Q1..

|  |
| --- |
| while True:  *# Get input values from the user*      total\_bandwidth\_khz = int(input(*"*Enter total bandwidth in kHz (e.g., 33000): *"*))      channel\_bandwidth\_khz = int(input(*"*Enter channel bandwidth in kHz (e.g., 50): *"*))      control\_bandwidth\_khz = int(input(*"*Enter control channel bandwidth in kHz (e.g., 1000): *"*))  *# Total available channels*      total\_available\_channels = total\_bandwidth\_khz // channel\_bandwidth\_khz  *# Number of control channels*      control\_channels = control\_bandwidth\_khz // channel\_bandwidth\_khz  *# Get cell reuse factors from the user*      reuse\_factors = input(*"*Enter reuse factors separated by commas (e.g., 4,7,12): *"*)      reuse\_factors = [int(n.strip()) for n in reuse\_factors.split(*"*,*"*)]  *# Calculate and print results for each reuse factor*      for reuse\_factor in reuse\_factors:  *# Total channels per cell*          channels\_per\_cell = total\_available\_channels // reuse\_factor    *# Voice channels per cell*          voice\_channels = (total\_available\_channels - control\_channels) // reuse\_factor    *# Control channels per cell*          control\_channels\_per\_cell = channels\_per\_cell - voice\_channels    *# Display results*          print(f*"*\nFor {reuse\_factor}-cell reuse:*"*)          print(f*"*  Total channels per cell: {channels\_per\_cell}*"*)          print(f*"*  Voice channels per cell: {voice\_channels}*"*)          print(f*"*  Control channels per cell: {control\_channels\_per\_cell}*"*)  *# Ask if the user wants to repeat the process*      repeat = input(*"*\nDo you want to enter another set of values? (yes/no): *"*).strip().lower()      if repeat != *'*yes*'*:          print(*"*Exiting the program.*"*)          break    *# 1. If a total of 33 MHz of bandwidth is allocated to a particular FDD cellular telephone*  *# system which uses two 25 kHz simplex channels to provide full duplex voice and*  *# control channels, compute the number of channels available per cell if a system uses-*  *# (a)  4-cell reuse.*  *# (b) 7-cell reuse.*  *# (c) 12-cell reuse.*  *# If 1 MHz of the allocated spectrum is dedicated to control channels, determine an*  *# equitable distribution of control channels and voice channels in each cell for each of*  *# the three systems.* |

Q2.

|  |
| --- |
| import math  *# Function to calculate frequency reuse factor Q based on cluster size N*  def calculate\_frequency\_reuse\_factor(N):      return math.sqrt(3 \* N)  *# Function to calculate S/I ratio in dB*  def calculate\_sir(q, n, i0):      return 10 \* math.log10((q \*\* n) / i0)  *# Main loop to allow multiple inputs*  while True:      try:  *# Get user input for required S/I ratio and number of co-channel cells*          required\_sir\_db = float(input(*"*Enter the required S/I ratio in dB (e.g., 15): *"*))          num\_cochannel\_cells = int(input(*"*Enter the number of co-channel interfering cells (e.g., 6): *"*))  *# Get path loss exponent and cluster sizes*          path\_loss\_exponent = int(input(*"*Enter the path loss exponent (e.g., 3 or 4): *"*))          cluster\_sizes = input(*"*Enter possible cluster sizes separated by commas (e.g., 7, 12): *"*)    *# Convert cluster sizes from string to list of integers*          cluster\_sizes = [int(size.strip()) for size in cluster\_sizes.split(*"*,*"*)]  *# Iterate over cluster sizes*          for N in cluster\_sizes:  *# Calculate frequency reuse factor Q*              Q = calculate\_frequency\_reuse\_factor(N)    *# Calculate S/I ratio*              sir\_db = calculate\_sir(Q, path\_loss\_exponent, num\_cochannel\_cells)    *# Display results*              print(f*"*\nFor path loss exponent n = {path\_loss\_exponent} and cluster size N = {N}:*"*)              print(f*"*  Frequency Reuse Factor (Q): {Q:.3f}*"*)              print(f*"*  Calculated S/I Ratio: {sir\_db:.2f} dB*"*)    *# Check if this meets the required S/I*              if sir\_db >= required\_sir\_db:                  print(f*"*  This cluster size (N = {N}) meets the required S/I ratio of {required\_sir\_db} dB.*"*)              else:                  print(f*"*  This cluster size (N = {N}) does NOT meet the required S/I ratio.*"*)      except ValueError:          print(*"*Invalid input. Please enter numeric values where required.*"*)    *# Ask if the user wants to repeat the process*      repeat = input(*"*\nDo you want to enter another set of values? (yes/no): *"*).strip().lower()      if repeat != *'*yes*'*:          print(*"*Exiting the program.*"*)          break    *# 2. If a signal to interference ratio of 15 dB is required for satisfactory forward channel*  *# performance of a cellular system, what is the frequency reuse factor and cluster size*  *# that should be used for maximum capacity if the path loss exponent is-*  *# (a) n = 4.*  *# (b) n = 3.*  *# Assume that there are 6 co-channels cells in the first tier and all of them are at the*  *# same distance from the mobile. Use suitable approximations.* |

