Roll no: 29

MCA: 1

Semester: 1

Subject: Introduction Of Python Programming

Name: Nirajkumar Taviyad

// 1. display differnce between dates .

```
from datetime import date
d1 = date(2005,5,5)
d2 = date(2023, 9, 11)
diff = d2 - d1
print(f"Difference is:",diff.days)
```

Output:

Difference is: 6703

// 2. display time since epoch in hours and minutes.

```
import time

sse = time.time()
totalseconds = int(sse)
hours = totalseconds // 3600
minutes = (totalseconds % 3600) // 60
print(f"Time since epoch: {hours} hours, {minutes} minutes")
```

Output:

Time since epoch: 488343 hours, 9 minutes

// 3. display your age in years ,month and days.

```
from datetime import datetime
from dateutil. relativedelta import relativedelta
d1 = datetime(2023, 5, 10)
```

```
d2 = datetime(2025, 7, 20)
rd = relativedelta(d2, d1)
print(f"{rd.years} years, {rd.months} months, {rd.days} days")
```

Output:

```
2 years, 2 months, 10 days
```

// 4. display trigonometric table of sin,cos and tanfrom datetime

```
import math
angles = [0, 30, 45, 60, 90]
print("Angle |  sin   |  cos   |  tan")
print("-----------------------------------")
for a in angles:
    rad = math.radians(a)
    s = math.sin(rad)
    c = math.cos(rad)
    if abs(c) < 1e-9:
        t = "undef"
    else:
        t = round(math.tan(rad), 4)
    print(f"{a:>5}° | {round(s,4):>7} | {round(c,4):>7} | {t:>7}")
```

Output:

```
Angle |  sin   |  cos   |  tan
-----------------------------------
    0° |     0.0 |     1.0 |     0.0
   30° |     0.5 |   0.866 |  0.5774
   45° |  0.7071 |  0.7071 |     1.0
   60° |   0.866 |     0.5 |  1.7321
   90° |     1.0 |     0.0 |   undef
```

// 5. Generate 10 random numbers.

```
import random

for i in range (1,10):
```

```
    randomnumbers=random.randint(1,100)
    print(randomnumbers)
```

Output:

```
77
75
56
69
99
27
87
82
40
```

// 6. Authentication: Ask username, password and compare.

```
cusername = "abc456"
cpassword = "acad256"
username = input("Enter username: ")
password = input("Enter password: ")
if username == cusername and password == cpassword:
    print("Login")
else:
    print("Failed.")
```

Output:

```
Enter username: abc456
Enter password: acad256
Login

Enter username: cadv546
Enter password: dvdsa562
Failed
```

// 7. Authentication:Ask Username,password and compare with encryption.

```
from cryptography.fernet import Fernet
key=Fernet.generate_key()
cipher=Fernet(key)
cusername='user123'
cpassword='pass123'.encode()
epassword=cipher.encrypt(cpassword)
username=input("Enter your username: ")
password=input("Enter your password: ").encode()
if(username==cusername and password==cipher.decrypt(epassword)):
    print('Authentication Successful!')
else:
   print('Authentication Failed!')
```

Output:

```
Enter your username: abc123
Enter your password: pass123
Authentication Successful!
```

// 8. Authentication:Ask Username,password and compare with hashing.

```
import hashlib
def hash_password(password):
    hashlib.sha1(password.encode()).hexdigest()
susername = "user1"
spassword_hash = hash_password("abc123")
input_username = input("Enter username: ")
input_password = input("Enter password: ")
input_password_hash = hash_password(input_password)
if input_username == susername and input_password_hash == spassword_hash:
     print("Authentication successful!")
else:
     print("Authentication failed!")
```

Output:

Enter username: user1

Enter password: abc123

Authentication successful!

// 9. Convert string “Hello$World” into Base64.

```
import base64
txt = "Hello$World"
txt_bytes = txt.encode("utf-8")
print(txt_bytes)
base64_bytes = base64.b64encode(txt_bytes)
print(base64_bytes)
base64_string = base64_bytes.decode("utf-8")
print(base64_string)
```

Output:

b'Hello$World'

b'SGVsbG8kV29ybGQ='

SGVsbG8kV29ybGQ=

// 10. Code for String Manipulation .

```
s = input("Enter a string to reverse: ")
reversed_s = s[::-1]
print("Reversed (slice):", reversed_s)
reverse = ""
for i in range(len(s) - 1, -1, -1):
    reverse += s[i]
print("Reversed (loop index):", reverse)

reverse2 = ""
```

```
for ch in s:
    reverse2 = ch + reverse2
print("Reversed (prepend):", reverse2)


reversed_list = [ s[i] for i in range(len(s) - 1, -1, -1) ]
reverse3 = "".join(reversed_list)
print("Reversed (list comprehension):", reverse3)
```

