string made of the middle three characters

Write a program to create a new string made of the middle three characters of an input string.

Given:

Case 1

```
str1 = "JhonDipPeta"
```

Output

```
Dip
```

Case 2

```
str2 = "JaSonAy"
```

Output

```
Son
```

```
# Case 1
str1 = "JhonDipPeta"
mid_index = len(str1) // 2
middle_three = str1[mid_index - 1:mid_index + 2]
print(middle_three)

# Case 2
str2 = "JaSonAy"
mid_index = len(str2) // 2
middle_three = str2[mid_index - 1:mid_index + 2]
print(middle_three)
```

Output:

Dip

Son

Exercise 2: Append new string in the middle of a given string

Given two strings, `s1` and `s2`. Write a program to create a new string `s3` by appending `s2` in the middle of `s1`.

Given:

```
s1 = "Ault"

s2 = "Kelly"
```

Expected Output:

```
AuKellylt
```

```
s1 = "Ault"
s2 = "Kelly"
mid_index = len(s1) // 2
s3 = s1[:mid_index] + s2 + s1[mid_index:]
print(s3)
```

Output:

AuKellylt

Exercise 3: Create a new string made of the first, middle, and last characters of each input string

Given two strings, `s1` and `s2`, write a program to return a new string made of s1 and s2's first, middle, and last characters.

Given:

```
s1 = "America"

s2 = "Japan"
```

Expected Output:

```
AJrpan
```

```
s1 = "America"
s2 = "Japan"
res = s1[0] + s2[0] + s1[len(s1) // 2] + s2[len(s2) // 2] + s1[-1] + s2[-1]
print(res)
```

Output:

AJrpan

Exercise 4: Arrange string characters such that lowercase letters should come first

Given string contains a combination of the lower and upper case letters. Write a program to arrange the characters of a string so that all lowercase letters should come first.

Given:

```
str1 = PyNaTive
```

Expected Output:

```
yaivePNT
```

```
str1 = "PyNaTive"
lower_chars = ""
upper_chars = ""
for char in str1:
    if char.islower():
        lower_chars += char
```

```
    else:
        upper_chars += char
sorted_string = lower_chars + upper_chars
print(sorted_string)
```

Output:

yaivePNT

Exercise 5: Count all letters, digits, and special symbols from a given string

Given:

```
str1 = "P@#yn26at^&i5ve"
```

Expected Outcome:

```
Total counts of chars, digits, and symbols



Chars = 8

Digits = 3

Symbol = 4
```

```
str1 = "P@#yn26at^&i5ve"
char_count, digit_count, symbol_count = 0, 0, 0
for char in str1:
    if char.isalpha():
        char_count += 1
    elif char.isdigit():
        digit_count += 1
    else:
        symbol_count += 1
print("Total counts of chars, digits, and symbols")
```

```
print(f"¥nChars = {char_count}¥nDigits = {digit_count}¥nSymbol =
{symbol_count}")
```

Output:

Total counts of chars, digits, and symbols

Chars = 8

Digits = 3

Symbol = 4

Exercise 6: Create a mixed String using the following rules

Given two strings, s1 and s2. Write a program to create a new string s3 made of the first char of s1, then the last char of s2, Next, the second char of s1 and second last char of s2, and so on. Any leftover chars go at the end of the result.

Given:

```
s1 = "Abc"

s2 = "Xyz"
```

Expected Output:

```
AzbycX
```

```
s1 = "Abc"
s2 = "Xyz"
s3 = ""
s2_reversed = s2[::-1]
min_len = min(len(s1), len(s2))
for i in range(min_len):
    s3 += s1[i] + s2_reversed[i]
s3 += s1[min_len:] + s2_reversed[min_len:]
print(s3)
```

Output:

AzbycX

Exercise 7: String characters balance Test

Write a program to check if two strings are balanced. For example, strings s1 and s2 are balanced if all the characters in the s1 are present in s2. The character's position doesn't matter.

Given:

Case 1:

```
s1 = "Yn"

s2 = "PYnative"
```

Expected Output:

```
True
```

Case 2:

```
s1 = "Ynf"
```

```
s2 = "PYnative"
```

Expected Output:

```
False
```

```
def is_balanced(s1, s2):
    for char in s1:
        if char not in s2:
            return False
    return True

# Case 1
s1 = "Yn"
s2 = "PYnative"
print(is_balanced(s1, s2))

# Case 2
s1 = "Ynf"
s2 = "PYnative"
print(is_balanced(s1, s2))
```

Output:

True

Exercise 8: Find all occurrences of a substring in a given string by ignoring the case

Write a program to find all occurrences of "USA" in a given string ignoring the case.

Given:

```
str1 = "Welcome to USA. usa awesome, isn't it?"
```

Expected Outcome:

```
The USA count is: 2
```

```
str1 = "Welcome to USA. usa awesome, isn't it?"
substring = "USA"
count = str1.lower().count(substring.lower())
print(f"The {substring} count is: {count}")
```

Output:

The USA count is: 2

Exercise 9: Calculate the sum and average of the digits present in a string

Given a string s1, write a program to return the sum and average of the digits that appear in the string, ignoring all other characters.

Given:

```
str1 = "PYnative29@#8496"
```

Expected Outcome:

```
Sum is: 38 Average is  6.333333333333333
```

```
str1 = "PYnative29@#8496"
total_sum, count = 0, 0
for char in str1:
    if char.isdigit():
        total_sum += int(char)
        count += 1
```

```
average = total_sum / count
print(f"Sum is: {total_sum} Average is {average}")
```

Output:

Sum is: 38 Average is 6.333333333333333

Exercise 10: Write a program to count occurrences of all characters within a string

Given:

```
str1 = "Apple"
```

Expected Outcome:

```
{'A': 1, 'p': 2, 'l': 1, 'e': 1}
```

```
from collections import Counter
str1 = "Apple"
char_counts = Counter(str1)
print(char_counts)
```

Output:

{'A': 1, 'p': 2, 'l': 1, 'e': 1}

Exercise 11: Reverse a given string

Given:

```
str1 = "PYnative"
```

Expected Output:

```
evitanYP
```

```
str1 = "PYnative"
reversed_string = str1[::-1]
print(f"Original String is: {str1}")
print(f"Reversed String is: {reversed_string}")
```

Output:

evitanYP

Exercise 12: Find the last position of a given substring

Write a program to find the last position of a substring "**Emma**" in a given string.

Given:

```
str1 = "Emma is a data scientist who knows Python. Emma works at google."
```

Expected Output:

```
Last occurrence of Emma starts at index 43
```

```
str1 = "Emma is a data scientist who knows Python. Emma works at google."
index = str1.rfind("Emma")
print(f"Last occurrence of Emma starts at index {index}")
```

Output:

Last occurrence of Emma starts at index 43

Exercise 13: Split a string on hyphens

Write a program to split a given string on hyphens and display each substring.

Given:

```
str1 = Emma-is-a-data-scientist
```

Expected Output:

```
Displaying each substring



Emma

is

a

data

scientist
```

```
str1 = "Emma-is-a-data-scientist"
substrings = str1.split('-')
print("Displaying each substring")
for substring in substrings:
    print(substring)
```

Output:

Displaying each substring

Emma

is

a

data

scientist