**Assignment 1&2:**

●  **Cows and Bull using conditional & looping statements**

import random

def compare\_numbers(secret\_number, user\_number):

cows = 0

bulls = 0

for i in range(len(secret\_number)):

if secret\_number[i] == user\_number[i]:

cows += 1

elif user\_number[i] in secret\_number:

bulls += 1

return cows, bulls

def main():

secret\_number = str(random.randint(1000, 9999))

attempts = 0

print("Welcome to the Cows and Bulls Game!")

while True:

user\_input = input("Enter a 4-digit number (or 'exit' to quit): ")

if user\_input.lower() == 'exit':

print(f"The secret number was {secret\_number}.")

break

if len(user\_input) != 4 or not user\_input.isdigit():

print("Please enter a valid 4-digit number.")

continue

attempts += 1

cows, bulls = compare\_numbers(secret\_number, user\_input)

if cows == 4:

print(f"Congratulations! You guessed the number {secret\_number} in {attempts} attempts.")

break

else:

print(f"{cows} cows, {bulls} bulls")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Assignment 3:**

● **Create Variables with any Object using Various Naming Conventions**

name=”rakesh”

age=19

place=”chennai”

● **Reverse a String using slicing**

a="priyanka"

b=a[: :-1]

print(b)

**Assignment 4:**

●  **Create various data types and experiment its attribute - 1) Name = “some name” - Convert above string into, upper, lower and capitalize - Replace ‘e’ with ‘E’ using attribute**

name = "some name"

upper\_name = name.upper()

lower\_name = name.lower()

capitalize\_name = name.capitalize()

print("Original:", name)

print("Uppercase:", upper\_name)

print("Lowercase:", lower\_name)

print("Capitalized:", capitalize\_name)

replace\_name = name.replace('e', 'E')

print("Replaced:", replace\_name)

**Output:**

Original: some name

Uppercase: SOME NAME

Lowercase: some name

Capitalized: Some name

Replaced: somE namE

● - 2) L = [1,2,3]

**- Extend above list by using [5,6,7] and remove 5th value**

L = [1, 2, 3]

print("Original List:", L)

extension = [5, 6, 7]

L.extend(extension)

print("Extended List:", L)

removed\_value = L.pop(4)

print("List after removing 5th value:", L)

print("Removed Value:", removed\_value)

**Output:**

Original List: [1, 2, 3]

Extended List: [1, 2, 3, 5, 6, 7]

List after removing 5th value: [1, 2, 3, 5, 7]

Removed Value: 6

● - 3) d = {‘mango’: 10, ‘banana’: 0, ‘apple’: 15, ‘orange’: 0, ‘pineapple’: 20}

- Remove out of stock fruits from above dictionary

- Update mango quantity into 15 & decrease pineapple by 5

d = {'mango': 10, 'banana': 0, 'apple': 15, 'orange': 0, 'pineapple': 20}

print("Original Dictionary:", d)

out\_of\_stock = [key for key, value in d.items() if value == 0]

for fruit in out\_of\_stock:

d.pop(fruit)

print("Dictionary after removing out of stock fruits:", d)

d['mango'] = 15

d['pineapple'] -= 5

print("Updated Dictionary:", d)

**Output:**

Original Dictionary: {'mango': 10, 'banana': 0, 'apple': 15, 'orange': 0, 'pineapple': 20}

Dictionary after removing out of stock fruits: {'mango': 10, 'apple': 15, 'pineapple': 20}

Updated Dictionary: {'mango': 15, 'apple': 15, 'pineapple': 15}

**Assignment 5:**

**1> String concatenation:**

str1 = "Hello, "

str2 = "world!"

result = str1 + str2

print(result)

Output: "Hello, world!"

**List concatenation:**

list1 = [1, 2, 3]

list2 = [4, 5, 6]

result = list1 + list2

print(result)

Output: [1, 2, 3, 4, 5, 6]

**2>string formatting using % operator**

name = "Alice"

age = 30

message = "My name is %s and I am %d years old." % (name, age)

print(message)

Output: "My name is Alice and I am 30 years old."

**string formatting using str.format()**

name = "Bob"

age = 25

message = "My name is {} and I am {} years old.".format(name, age)

print(message)

Output: "My name is Bob and I am 25 years old."

**String formatting using template:**

from string import Template

name = "Eve"

age = 35

template = Template("My name is $name and I am $age years old.")

message = template.substitute(name=name, age=age)

print(message) # Output: "My name is Eve and I am 35 years old."

