1.Difference between list and tuple.

Ans:- list:-

:List is mutable, you can change, add, or remove elements after creation.

:Syntax for list is [ ].

:List is some what slower when compare to tuple.

Example:

```
my_list = [1, 2, 3]
print(type(my_list))
```

Tuple:-

:Tuple is immutable, you can't change, add, or remove elements after the creation.

: Tuple Defined with parentheses ()

: tuple is faster when compare to list.

Example:

```
my_tuple = (1, 2, 3)
print(type(my_tuple))
```

2.How does left shift and right shift operators work? Give detailed explanation with examples.

Ans:- Left Shift (<<):

Moves all bits of the number to the left by the given count. New empty bits on the right are filled with 0. Effectively multiplies the number by 2n2^n2n.

Example:-

```
x = 5        # binary: 0000 0101
y = x << 1    # shift left by 1
z = x << 3    # shift left by 3

print(y)  # 10   (binary: 0000 1010)
```

print(z) # 40   (binary: 0010 1000)

5<<1 → 5 * 2**1=10

5<<3 → 5* 2**3=40.

Right Shift (>>)

Moves all bits to the right by the given count.

Effectively divides the number by 2n2^n2n, (floor division).

Example:-

x = 20   # binary: 0001 0100

y = x >> 1   # shift right by 1

z = x >> 2   # shift right by 2

print(y) # 10   (20 // 2 = 10)

print(z) # 5   (20 // 4 = 5).

3. A treasure chest opens only if the correct code (say 1234) is entered. The player has 3 attempts. Write a program to simulate this lock system.

Ans:-

```
correct_code = "1234"
max_attempts = 3
for attempt in range (1, max_attempts + 1):
    entry = input(f"Attempt {attempt}: Enter the treasure chest code: ")
    if entry == correct_code:
        print("Chest opened! You found the treasure.")
        break
    else:
        if attempt < max_attempts:
```

```
        print(f"Incorrect code. You have {max_attempts - attempt} attempts left.")
    else:
        print("Chest locked. Access deined.").
```

4. A teacher records the marks of 5 students in a list.

Write a program to calculate:

Average marks Highest marks

Student index with lowest marks.

Ans:-

```
marks = [85, 92, 76, 64, 90]
average = sum(marks) / len(marks)
highest = max(marks)
lowest_index = marks.index(min(marks))

print("Marks of students:", marks)
print("Average marks:", average)
print("Highest marks:", highest)
print("Student index with lowest marks:", lowest_index)
```

Output:-

```
Marks of students: [85, 92, 76, 64, 90]
Average marks: 81.4
Highest marks: 92
Student index with lowest marks: 3
```

5. You are organizing a party . Write a program that takes a list of guest names and:

Greets each guest by name

Finds if your name is present in the guest list

Prints the number of guests whose names start with "A".

Ans:-

```
guests = ["ballaya", "Bob", "Ananya", "Charlie", "Aditi", "Vishnu"]

for guest in guests:
    print(f"Welcome to the party, {guest}!")

print()  # just a blank line

my_name = "Vishnu"

if my_name in guests:
    print(f" Yes! {my_name}, you are on the guest list.")

else:
    print(f" Sorry, {my_name}, you are not on the guest list.")

count_A = sum(1 for g in guests if g.startswith("A"))

print(f"Number of guests whose names start with 'A': {count_A}")
```

Output:

```
Welcome to the party, ballaya!

Welcome to the party, Bob!

Welcome to the party, Ananya!

Welcome to the party, Charlie!

Welcome to the party, Aditi!

Welcome to the party, Vishnu!

Number of guests whose names start with 'A': 2
```

6. A detective suspects that some numbers are prime codes. Write a function that takes a number and checks if it is prime or not.

Ans:-

```
def is_prime(n):
    if n <= 1:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True
num = int(input("Enter a number: "))
if is_prime(num):
    print(f"{num} is a PRIME code ")
else:
    print(f"{num} is NOT a prime code ")
```

Output:

Enter a number: 17

17 is a Prime code

Enter a number: 20

20 is Not a prime code .

7. What are decorators and write examples for it.

Ans:- Decorator in Python is a function that takes another function as input, adds some extra functionality to it, and returns a new function without changing the original function’s code.

Example:-

```
def my_decorator(func):
    def wrapper():
        print("Before the function is called")
        func()
        print(" After the function is called")
    return wrapper
def say_hello():
    print("Hello, World!")
say_hello()
```

Output:

Before the function is called

Hello, World!