Output:

```
Enter a string to reverse: uday
Reversed (slice): yadu
Reversed (loop index): yadu
Reversed (prepend): yadu
Reversed (list comprehension): yadu
```

List:

1. Sum of All Elements in a List

```
nums = [2, 4, 6, 8, 10]
total = 0
for n in nums:
    total += n
print("Sum:", total)
```

2. Find the Maximum Element in a List

```
nums = [3, 7, 1, 9, 4]
maximum = nums[0]
```

```
for n in nums:
    if n > maximum:
        maximum = n
print("Max:", maximum)
```

3. Reverse a List

```
items = ['a', 'b', 'c', 'd', 'e']
reversed_list = items[::-1]
print("Reversed:", reversed_list)
```

4. Remove Duplicates from a List

```
nums = [1, 2, 2, 3, 4, 4, 5]
unique = list(set(nums))
print("Unique elements:", unique)
```

5. List Comprehension to Create a New List

```
nums = [1, 2, 3, 4, 5]
squares = [x**2 for x in nums]
print("Squares:", squares)
```

DICTiONARIES:

1. Count Frequency of Characters in a String

```
s = "hello world"
freq = {}
for ch in s:
    freq[ch] = freq.get(ch, 0) + 1
```

```
print(freq)
```

2. Merge Two Dictionaries (Add Values When Keys Overlap)

```
dict1 = {"a": 1, "b": 2, "c": 3}
dict2 = {"b": 10, "c": 5, "d": 6}
merged = dict1.copy()
for k, v in dict2.items():
    merged[k] = merged.get(k, 0) + v
print(merged)
```

3. Find the Sum of All Values in a Dictionary

```
scores = {"Alice": 85, "Bob": 90, "Charlie": 78}
total = sum(scores.values())
print("Total score:", total)
```

4. Invert a Dictionary (Swap Keys and Values)

```
original = {"x": 1, "y": 2, "z": 3}
inverted = {v: k for k, v in original.items()}
print(inverted)
```

5. Summation of Nested Dictionary Values

```
test_dict = {
    'gfg': {'a': 4, 'b': 5, 'c': 8},
    'is': {'a': 8, 'c': 10},
    'best': {'c': 19, 'b': 10}
}
res = {}
for sub in test_dict.values():
```

```
    for key, val in sub.items():
        res[key] = res.get(key, 0) + val
print("Summed nested dictionary:", res)
```

SET:

1. Remove Duplicates from a List

```
n = [1, 2, 2, 3, 4, 4, 5]
unique_nums = set(n)
print("Unique:", unique_nums)
```

2. Set Union, Intersection, Difference, Symmetric Difference

```
A = {0, 2, 4, 6, 8}
B = {1, 2, 3, 4, 5}

print("Union:", A | B)
print("Intersection:", A & B)
print("Difference (A - B):", A - B)
print("Symmetric Difference:", A ^ B)
```

3. Check Membership in a Set

```
fruits = {"apple", "banana", "cherry"}
fruit = "banana"
if fruit in fruits:
    print(f"{fruit} is in the set")
else:
    print(f"{fruit} is not in the set")
```

4. Add and Remove Elements in a Set

```
s = {1, 2, 3}
s.add(4)
s.update([5, 6])
print("After adding:", s)
```

```
s.remove(2)
s.discard(10)
print("After removing:", s)
```

5. Loop Through a Set

```
s = {"apple", "banana", "cherry"}
for item in s:
    print(item)
```

TUPLE:

1. Lexicographical Comparison (Tuple Comparison)

```
t1 = (1, 5, 3)
t2 = (1, 4, 9)

print("t1 == t2:", t1 == t2)
print("t1 != t2:", t1 != t2)
print("t1 < t2:", t1 < t2)
print("t1 > t2:", t1 > t2)
```

2. Membership Testing (`in` / `not in`)

```
tup = (10, 20, 30, 40)
print(20 in tup)      # True
print(50 not in tup)  # True
```

3. Bitwise AND Between Two Tuples (Element-wise)

```
t1 = (10, 4, 6, 9)
t2 = (5, 2, 3, 3)

res = tuple(a & b for a, b in zip(t1, t2))
print("Bitwise AND result:", res)
```

4. Bitwise AND Using `operator.iand`

```
import operator
```

```
t1 = (10, 4, 6, 9)
t2 = (5, 2, 3, 3)

res = tuple(map(operator.iand, t1, t2))
print("Bitwise AND with iand:", res)
```