Q3.

|  |
| --- |
| *# Problem-3: Calculate the number of users supported for blocking probabilities*  *# for different numbers of trunked channels in a blocked calls cleared system.*  *# Erlang B capacity table for 0.5% GOS (blocking probability)*  erlang\_table = {      1: 0.005,      2: 0.105,      4: 0.701,      5: 1.1300,      10: 3.9600,      20: 11.1000,      100: 80.9000  }  def calculate\_supported\_users(trunked\_channels, traffic\_intensity\_per\_user):  *# Get the offered traffic intensity (A) from the table based on trunked channels*      if trunked\_channels in erlang\_table:          A = erlang\_table[trunked\_channels]  *# Calculate total number of users (U)*          U = A / traffic\_intensity\_per\_user    *# Since one channel can only support one user, round up to the next whole number*          return max (1,int(U) )  *# Ensure at least 1 user can be supported*      else:          return None  *# Return None if trunked channels not found in the table*  while True:  *# Get user inputs for blocking probability and traffic intensity*      blocking\_probability = float(input(*"*Enter the blocking probability (in 0.5 %): *"*))      traffic\_intensity\_per\_user = float(input(*"*Enter the traffic intensity per user (in Erlangs 0.1): *"*))  *# Get trunked channels from user*      trunked\_channels\_input = input(*"*Enter the trunked channels (comma-separated, e.g., 1, 5, 10, 20, 100): *"*)      trunked\_channels\_list = list(map(int, trunked\_channels\_input.split(*'*,*'*)))  *# Output results for each number of trunked channels*      for channels in trunked\_channels\_list:          users\_supported = calculate\_supported\_users(channels, traffic\_intensity\_per\_user)          if users\_supported is not None:              print(f*"*Trunked Channels: {channels}, Supported Users: {users\_supported}*"*)          else:              print(f*"*Trunked Channels: {channels} not found in the table.*"*)  *# Ask if the user wants to input more values*      continue\_input = input(*"*Do you want to input another set of values? (yes/no): *"*).strip().lower()      if continue\_input != *'*yes*'*:          break  *# 3. How many users can be supported for 0.5% blocking probability for the following*  *# number of trunked channels in a blocked calls cleared system?*  *# (a) 1,*  *# (b) 5,*  *# (c) 10,*  *# (d) 20,*  *# (e) 100.*  *# Assume each user generates 0.1 Erlangs of traffic.* |

Q4.

