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=”Priyanka”

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.
```