**string formatting using str.format\_map():**

data = {'name': 'David', 'age': 28}

message = "My name is {name} and I am {age} years old.".format\_map(data)

print(message) # Output: "My name is David and I am 28 years old."

4>**Arithmetic operator:**

Addition:

num1 = 10

num2 = 5

result = num1 + num2

print(result)

Output: 15

Subtraction:

num1 = 20

num2 = 8

result = num1 - num2

print(result)

Output: 12

Multiplication:

num1 = 7

num2 = 3

result = num1 \* num2

print(result)

Output: 21

Division:

num1 = 15

num2 = 3

result = num1 / num2

print(result)

Output: 5.0

**Assignment operators(except +=&=):**

**subtraction:**

num = 10

num -= 3

print(num)

Output: 7

**multiplication:**

num = 5

num \*= 4

print(num)

Output: 20

**Division:**

num = 15

num /= 3

print(num)

Output: 5.0

**Modulus assignment:**

num = 25

num %= 7

print(num)

Output: 4 (remainder of division)

**Assignment-6**

* **Explore operators - Write a program to find power of given number**

def calculate\_power(base, exponent):

result = base \*\* exponent

return result

base = float(input("Enter the base number: "))

exponent = int(input("Enter the exponent: "))

power\_result = calculate\_power(base, exponent)

print(f"{base} raised to the power of {exponent} is: {power\_result}”)

**Explain left/right shift with examples**

* Left Shift (<<): The left shift operator shifts the bits of a binary number to the left by a specified number of positions. This is equivalent to multiplying the number by 2 raised to the power of the shift count.

Syntax: number << shift\_count

num = 5

shifted = num << 2

* Right Shift (>>): The right shift operator shifts the bits of a binary number to the right by a specified number of positions. This is equivalent to dividing the number by 2 raised to the power of the shift count (integer division).

Syntax: number >> shift\_count

num = 20

shifted = num >> 2

* Left Shift Example:
  + Initial binary representation of num: 0101
  + Shifted left by 2 positions: 010100
  + Decimal equivalent of the shifted value: 20
* Right Shift Example:
  + Initial binary representation of num: 010100
  + Shifted right by 2 positions: 0101
  + Decimal equivalent of the shifted value: 5

**How & bitwise operator works**

**Bitwise AND operator** Returns 1 if both the bits are 1 else 0.

**Bitwise or operator** Returns 1 if either of the bit is 1 else 0.

**Bitwise not operator:** Returns one’s complement of the number.

**Bitwise xor operator:** Returns 1 if one of the bits is 1 and the other is 0 else returns false

a = 10

b = 4

print("a & b =", a & b)

print("a | b =", a | b)

print("~a =", ~a)

print("a ^ b =", a ^ b)

**- How and operator, & operator defers each other**

Logical AND Operator (and): The and operator is used for boolean logic operations. It takes two boolean expressions and returns True if both expressions are True, otherwise it returns False.

a = True

b = False

result = a and b

print(result) # Output: False

Bitwise AND Operator (&):The & operator is a bitwise operator that performs bitwise AND operations on individual bits of integers. It compares corresponding bits of two integers and produces a new integer where each bit is set to 1 only if the corresponding bits in both operands are also 1.

a = 25 # Binary: 11001

b = 18 # Binary: 10010

result = a & b

print(result) # Output: 16

**Assignment-7**

**- Refer capitalize function in shared program files, replicate .upper() and .lower() functions**

def my\_upper(s):

result = ""

for char in s:

if 'a' <= char <= 'z':

result += chr(ord(char) - 32)

else:

result += char

return result

def my\_lower(s):

result = ""

for char in s:

if 'A' <= char <= 'Z':

result += chr(ord(char) + 32)

else:

result += char

return result

input\_string = "Hello, World!"

upper\_result = my\_upper(input\_string)

lower\_result = my\_lower(input\_string)

print("Original:", input\_string)

print("Custom Upper:", upper\_result)

print("Custom Lower:", lower\_result)  
**output:**

Original: Hello, World!

Custom Upper: HELLO, WORLD!

Custom Lower: hello, world!