|  |
| --- |
| *# Constants*  blocking\_probability = 0.02  *# 2%*  lambda\_calls\_per\_hour = 2  *# Average number of calls per user per hour*  average\_call\_duration\_hours = 3 / 60  *# Average call duration in hours*  total\_population = 2000000  *# Total number of residents*  *# Traffic intensity calculation*  A = lambda\_calls\_per\_hour \* average\_call\_duration\_hours  *# Offered traffic intensity in Erlangs*  *# Function to calculate users supported by each system*  def calculate\_users\_supported(cells, channels, A\_table):      A\_e = A\_table.get(channels, 0)  *# Get the Erlangs capacity from the table*      if A\_e == 0:          raise ValueError(f*"*No data for channels: {channels}*"*)        total\_users = A\_e / A \* cells  *# Total users supported*      return int(total\_users)  *# Erlangs capacity lookup table for different channel configurations*  A\_table = {      19: 12,   *# For System A*      57: 45,   *# For System B*      100: 88   *# For System C*  }  *# Main loop to collect inputs for multiple systems*  while True:      user\_inputs = []        while True:          system\_name = input(*"*Enter the system name (A, B, C): *"*).strip().upper()          if system\_name in [*'*A*'*, *'*B*'*, *'*C*'*]:              cells = int(input(f*"*Enter the number of cells for System {system\_name}: *"*))              channels = int(input(f*"*Enter the number of channels per cell for System {system\_name}: *"*))              user\_inputs.append((system\_name, cells, channels))  *# No feedback for invalid input, just continue asking for valid input*  *# Option to stop inputting systems*          continue\_input = input(*"*Do you want to input another system? (yes/no): *"*).strip().lower()          if continue\_input != *'*yes*'*:              break  *# Calculations and results*      total\_users\_supported = 0      for system\_name, cells, channels in user\_inputs:          users = calculate\_users\_supported(cells, channels, A\_table)          market\_penetration = (users / total\_population) \* 100          total\_users\_supported += users          print(f*"*System {system\_name}: {users} users, Market Penetration: {market\_penetration:.3f}%*"*)  *# Combined results*      combined\_market\_penetration = (total\_users\_supported / total\_population) \* 100      print(f*"*Total Users Supported: {total\_users\_supported}, Combined Market Penetration: {combined\_market\_penetration:.3f}%*"*)  *# Prompt to continue the outer loop*      continue\_outer = input(*"*Do you want to input values for another batch? (yes/no): *"*).strip().lower()      if continue\_outer != *'*yes*'*:          break  *# 4. An urban area has a population of 2 million residents. Three competing trunked*  *# mobile networks (systems A, B, and C) provide cellular service in this area. System A*  *# has 394 cells with 19 channels each, system B has 98 cells with 57 channels each, and*  *# system C has 49 cells, each with 100 channels. Find the number of users that can be*  *# supported at 2% blocking if each user averages 2 calls per hour at an average call*  *# duration of 3 minutes. Assuming that all three trunked systems are operated at*  *# maximum capacity, compute the percentage market penetration of each cellular*  *# provider.* |

Q5.

