2.1 challenges

class BankAccount:

def \_\_init\_\_(self, account\_number, account\_holder\_name, initial\_balance=0.0):

self.\_\_account\_number = account\_number

self.\_\_account\_holder\_name = account\_holder\_name

self.\_\_account\_balance = initial\_balance

def deposit(self, amount):

if amount > 0:

self.\_\_account\_balance += amount

# self.\_\_account\_balance = self.\_\_account\_balance+amount

print("Deposited ₹{}. New balance: ₹{}".format(amount,

self.\_\_account\_balance))

else:

print("Invalid deposit amount.")

def withdraw(self, amount):

if amount > 0 and amount <= self.\_\_account\_balance:

self.\_\_account\_balance -= amount

# self.\_\_account\_balance = self.\_\_account\_balance - amount

print("Withdrew ₹{}. New balance: ₹{}".format(amount,

self.\_\_account\_balance))

else:

print("Invalid withdrawal amount or insufficient balance.")

def display\_balance(self):

print("Account balance for {} (Account #{}): ₹{}".format(

self.\_\_account\_holder\_name, self.\_\_account\_number,

self.\_\_account\_balance))

# Create an instance of the BankAccount class

account = BankAccount(account\_number="123456789",

account\_holder\_name="Hari Prabu",

initial\_balance=5000.0)

# Test deposit and withdrawal functionality

account.display\_balance()

account.deposit(500.0)

account.withdraw(200.0)

account.withdraw(20000.0)

account.display\_balance()

2.2 chellenge

# Define the base class Player

class Player:

def play(self):

print("The player is playing cricket.")

# Define the derived class Batsman

class Batsman(Player):

def play(self):

print("The batsman is batting.")

# Define the derived class Bowler

class Bowler(Player):

def play(self):

print("The bowler is bowling.")

# Create objects of Batsman and Bowler classes

batsman = Batsman()

bowler = Bowler()

# Call the play() method for each object

batsman.play()

bowler.play()

3.1

def linearSearchProduct(productList, targetProduct):

indices = []

for index, product in enumerate(productList):

if product == targetProduct:

indices.append(index)

return indices

# Example usage:

products = ["shoes", "boot", "loafer", "shoes", "sandal", "shoes"]

target = "shoes"

target2 = 'apple'

result = linearSearchProduct(products, target)

print(result)

3.2

class Student:

def \_\_init\_\_(self, name, roll\_number, cgpa):

self.name = name

self.roll\_number = roll\_number

self.cgpa = cgpa

def sort\_students(student\_list):

# Sort the list of students in descending order of CGPA

sorted\_students = sorted(student\_list,

key=lambda student: student.cgpa,

reverse=True)

# Syntax - lambda arg:exp

return sorted\_students

# Example usage:

students = [

Student("Hari", "A123", 7.8),

Student("Srikanth", "A124", 8.9),

Student("Saumya", "A125", 9.1),

Student("Mahidhar", "A126", 9.9),

]

sorted\_students = sort\_students(students)

# Print the sorted list of students

for student in sorted\_students:

print("Name: {}, Roll Number: {}, CGPA: {}".format(student.name,

student.roll\_number,

student.cgpa))