**- Create a odd sequence from given sequence [1,2,34,65,1,2,65,66,44,33,22,87,123412,09,78,76]**

original\_sequence = [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]

odd\_sequence = []

for num in original\_sequence:

if num % 2 != 0:

odd\_sequence.append(num)

print("Original Sequence:", original\_sequence)

print("Odd Sequence:", odd\_sequence)

**output:**

Original Sequence: [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]

Odd Sequence: [1, 65, 1, 65, 33, 87, 9]

- **{‘apple’: 10, ‘mango’: 20, ‘pineapple’: 25, ‘orange’: 30, ‘strawberry’: 50, ‘jackfruit’: 10}**

**Generate a comprehension fruits which has more than 20**

data = {'apple': 10, 'mango': 20, 'pineapple': 25, 'orange': 30, 'strawberry': 50, 'jackfruit': 10}

fruits = {fruit: value for fruit, value in data.items() if value > 20}

print(fruits)

**output:**

{'pineapple': 25, 'orange': 30, 'strawberry': 50}

**Assignment-8:**

**- Create a function to replicate built-in -sum()**

def custom\_sum(iterable, start=0):

total = start

for item in iterable:

total += item

return total

numbers = [1, 2, 3, 4, 5]

total = custom\_sum(numbers)

print("Sum using custom\_sum:", total)

built\_in\_total = sum(numbers)

print("Sum using built-in sum:", built\_in\_total)

**output:**

Sum using custom\_sum: 15

Sum using built-in sum: 15

-  **Create a function to replicate string attribute like, ljust(), rjust() - Refer Practice files**

def custom\_ljust(s, width, fillchar=' '):

if len(s) >= width:

return s

return s + (fillchar \* (width - len(s)))

def custom\_rjust(s, width, fillchar=' '):

if len(s) >= width:

return s

return (fillchar \* (width - len(s))) + s

text = "Hello"

width = 10

left\_justified = custom\_ljust(text, width, fillchar='-')

right\_justified = custom\_rjust(text, width, fillchar='\*')

print("Left Justified:", left\_justified)

print("Right Justified:", right\_justified)

**output:**

Left Justified: Hello-----

Right Justified: \*\*\*\*\*Hello

**- Create a function to find, Palindrome, Fibo and Factorials**

def is\_palindrome(s):

s = s.lower() # Convert to lowercase for case-insensitive comparison

s = s.replace(" ", "") # Remove spaces

return s == s[::-1]

def generate\_fibonacci(n):

fibonacci = [0, 1]

while len(fibonacci) < n:

next\_fib = fibonacci[-1] + fibonacci[-2]

fibonacci.append(next\_fib)

return fibonacci

def calculate\_factorial(n):

if n == 0:

return 1

else:

return n \* calculate\_factorial(n - 1)

word = "Racecar"

print(f"'{word}' is a palindrome:", is\_palindrome(word))

num\_terms = 10

fibonacci\_sequence = generate\_fibonacci(num\_terms)

print(f"Fibonacci sequence up to {num\_terms} terms:", fibonacci\_sequence)

num = 5

factorial = calculate\_factorial(num)

print(f"Factorial of {num}:", factorial)

**output:**

'Racecar' is a palindrome: True

Fibonacci sequence up to 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

Factorial of 5: 120

-  **Create a function to generate range of numbers -**

def custom\_range(start, stop, step=1):

result = []

current = start

if step > 0:

while current < stop:

result.append(current)

current += step

elif step < 0:

while current > stop:

result.append(current)

current += step

return result

print(custom\_range(1, 10))

print(custom\_range(10, 1, -1))

print(custom\_range(0, 20, 5))

**output:**

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[10, 9, 8, 7, 6, 5, 4, 3, 2]

[0, 5, 10, 15]

**Assignment-9:**

**- Refer capitalize function in shared program files, replicate .upper() and .lower() functions**

def my\_upper(s):

result = ""

for char in s:

if 'a' <= char <= 'z':

result += chr(ord(char) - 32)

else:

result += char

return result

def my\_lower(s):

result = ""

for char in s:

if 'A' <= char <= 'Z':

result += chr(ord(char) + 32)

else:

result += char

return result

input\_string = "Hello, World!"

upper\_result = my\_upper(input\_string)

lower\_result = my\_lower(input\_string)

print("Original:", input\_string)

print("Custom Upper:", upper\_result)

print("Custom Lower:", lower\_result)  
**output:**

Original: Hello, World!

Custom Upper: HELLO, WORLD!

Custom Lower: hello, world!