|  |
| --- |
| def main():      while True:          try:  *# Given values*              total\_coverage\_area = float(input(*"*Enter total coverage area in square miles (default 1300): *"*) or 1300)              cell\_radius = float(input(*"*Enter cell radius in miles (default 4): *"*) or 4)              allocated\_spectrum = float(input(*"*Enter allocated spectrum in Hz (default 40,000,000): *"*) or 40\_000\_000)              channel\_width = float(input(*"*Enter channel width in Hz (default 60,000): *"*) or 60\_000)              frequency\_reuse\_factor = int(input(*"*Enter frequency reuse factor (default 7): *"*) or 7)              offered\_traffic\_per\_user = float(input(*"*Enter offered traffic per user in Erlangs (default 0.03): *"*) or 0.03)      *# (a) Calculate the number of cells in the service area*              area\_of\_cell = 2.5981 \* (cell\_radius \*\* 2)  *# area in square miles*              number\_of\_cells = total\_coverage\_area / area\_of\_cell              number\_of\_cells = int(number\_of\_cells)  *# Use integer value for number of cells*  *# (b) Calculate the number of channels per cell*              total\_channels\_per\_cell = allocated\_spectrum // (channel\_width \* frequency\_reuse\_factor)  *# Use integer division*  *# (c) Traffic intensity of each cell (already given)*  *# This is taken from the Erlang B table based on 95 channels per cell and a 2% GOS*              traffic\_intensity\_per\_cell = 84  *# in Erlangs (reference value)*    *# (d) Calculate the maximum carried traffic*              maximum\_carried\_traffic = number\_of\_cells \* traffic\_intensity\_per\_cell  *# (e) Calculate the total number of users that can be served for 2% GOS*              total\_users = maximum\_carried\_traffic / offered\_traffic\_per\_user  *# (f) Calculate the number of mobiles per channel*              total\_channels = allocated\_spectrum // channel\_width  *# Use integer division*              mobiles\_per\_channel = total\_users / total\_channels  *# (g) Calculate the theoretical maximum number of users that could be served at one time by the system*              theoretical\_max\_users = total\_channels\_per\_cell \* number\_of\_cells  *# Print results*              print(f*"*a) Number of cells in the service area: {number\_of\_cells} cells*"*)              print(f*"*b) Number of channels per cell: {total\_channels\_per\_cell} channels/cell*"*)              print(f*"*c) Traffic intensity of each cell: {traffic\_intensity\_per\_cell} Erlangs/cell*"*)              print(f*"*d) Maximum carried traffic: {maximum\_carried\_traffic:.2f} Erlangs*"*)              print(f*"*e) Total number of users that can be served for 2% GOS: {total\_users:.0f} users*"*)              print(f*"*f) Number of mobiles per channel: {mobiles\_per\_channel:.2f} mobiles/channel*"*)              print(f*"*g) Theoretical maximum number of users that could be served at one time: {theoretical\_max\_users} users*"*)    *# Ask if the user wants to input another set of values*              continue\_outer = input(*"*Do you want to input values for another batch? (yes/no): *"*).strip().lower()              if continue\_outer != *'*yes*'*:                  print(*"*Exiting the program.*"*)                  break  *# Exit the loop and the program*          except ValueError as e:              print(f*"*Invalid input: {e}. Please enter numerical values.*"*)  if \_\_name\_\_ == *"*\_\_main\_\_*"*:      main()    *# 5. A certain city has an area of 1,300 square miles and is covered by a cellular system*  *# using a 7-cell reuse pattern. Each cell has a radius of 4 miles and the city is allocated*  *# 40 MHz of spectrum with a full duplex channel bandwidth of 60 kHz. Assume a GOS*  *# of 2% for an Erlang B system is specified. If the offered traffic per user is 0.03*  *# Erlangs, compute-*  *# a) The number of cells in the service area,*  *# b) The number of channels per cell,*  *# c) Traffic intensity of each cell,*  *# d) The maximum carried traffic,*  *# e) The total number of users that can be served for 2% GOS,*  *# f) The number of mobiles per channel, and*  *# g) The theoretical maximum number of users that could be served at one time by*  *# the system.* |

Q6.