5. Bitwise XOR Between Two Tuples

```
from operator import xor

t1 = (10, 4, 6, 9)
t2 = (5, 2, 3, 3)

res = tuple(map(xor, t1, t2))
print("Bitwise XOR result:", res)
```

CLASS,CONSTRUCTOR,DESTRUCTORES,INHERITANCE,FILES:

1. Simple Class with Constructor and Destructor

```
class Person:
    def __init__(self, name):
        self.name = name
        print(f"Person {self.name} is created")

    def __del__(self):
        print(f"Person {self.name} is destroyed")

p = Person("Alice")
del p
```

2. Parameterized Constructor and Destructor

```
class Employee:
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary
```

```
        print(f"Employee {self.name} created, age: {self.age}, salary: {self.salary}")

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}, Salary: {self.salary}")

    def __del__(self):
        print(f"Employee {self.name} is destroyed")

e1 = Employee("John", 30, 50000)
e1.display()
del e1
```

3. Single Inheritance

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("Animal speaking...")

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def speak(self):
        print(f"{self.name} says Woof! I am a {self.breed}")

d = Dog("Buddy", "Golden Retriever")
d.speak()
```

4. Multilevel Inheritance

```
class Grandparent:
    def __init__(self, gp_name):
        self.gp_name = gp_name

    def show_gp(self):
```

```
        print("Grandparent:", self.gp_name)

class Parent(Grandparent):
    def __init__(self, gp_name, p_name):
        super().__init__(gp_name)
        self.p_name = p_name

    def show_parent(self):
        print("Parent:", self.p_name)

class Child(Parent):
    def __init__(self, gp_name, p_name, c_name):
        super().__init__(gp_name, p_name)
        self.c_name = c_name

    def show_child(self):
        print("Child:", self.c_name)

c = Child("Grandma", "Dad", "Me")
c.show_gp()
c.show_parent()
c.show_child()
```

5. Multiple Inheritance

```
class Flyer:
    def fly(self):
        print("Flying")

class Swimmer:
    def swim(self):
        print("Swimming")

class Duck(Flyer, Swimmer):
    def __init__(self, name):
        self.name = name

    def display(self):
```

```
        print(f"{self.name} can do:")

duck = Duck("Donald")
duck.display()
duck.fly()
duck.swim()
```

6. Hybrid Inheritance (Constructor Chains)

```
class Animal:
    def __init__(self, x):
        print(x[0], "is an animal")

class Marine(Animal):
    def __init__(self, x):
        print(x[0], x[1], "swim")
        super().__init__(x)

class Wings(Animal):
    def __init__(self, x):
        print(x[0], x[1], "fly")
        super().__init__(x)

class Duck(Marine, Wings):
    def __init__(self, x):
        print(x[0])
        super().__init__(x)

Duck(["Duck", "can"])
```

7. Class with Resource Management

```
class FileHandler:
    def __init__(self, filename):
        self.file = open(filename, 'w')
        print("File opened:", filename)
```

```
    def write_data(self, data):
        self.file.write(data)

    def __del__(self):
        self.file.close()
        print("File closed.")

fh = FileHandler("test.txt")
fh.write_data("Hello world")
del fh
```

8. Inheritance + Destructor

```
class Vehicle:
    def __init__(self, brand):
        self.brand = brand
        print(f"Vehicle {self.brand} created")

    def __del__(self):
        print(f"Vehicle {self.brand} destroyed")

class Car(Vehicle):
    def __init__(self, brand, model):
        super().__init__(brand)
        self.model = model
        print(f"Car model {self.model} created")

    def __del__(self):
        print(f"Car model {self.model} destroyed")
        super().__del__()

c = Car("Toyota", "Corolla")
del c
```

9. Class With Counting Instances

```
class Counter:
```

```
    count = 0

    def __init__(self):
        Counter.count += 1
        print("Created. Count =", Counter.count)

    def __del__(self):
        Counter.count -= 1
        print("Destroyed. Count =", Counter.count)

    @classmethod
    def how_many(cls):
        return cls.count

a = Counter()
b = Counter()
print("Current count:", Counter.how_many())
del a
del b
```

10. Inheritance + Method Override + Destructor

```
class Shape:
    def __init__(self):
        print("Shape created")

    def area(self):
        return 0

    def __del__(self):
        print("Shape destroyed")

class Rectangle(Shape):
    def __init__(self, width, height):
        super().__init__()
        self.width = width
        self.height = height
        print("Rectangle created")
```

```
    def area(self):
        return self.width * self.height

    def __del__(self):
        print("Rectangle destroyed")
        super().__del__()

r = Rectangle(5, 3)
print("Area:", r.area())
del r
```