FUNCTIONS

1. Prime Number Checker

Write a function is_prime(n) that returns True if n is prime, else False . Use it to print all prime numbers between 1 and 100.

```
def is_prime(n: int):

    if n < 2:
            return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

for i in range(100):
    if is_prime(i+1):
        print("Prime:", i+1)
```

2. Temperature Converter

Write a function convert_temp(value, unit) that converts: Celsius to Fahrenheit, Fahrenheit to Celsius Use conditionals inside the function.

```
def convert_temp(value, unit):
    if unit == "C":
        fahern = (value * (9/5)) + 32
        return fahern
    else:
        celsius = (value - 32) * (5 / 9)
        return celsius

print(convert_temp(56, "F"))
```

3. Recursive Factorial Function

Create a function factorial(n) using recursion to return the factorial of a number.

```
def factorial(n):
    if n == 1 or n == 0:
        return 1

    return n * factorial(n-1)

print(factorial(5))
```

CLASSES

4. Class: Rectangle

Attributes: length , width

Methods:

area()

perimeter()

is_square() → returns True if length == width

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.width * self.length

    def perimeter(self):
        return 2 * (self.length + self.width)

    def is_square(self):
        if self.length == self.width:
            return True
        return False

r1 = Rectangle(3, 23)
print(r1.area())
print(r1.perimeter())
print(r1.is_square())
```

5. Class: BankAccount

Attributes: name , balance

Methods:

deposit(amount)

withdraw(amount)

get_balance()

Prevent withdrawal if balance is insufficient

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

    def deposit(self, amt):
```

```python
        self.balance += amt

    def withdraw(self, amt):
        if self.balance >= amt:
            self.balance -= amt
        else:
            print("Insufficient balance!!")

    def get_balance(self):
        return self.balance

b1 = BankAccount("Tharun", 1000)
b1.deposit(250)
b1.withdraw(50)
print(b1.get_balance())
```

6. Class: Book

Attributes: title , author , price , in_stock

Method: sell(quantity)

Reduces stock

Throws an error if quantity exceeds stock

```python
class Book:
    def __init__(self, title, author, price, in_stock):
        self.title = title
        self.author = author
        self.price = price
        self.in_stock = in_stock

    def sell(self, quantity):
        if self.in_stock >= quantity:
            self.in_stock -= quantity
            print("Done")
        else:
            raise ValueError("Out of Stock!!")

b1 = Book("Pytorch", "Daniel Brouke", 1299, 25)
b1.sell(26)
```

7. Student Grade System

Create a class Student with:

Attributes: name , marks (a list)

Method:

average()

grade() — returns A/B/C/F based on average

```
class Student:
    def __init__(self, name, marks:list):
        self.name = name
        self.marks = marks

    def average(self):
        return sum(self.marks) / len(self.marks)

    def grade(self):
        average = self.average()

        if average >= 90:
            return "A"
        elif average >= 70:
            return "B"
        elif average >= 50:
            return "C"
        else:
            return "F"

s1 = Student("Tharun", [90, 87, 89, 100, 67])
print(s1.average())
print(s1.grade())
```

INHERITANCE

8. Person → Employee

Class Person : name , age

Class Employee inherits Person , adds emp_id , salary

Method display_info() shows all details

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

class Employee(Person):
    def __init__(self, name, age, emp_id, salary):
        super().__init__(name, age)
        self.emp_id = emp_id
        self.salary = salary

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}, ID: 
{self.emp_id}, Salary: {self.salary}")
```

```
e1 = Employee("Tharun", 21, 1, 45_000)
e1.display_info()
```

9. Vehicle → Car , Bike

Base Class: Vehicle(name, wheels)

Subclasses:

Car : additional attribute fuel_type

Bike : additional attribute is_geared

Override a method description() in both

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

    def description(self):
        print(f"Name: {self.name}")
        print(f"Wheels: {self.wheels}")

class Car(Vehicle):
    def __init__(self, name, wheels, fuel_type):
        super().__init__(name, wheels)
        self.fuel_type = fuel_type

    def description(self):
        super().description()
        print(f"Fuel Type: {self.fuel_type}")

class Bike(Vehicle):
    def __init__(self, name, wheels, is_geared):
        super().__init__(name, wheels)
        self.is_geared = is_geared

    def description(self):
        super().description()
        print(f"Geared: {self.is_geared}")

c1 = Car("Jeep Compass", 4, "Disel")
c1.description()

print("__________________________________")

b1 = Bike("Contiential GT 650", 2, True)
b1.description()
```

10. Polymorphism with Animals

Base class Animal with method speak()

Subclasses Dog , Cat , Cow override speak() with unique sounds

Call speak() on each object in a loop

```python
class Animal:
    def speak(self):
        print("Animal Sound")

class Dog(Animal):
    def speak(self):
        print("Bow wow")

class Cat(Animal):
    def speak(self):
        print("Meow")

class Cow(Animal):
    def speak(self):
        print("Moo...")

animals = (Dog(), Cat(), Cow())

for i in animals:
    i.speak()
```