|  |
| --- |
| import math  def power\_conversion(transmitter\_power\_watts):      power\_mw = transmitter\_power\_watts \* 1000      power\_dbm = 10 \* math.log10(power\_mw)      power\_dbw = 10 \* math.log10(transmitter\_power\_watts)      return power\_dbm, power\_dbw  def received\_power(distance\_m, transmitter\_power\_w, gt, gr, frequency\_hz):      c = 3 \* 10\*\*8  *# Speed of light in m/s*      wavelength = c / frequency\_hz      pr = (transmitter\_power\_w \* gt \* gr \* (wavelength\*\*2)) / ((4 \* math.pi \* distance\_m)\*\*2)      pr\_dbm = 10 \* math.log10(pr \* 1000)      return pr\_dbm  def received\_power\_at\_distance(pr\_100\_dbm, distance\_m, reference\_distance\_m):      path\_loss = 20 \* math.log10(distance\_m / reference\_distance\_m)      pr\_new\_dbm = pr\_100\_dbm - path\_loss      return pr\_new\_dbm  while True:  *# User inputs*      transmitter\_power\_w = float(input(*"*Enter the transmitter power in watts: *"*))      gt = float(input(*"*Enter the transmitter gain (in linear scale, e.g., 1 for unity gain): *"*))      gr = float(input(*"*Enter the receiver gain (in linear scale, e.g., 1 for unity gain): *"*))      frequency\_hz = float(input(*"*Enter the carrier frequency in MHz: *"*)) \* 10\*\*6  *# Convert MHz to Hz*  *# Convert power*      power\_dbm, power\_dbw = power\_conversion(transmitter\_power\_w)  *# Calculate received power at 100 m*      distance\_100\_m = 100      received\_power\_100\_dbm = received\_power(distance\_100\_m, transmitter\_power\_w, gt, gr, frequency\_hz)  *# Calculate received power at 10 km*      reference\_distance\_m = 100      distance\_10\_km = 10000      received\_power\_10\_km\_dbm = received\_power\_at\_distance(received\_power\_100\_dbm, distance\_10\_km, reference\_distance\_m)  *# Display results*      print(f*"*\nFinal Results:*"*)      print(f*"*(a) Transmitter power P\_t in dBm: {power\_dbm:.1f} dBm*"*)      print(f*"*(b) Transmitter power P\_t in dBW: {power\_dbw:.1f} dBW*"*)      print(f*"*(c) Received power P\_r at 100 m in dBm: {received\_power\_100\_dbm:.1f} dBm*"*)      print(f*"*(d) Received power P\_r at 10 km in dBm: {received\_power\_10\_km\_dbm:.1f} dBm*"*)    *# Check if the user wants to continue*      continue\_choice = input(*"*\nDo you want to perform another calculation? (yes/no): *"*).strip().lower()      if continue\_choice != *'*yes*'*:          break  *# 6. If a transmitter produces 50 watts of power, express the transmit power in units of*  *# a) dBm,*  *# and b) dBW.*  *# If 50 watts is applied to a unity gain antenna with a 900 MHz carrier frequency,*  *# a) Find the received power in dBm at a free space distance of 100 m from the*  *# antenna,*  *# b) What is P (10 km)?*  *# Assume unity gain for the receiver antenna.* |

Q7.

|  |
| --- |
| import math  def calculate\_path\_loss(fc, hte, hre, d):  *# Calculate the correction factor for effective mobile antenna height*      a\_hre = 3.2 \* (math.log10(11.75 \* hre)) \*\* 2 - 4.97  *# Calculate path loss using the Okumura-Hata model*      path\_loss = (69.55 +                   26.16 \* math.log10(fc) -                   13.82 \* math.log10(hte) -                   a\_hre +                   (44.9 - 6.55 \* math.log10(hte)) \* math.log10(d))        return path\_loss  while True:  *# Input values from the user*      try:          fc = float(input(*"*Enter the frequency (in MHz): *"*))  *# Frequency in MHz*          hte = float(input(*"*Enter the base station height (in meters): *"*))  *# Base station height in meters*          hre = float(input(*"*Enter the mobile station height (in meters): *"*))  *# Mobile station height in meters*          d = float(input(*"*Enter the distance (in kilometers): *"*))  *# Distance in kilometers*  *# Calculate path loss*          path\_loss = calculate\_path\_loss(fc, hte, hre, d)          print(f*"*The path loss is approximately {path\_loss:.2f} dB.*"*)      except ValueError:          print(*"*Please enter valid numerical values.*"*)  *# Ask if the user wants to continue or exit*      continue\_input = input(*"*Do you want to calculate path loss for another set of values? (yes/no): *"*).strip().lower()      if continue\_input != *'*yes*'*:          break  *# 7. Determine the path loss of a 900MHz cellular system in a large city from a base*  *# station with the height of 100m and mobile station installed in a vehicle with antenna*  *# height of 2m. The distance between mobile and base station is 4Km.* |

