**ASSIGNMENT 2**

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

1. Display the difference in dates

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

1. **Display time since epoch in hours and minutes**

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**

1. **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.**

1. **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**

1. **Generate 10 random numbers**

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**

1. **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.**

1. **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.**

1. **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.**

1. **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=**

1. **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**