```
"""
Module with test simulation function to help pick the optimal number of units to produce.

BANA 5440 - Assignment 1 (Group)

Author: Maxwell Barnes, Justin Bowser, Elise Donkor, Rohini Iyengar, Nisha Ramasubramonian
Date: November 9, 2022
"""

#import necessary modules
import numpy as np

def create_demand_normally_dist(mean_value, standard_dev, total_numbers):
    """
    create create_demand_randomly_dist: This function will generate a list of integers normal
    : param
        mean_value: demand mean
        standard_dev: demand standard deviation
        total_numbers: demand samples
    : return: demand list
    """
    # generate normally distirubted list
    demand_list = np.random.normal(mean_value,standard_dev,total_numbers)

    #convert each list item to an integer.
    for i in range(len(demand_list)):
        demand_list[i] = int(demand_list[i])

    #return the list
    return demand_list

def create_demand_uniformly_dist(low,high,demand_samples):
    """
    create_demand_uniformly_dist: This function will generate a list of integers normally dis
    : param
        low: low value of the range of number of samples
        high: high value of the range of number of samples
        demand_samples: number of elements in the list that we want drawn
    : return: demand list
    """
    #generate uniformly distirubted list
    demand_list = np.random.uniform(low,high,demand_samples)

    #convert each list item to an integer.
    for i in range(len(demand_list)):
        demand_list[i] = int(demand_list[i])

    #return the list
    return demand_list
```

```
def simulate_sales(demand_list, units_produced):
    """
    simulate_sales: This function will convert a list of demand to a list of projected sales
    : param
        demand_list: computed demand values
        units_produced: integer number of unitts produced
    : return: list of simulated sales
    """

    #create list for sales
    simulated_sales = []

    #For each value in the demand list and test it against the units produced.
    for i in range(len(demand_list)):
        #If the demand is greater than units produced; sales are equal to units produced.
        if demand_list[i] > units_produced:
            simulated_sales.append(units_produced)
        #In any other case sales are equal to demand.
        else:
            simulated_sales.append(demand_list[i])

    #return the list
    return simulated_sales

def simulate_profit(simulated_sales, units_produced, production_cost, disposal_cost, retail_p
    """
    simulate_profit: This function will convert the list of sales to profit. Profit = Revenue
    : param
        simpulated_sales computed demand values
        units_produced: demand standard deviation
        production_cost:
        disposal_cost:
        retail_price:
        demand_list:
    : return: list or simulated profit values
    """
    #create profit list
    simulated_profit = []

    # for each value in the sales list
    for i in range(len(simulated_sales)):

        #calculate revenue
        revenue = simulated_sales[i] * retail_price

        #if demand is greater than or equal to units produced calculate profit
        if demand_list[i] >= units_produced:
            simulated_profit.append(revenue - production_cost)
        #Otherwise calculate profit with disposal costs equal to disposal cost * units overpr
        else:
            #calculate units overproduced
            units_over =  units_produced - demand_list[i]
```

```
            #calculate total disposal cost
            total_disposal_cost = units_over * disposal_cost
            #calculate profit and add it to the profit list
            simulated_profit.append(revenue - production_cost - total_disposal_cost)

    #return the list
    return simulated_profit

def test(total_cost, retail_price, disposal_cost1, disposal_cost2 = 0 , chance = 0):
    """
    test: Core test function - This function creates a normalized demand array and finds the
    : param
        total_cost: production cost per unit
        retail_price: retail price per unit
        disposal_cost1: current disposal cost per unit
        disposal_cost2: (optional) potential increased disposal cost
        chance: (optional) chance of increase as a decimal
    """

    #calculate disposal cost
    disposal_cost = (disposal_cost1 * (1-chance)) + (disposal_cost2 * chance)

    #static variables
    demand_mean = 150
    demand_stdev = 20
    demand_samples = 1000

    #Manufacture test case variables
    optimized = (0,0,0)

    #test various produciton units in a range of 100 to 200
    for units_produced in range(100,200,5):

        #calculate production cost for each units produced
        production_cost = units_produced * total_cost

        #run functions
        demand_list = create_demand_normally_dist(demand_mean,demand_stdev,demand_samples)
        simulated_sales = simulate_sales(demand_list, units_produced)
        simulated_profit = simulate_profit(simulated_sales, units_produced, production_cost,

        # Get statistics to output
        average = np.mean(simulated_profit)
        stdev = np.std(simulated_profit)

        #set optimized stats to current test value so long as it is greater than the previous
        if average > optimized[1]:
            optimized = (units_produced, average, stdev)

    #print results in readable format
    print("Optimal units manufactured = "  + str(optimized[0]) + "\nMaximum Profit = " + "$"
```

```
def bonus(total_cost, retail_price, disposal_cost, chance):
    """
    bonus: bonus test function - This function is for the bonus question only.
    This function creates a demand array.
    A normalized and uniformly distributed array based on a uner inputted  varible.
    This function finds the optimized profit based on various variables.
    : param
        total_cost: production cost per unit
        retail_price: retail price per unit
        disposal_cost: current disposal cost per unit
        chance: (optional) chance of increase as a decimal
    """
    #static variables
    demand_mean = 150
    demand_stdev = 20
    demand_samples = 1000
    low = 100
    high = 200

    #Manufacture test case variables
    optimized = (0,0,0)

    #test various produciton units in a range of 100 to 200
    for units_produced in range(100,200,5):

        #calculate production cost for each units produced
        production_cost = units_produced * total_cost

        #create local demand list of the two demand generations
        demand_list = []
        demand_list_norm = create_demand_normally_dist(demand_mean,demand_stdev,int(demand_sa
        demand_list_uniform = create_demand_uniformly_dist(low,high,int(demand_samples * (1-c
        demand_list.extend(demand_list_norm)
        demand_list.extend(demand_list_uniform)

        #run functions
        simulated_sales = simulate_sales(demand_list, units_produced)
        simulated_profit = simulate_profit(simulated_sales, units_produced, production_cost,

        # Get statistics to output
        average = np.mean(simulated_profit)
        stdev = np.std(simulated_profit)

            #set optimized stats to current test value so long as it is greater than the prev
        if average > optimized[1]:
            optimized = (units_produced, average, stdev)

    #print results in readable format
    print("Optimal units manufactured = "  + str(optimized[0]) + "\nMaximum Profit = " + "$"
```

```
# Question 1 Test
test(28.5,150,8.5)
```

```
     Optimal units manufactured = 165
     Maximum Profit = $17202.74
     Maximum Profit Std Deviation = $2497.58
```

```
# Question 2 Test
test(28.5,150,8.5,17,0)
```

```
     Optimal units manufactured = 160
     Maximum Profit = $17217.2
     Maximum Profit Std Deviation = $2285.3
```

```
#Bonus Question Test
bonus(28.5,150,8.5,.5)
```

```
     Optimal units manufactured = 175
     Maximum Profit = $17094.11
     Maximum Profit Std Deviation = $3480.5
```