Q8.

|  |
| --- |
| import math  def calculate\_path\_loss(fc, hb, d):  *# Calculate the T-R separation distance in km*      d\_km = math.sqrt(20\*\*2 + 30\*\*2) / 1000  *# Convert meters to kilometers*  *# Calculate path loss using the Okumura-Hata model*      path\_loss = (135.41 +                   12.49 \* math.log10(fc) -                   4.99 \* math.log10(hb) +                   (46.84 - 2.34 \* math.log10(hb)) \* math.log10(d\_km))        return path\_loss  while True:  *# Input values from the user*      try:          fc = float(input(*"*Enter the frequency (in GHz, e.g., 1.8): *"*))  *# Frequency in GHz*          hb = float(input(*"*Enter the base station height (in meters): *"*))  *# Base station height in meters*  *# Calculate path loss*          path\_loss = calculate\_path\_loss(fc, hb, 0)  *# d is calculated within the function*          print(f*"*The path loss is approximately {path\_loss:.2f} dB.*"*)      except ValueError:          print(*"*Please enter valid numerical values.*"*)  *# Ask if the user wants to continue or exit*      continue\_input = input(*"*Do you want to calculate path loss for another set of values? (yes/no): *"*).strip().lower()      if continue\_input != *'*yes*'*:          break  *# 8. Determine the path loss between base station (BS) and mobile station (MS) of a*  *# 1.8GHz PCS system operating in a high-rise urban area. The MS is located in a*  *# perpendicular street to the location of the BS. The distances of the BS and MS to the*  *# corner of the street are 20 and 30 meters, respectively. The base station height is 20m.* |

Q9.

|  |
| --- |
| import math  *# Constants*  c = 3 \* 10\*\*8  *# Speed of light in m/s*  *# Function to perform calculations based on user input*  def calculate\_antenna\_parameters():  *# User inputs*      f = float(input(*"*Enter frequency in MHz: *"*))      g = float(input(*"*Enter gain of antenna in dB: *"*))    *# Question (a)*  *# Convert gain from dB to linear scale*      gain\_linear = 10 \*\* (g / 10)  *# Gain in linear scale*  *# Calculate wavelength in meters*      lambda\_ = c / (f \* 10\*\*6)  *# Calculate length of the antenna (1/4 of wavelength)*      L = lambda\_ / 4  *# Length of the antenna in meters*  *# Display results for part (a)*      print(*'*For (a)*'*)      print(*'*---------*'*)      print(f*'*Length of the antenna: {L \* 100:.2f} cm*'*)  *# Convert to cm*      print(f*'*Gain of the antenna: {gain\_linear:.2f} = {g:.2f} dB\n*'*)  *# Question (b)*      d = float(input(*"*Enter T-R separation distance in meters: *"*))      E0 = float(input(*"*Enter electric-field strength in V/m (e.g., 0.001): *"*))  *# Directly input 0.001 instead of 10\*\*-3*      d0 = float(input(*"*Enter reference distance in meters: *"*))      ht = float(input(*"*Enter height of transmitting antenna in meters: *"*))      hr = float(input(*"*Enter height of receiving antenna in meters: *"*))  *# Electric Field Calculation*      Er\_d = (2 \* E0 \* d0 \* 2 \* math.pi \* ht \* hr) / (lambda\_ \* d\*\*2)  *# Electric Field in V/m*  *# Effective Aperture Calculation*      Ae = (gain\_linear \* lambda\_\*\*2) / (4 \* math.pi)  *# Effective Aperture in m²*  *# Received Power Calculation*      Pr = (Er\_d\*\*2 / (120 \* math.pi)) \* Ae  *# Received power in watts*      Pr\_dB = 10 \* math.log10(Pr)  *# Received power in dBW*      Pr\_dBm = Pr\_dB + 30  *# Convert to dBm*  *# Display results for part (b)*      print(*'*For (b)*'*)      print(*'*---------*'*)      print(f*'*Electric Field, Er(d): {Er\_d:.9f} V/m*'*)      print(f*'*Effective Aperture, Ae: {Ae:.3f} m²*'*)      print(f*'*Received power at {d} meters distance: {Pr\_dB:.2f} dBW*'*)      print(f*'*Received power at {d} meters distance: {Pr\_dBm:.2f} dBm\n*'*)  *# Main loop for multiple inputs*  while True:      calculate\_antenna\_parameters()    *# Prompt to continue or exit*      continue\_input = input(*"*Do you want to calculate for another antenna configuration? (yes/no): *"*).strip().lower()      if continue\_input != *'*yes*'*:          break    *# 9. A mobile is located 5 km away from a base station and uses a vertical λ /4 monopole*  *# antenna with a gain of 2.55 dB to receive cellular 3 radio signals. The E-field at 1 km*  *# from the transmitter is measured to be V/m. The carrier frequency used for this*  *# system is 900 MHz.*  *# a) Find the length and the gain of the receiving antenna.*  *# b) Find the received power at the mobile using the 2-ray ground reflection model*  *# assuming the height of the transmitting antenna is 50 m and the receiving*  *# antenna is 1.5m above ground.* |