After the function is called

8. Create a Car class with attributes brand, model, and price. Add a method display_info() that prints car details. Write any 2 relevant static variables in the class. Create at least 3 car objects and display their info. Also print the static variables seperately without using methods. Write a method to return model of a given of a particular object. Also include a method to change model for a particular car.

Ans:-

```
class Car:
 wheels = 4
  vehicle_type = "Automobile"
    def __init__(self, brand, model, price):
        self.brand = brand
        self.model = model
        self.price = price
    def display_info(self):
```

```
print(f"Brand: {self.brand}, Model: {self.model}, Price: {self.price}")
    def get_model(self):
        return self.model
    def set_model(self, new_model):
        self.model = new_model
car1 = Car("Toyota", "Corolla", 20000)
car2 = Car("Honda", "Civic", 22000)
car3 = Car("Tesla", "Model 3", 35000)
print("Car Details:")
car1.display_info()
car2.display_info()
car3.display_info()
print("\nStatic Variables:")
print("Wheels:", Car.wheels)
print("Vehicle Type:", Car.vehicle_type)
print("\nModel of car1:", car1.get_model())
car1.set_model("Camry")
print("Updated model of car1:", car1.get_model())
```

Output:

```
Car Details:
Brand: Toyota, Model: Corolla, Price: 20000
Brand: Honda, Model: Civic, Price: 22000
Brand: Tesla, Model: Model 3, Price: 35000
Static Variables:
Wheels: 4
```

Vehicle Type: Automobile

Model of car1: Corolla

Updated model of car1: Camry

9. What are the different types of variables and methods. Explain them.

Ans:-

Types of Variables

1. Instance Variables

- Belong to individual objects (instances) of a class.
- Defined inside methods, typically in __init__.
- Example: self.brand in a Car class.

2. Class Variables (Static Variables)

- Belong to the class itself rather than any instance.
- Shared across all instances.
- Defined directly inside the class but outside any method.
- Example: wheels in a Car class representing number of wheels.

3. Local Variables

- Defined inside methods or functions.
- Only accessible within the method.
- Used for temporary storage or computation.

Types of Methods

1. Instance Methods

- The most common type.
- Operate on instance data via self.
- Can access instance variables and class variables.
- Defined with def method_name(self, ...).

- Example: display_info(self) in a Car class to print attributes.

2. Class Methods

- Operate on the class itself, not instances.
- Use the decorator @classmethod.
- Take cls as the first parameter instead of self.
- Can access or modify class variables.
- Example: A method that returns or updates static variables.

3. Static Methods

- Defined with @staticmethod.
- Do not take self or cls.
- Behave like plain functions but belong to the class's namespace.
- Cannot access instance or class data directly.
- Useful for utility functions related to the class but independent of instances.

4. Special Methods (Magic Methods)

- Begin and end with double underscores, like __init__, __str__.
- Provide built-in behavior for classes (construction, representation).
- Not called directly but through language syntax.

10. What are the different types of scopes available in Python? Give example for each of this scopes.

Ans:- Python follows the **LEGB Rule** for scope resolution:

Local → Enclosing → Global → Built-in.

Local:-

Variables created inside a function belong to the local scope.

They can only be accessed inside that function

Example:

```
def my_function():
    x = 10
    print("Local x:", x)
my_function()
```

Enclosing:-

This is the scope of outer functions.

The inner function can access variables of its enclosing function using nonlocal.

Example:-

```
def outer():
    y = 20
    def inner():
        nonlocal y   # refers to 'y' in outer function
        y = 25
        print("Inner y:", y)
    inner()
    print("Outer y:", y)
outer()
```

Output:-

Inner y: 25

Outer y: 25

Global:-

Variables created outside all functions are in the global scope.

They can be accessed inside functions using the global keyword if modification is needed.

Example:-

```
z = 30
def my_func():
```

```
    global z
    z = 40
    print("Inside function, z:", z)
my_func()
print("Outside function, z:", z)
```

Output:-

```
Inside function, z: 40
Outside function, z: 40
```

Build-in:-

These are names that are predefined in Python (keywords, functions, exceptions, etc.).

Examples: print(), len(), range(), etc.

Example:-

```
print(len([1, 2, 3]))   # len() is from built-in scope
len = 100.
```