**- Create a odd sequence from given sequence [1,2,34,65,1,2,65,66,44,33,22,87,123412,09,78,76]**

original\_sequence = [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]

odd\_sequence = []

for num in original\_sequence:

if num % 2 != 0:

odd\_sequence.append(num)

print("Original Sequence:", original\_sequence)

print("Odd Sequence:", odd\_sequence)

**output:**

Original Sequence: [1, 2, 34, 65, 1, 2, 65, 66, 44, 33, 22, 87, 123412, 9, 78, 76]

Odd Sequence: [1, 65, 1, 65, 33, 87, 9]

- **{‘apple’: 10, ‘mango’: 20, ‘pineapple’: 25, ‘orange’: 30, ‘strawberry’: 50, ‘jackfruit’: 10}**

**Generate a comprehension fruits which has more than 20**

data = {'apple': 10, 'mango': 20, 'pineapple': 25, 'orange': 30, 'strawberry': 50, 'jackfruit': 10}

fruits = {fruit: value for fruit, value in data.items() if value > 20}

print(fruits)

**output:**

{'pineapple': 25, 'orange': 30, 'strawberry': 50}

**Assignment-10:**

**- Create a function to replicate built-in -sum()**

def custom\_sum(iterable, start=0):

total = start

for item in iterable:

total += item

return total

numbers = [1, 2, 3, 4, 5]

total = custom\_sum(numbers)

print("Sum using custom\_sum:", total)

built\_in\_total = sum(numbers)

print("Sum using built-in sum:", built\_in\_total)

**output:**

Sum using custom\_sum: 15

Sum using built-in sum: 15

-  **Create a function to replicate string attribute like, ljust(), rjust() - Refer Practice files**

def custom\_ljust(s, width, fillchar=' '):

if len(s) >= width:

return s

return s + (fillchar \* (width - len(s)))

def custom\_rjust(s, width, fillchar=' '):

if len(s) >= width:

return s

return (fillchar \* (width - len(s))) + s

text = "Hello"

width = 10

left\_justified = custom\_ljust(text, width, fillchar='-')

right\_justified = custom\_rjust(text, width, fillchar='\*')

print("Left Justified:", left\_justified)

print("Right Justified:", right\_justified)

**output:**

Left Justified: Hello-----

Right Justified: \*\*\*\*\*Hello

**- Create a function to find, Palindrome, Fibo and Factorials**

def is\_palindrome(s):

s = s.lower() # Convert to lowercase for case-insensitive comparison

s = s.replace(" ", "") # Remove spaces

return s == s[::-1]

def generate\_fibonacci(n):

fibonacci = [0, 1]

while len(fibonacci) < n:

next\_fib = fibonacci[-1] + fibonacci[-2]

fibonacci.append(next\_fib)

return fibonacci

def calculate\_factorial(n):

if n == 0:

return 1

else:

return n \* calculate\_factorial(n - 1)

word = "Racecar"

print(f"'{word}' is a palindrome:", is\_palindrome(word))

num\_terms = 10

fibonacci\_sequence = generate\_fibonacci(num\_terms)

print(f"Fibonacci sequence up to {num\_terms} terms:", fibonacci\_sequence)

num = 5

factorial = calculate\_factorial(num)

print(f"Factorial of {num}:", factorial)

**output:**

'Racecar' is a palindrome: True

Fibonacci sequence up to 10 terms: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

Factorial of 5: 120

-  **Create a function to generate range of numbers -**

def custom\_range(start, stop, step=1):

result = []

current = start

if step > 0:

while current < stop:

result.append(current)

current += step

elif step < 0:

while current > stop:

result.append(current)

current += step

return result

print(custom\_range(1, 10))

print(custom\_range(10, 1, -1))

print(custom\_range(0, 20, 5))

**output:**

[1, 2, 3, 4, 5, 6, 7, 8, 9]

[10, 9, 8, 7, 6, 5, 4, 3, 2]

[0, 5, 10, 15]

**Assignment 11:**

**1>**

def custom\_range(start, stop=None, step=1):

if stop is None:

stop = start

start = 0

current = start

while (step > 0 and current < stop) or (step < 0 and current > stop):

yield current

current += step

for num in custom\_range(5):

print(num, end=" ")

Output: 0 1 2 3 4

print()

for num in custom\_range(2, 10, 2):

print(num, end=" ")

Output: 2 4 6 8

print()

for num in custom\_range(10, 2, -2):

print(num, end=" ")

Output: 10 8 6 4

2>

def custom\_range\_recursive(start, stop=None, step=1):

if stop is None:

stop = start

start = 0

if (step > 0 and start >= stop) or (step < 0 and start <= stop):

return []

else:

return [start] + custom\_range\_recursive(start + step, stop, step)

for num in custom\_range\_recursive(5):

print(num, end=" ")

Output: 0 1 2 3 4

print()

for num in custom\_range\_recursive(2, 10, 2):

print(num, end=" ")

Output: 2 4 6 8

print()

for num in custom\_range\_recursive(10, 2, -2):

print(num, end=" ")