Q10.

|  |
| --- |
| import math  def calculate\_erlang\_c(R, N, Au):  *# Calculate area covered per cell*      area = round(2.5981 \* R\*\*2)  *# Area in sq km*      C = N / 4  *# Number of channels per cell (assuming a 4-cell system)*  *# Traffic intensity at C=15, GOS=0.05*      A = 9  *# Given traffic intensity*  *# Question (a)*      U = A / Au  *# Number of users*      U\_per = U / area  *# Number of users per square km*      print(f*"*(a) Number of users per square km: {int(U\_per)} users/sq km\n*"*)  *# Question (b)*      lamda = 1  *# Lambda = 1 call/hour*      H = (Au / lamda) \* 3600  *# Holding time in seconds*      t = 10  *# Time in seconds*      Prb = math.exp(-(C - A) \* t / H)  *# Probability that a delayed call will wait more than t seconds*      print(f*"*(b) The probability that a delayed call will have to wait: {Prb \* 100:.2f}%\n*"*)  *# Question (c)*      Prc = 0.05 \* Prb  *# Probability that a call is delayed more than 10 seconds*      print(f*"*(c) The probability that a call will be delayed: {Prc \* 100:.2f}%\n*"*)  *# Main loop for user input*  while True:      try:  *# Collect user inputs*          R = float(input(*"*Enter the cell radius in km: *"*))          N = int(input(*"*Enter the total number of channels: *"*))          Au = float(input(*"*Enter the traffic per user in Erlangs: *"*))  *# Calculate and display results*          calculate\_erlang\_c(R, N, Au)      except ValueError:          print(*"*Invalid input. Please enter numeric values.*"*)  *# Prompt to continue or exit*      continue\_input = input(*"*Do you want to calculate again? (yes/no): *"*).strip().lower()      if continue\_input != *'*yes*'*:          break  *# 10. A hexagonal cell within a 4-cell system has a radius of 1.387 km. A total of 60*  *# channels are used within the entire system. If the load per user is 0.029 Erlangs, and*  *# λ= call/hour, compute the following for an Erlang C system that has a 5% probability*  *# of a delayed call-*  *# a) How many users per square kilometer will this system support?*  *# b) What is the probability that a delayed call will have to wait for more than 10s?*  *# c) What is the probability that a call will be delayed for more than 10 seconds?* |