Output: 10 8 6 4

3>

Recursive function :

def gcd\_recursive(a, b):

if b == 0:

return a

else:

return gcd\_recursive(b, a % b)

num1 = 48

num2 = 18

result = gcd\_recursive(num1, num2)

print(f"GCD of {num1} and {num2} is {result}")

Lambda function:

gcd\_lambda = lambda a, b: a if b == 0 else gcd\_lambda(b, a % b)

num1 = 48

num2 = 18

result = gcd\_lambda(num1, num2)

print(f"GCD of {num1} and {num2} is {result}")

4>

def calculate\_square(number):

return number \*\* 2

- mymodule.py

from mymodule import calculate\_square

num = 5

square = calculate\_square(num)

print(f"The square of {num} is {square}")

**Assignment 12:**

**1>**

def is\_palindrome(word):

word = word.lower()

return word == word[::-1]

def fibonacci(n):

if n <= 0:

return []

elif n == 1:

return [0]

elif n == 2:

return [0, 1]

else:

fibo\_sequence = [0, 1]

for i in range(2, n):

next\_fibo = fibo\_sequence[i - 1] + fibo\_sequence[i - 2]

fibo\_sequence.append(next\_fibo)

return fibo\_sequence

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

from mymodule import math\_functions

word = "radar"

if math\_functions.is\_palindrome(word):

print(f"{word} is a palindrome.")

else:

print(f"{word} is not a palindrome.")

n = 10

fibonacci\_sequence = math\_functions.fibonacci(n)

print(f"Fibonacci sequence of length {n}: {fibonacci\_sequence}")

num = 5

fact = math\_functions.factorial(num)

print(f"Factorial of {num} is {fact}")

2>

from mymodule import math\_functions

def main():

word = "radar"

if math\_functions.is\_palindrome(word):

print(f"{word} is a palindrome.")

else:

print(f"{word} is not a palindrome.")

n = 10

fibonacci\_sequence = math\_functions.fibonacci(n)

print(f"Fibonacci sequence of length {n}: {fibonacci\_sequence}")

num = 5

fact = math\_functions.factorial(num)

print(f"Factorial of {num} is {fact}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

3>

class MathFunctions:

@staticmethod

def is\_palindrome(word):

word = word.lower()

return word == word[::-1]

@staticmethod

def fibonacci(n):

if n <= 0:

return []

elif n == 1:

return [0]

elif n == 2:

return [0, 1]

else:

fibo\_sequence = [0, 1]

for i in range(2, n):

next\_fibo = fibo\_sequence[i - 1] + fibo\_sequence[i - 2]

fibo\_sequence.append(next\_fibo)

return fibo\_sequence

@staticmethod

def factorial(n):

if n == 0:

return 1

else:

return n \* MathFunctions.factorial(n - 1)

math\_functions = MathFunctions()

word = "radar"

if math\_functions.is\_palindrome(word):

print(f"{word} is a palindrome.")

else:

print(f"{word} is not a palindrome.")

n = 10

fibonacci\_sequence = math\_functions.fibonacci(n)

print(f"Fibonacci sequence of length {n}: {fibonacci\_sequence}")

num = 5

fact = math\_functions.factorial(num)

print(f"Factorial of {num} is {fact}")

**Assignment -13**:

Single inheritance

class Animal:

def speak(self):

Pass

class Dog(Animal):

def speak(self):

return "Woof!"

dog = Dog()

print(dog.speak())

Output: "Woof!"

Multiple inheritance:

class Flyer:

def fly(self):

pass

class Swimmer:

def swim(self):

pass

class Duck(Flyer, Swimmer):

def quack(self):

return "Quack!"

duck = Duck()

print(duck.fly())

print(duck.swim())

print(duck.quack())

Output: "Quack!"

Multilevel inheritance:

class Vehicle:

def start(self):

return "Engine started."

class Car(Vehicle):

def drive(self):

return "Car is being driven."

class SportsCar(Car):

def speed\_up(self):

return "Sports car is speeding up."

car = SportsCar()

print(car.start())

print(car.drive())

print(car.speed\_up())

Output: "Engine started."

Output: "Car is being driven.”

Output: " Sports car is speeding up.”

**Assignment -14**:

def divide(a, b):

try:

return a / b

except ZeroDivisionError:

return "Division by zero not allowed."

except TypeError:

return "Invalid input types."

try:

print(undefined\_variable) # Causes a NameError

except NameError:

print("Variable is not defined.")

result = divide(10, '2')

print(result)

output:

Variable is not defined.

Invalid input types.