Aayush Shah
D1 - 19BCE245
25 February 2021

Practical 2

A. The bell shaped Gaussian function, is one of the most widely used functions in science and technology. The parameters m and s > 0 are prescribed real numbers. Make a program for evaluating this function for different values of s, x and m. Ask the user to input the values

```
#!/usr/bin/env python3

import math

def cal_gaussian(s,x,m):
  answer = float(1)
  answer /= (s*math.sqrt(2*math.pi))
  answer *= math.exp((-1/2)*(((x-m)/s)**2))
  return answer

s = float(0)
while(s<1):
  s = float(input("Enter s : "))
  if(s<1):
      print("Value of s must be greater than zero.")
x = float(input("Enter x : "))
m = float(input("Enter m : "))
print("The answer is : " , cal_gaussian(s, x, m))
```

OUTPUT :

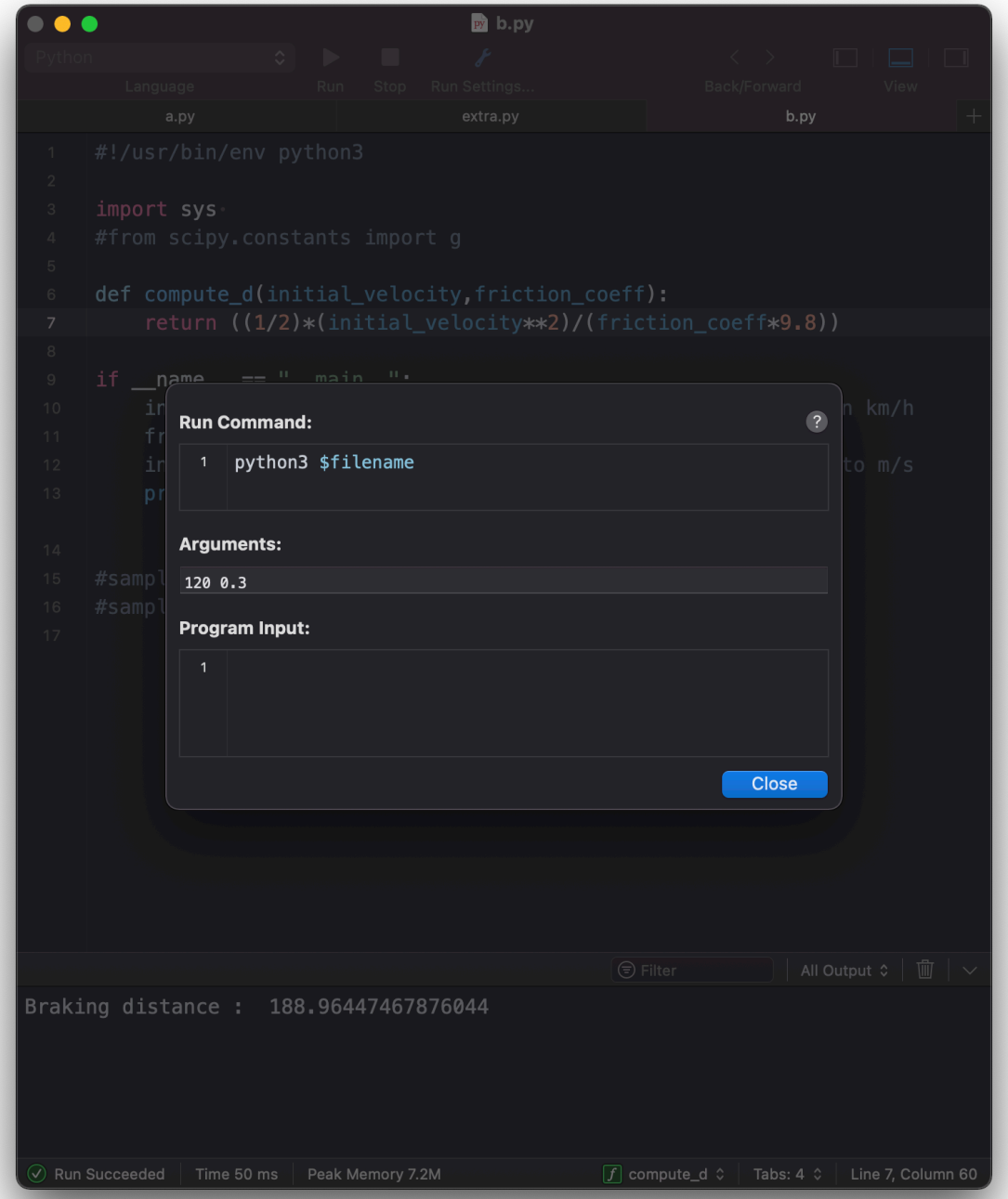
```
Enter s : 5
Enter x : 10
Enter m : 15
The answer is :  0.04839414490382868
```

Run Succeeded | Time 55 ms | Peak Memory 7.3M | Symbol | Tabs: 4 | 18 Lines, 404 Characters

B. A car driver, driving at velocity v_0, suddenly puts on the brake. What is braking distance d needed to stop the car? One can derive, using Newton's second law of motion or a corresponding energy equation, that Make a program for computing d above equation, when the initial car velocity v_0 and the friction coefficient μ are given on the command line. Run the program for two cases: $v_0 = 120$ and $v_0 = 50$ km/h, both with μ = 0.3 (μ is dimensionless).

```
#!/usr/bin/env python3

import sys
from scipy.constants import g

def compute_d(initial_velocity,friction_coeff):
    return ((1/2)*(initial_velocity**2)/(friction_coeff*g))

if __name__ == "__main__":
    initial_velocity = float(sys.argv[1])   #to be enetered in km/h
    friction_coeff = float(sys.argv[2])
    initial_velocity = initial_velocity/3.6 #converting it into m/s
    print("Braking distance : ", compute_d(initial_velocity, friction_coeff))

#sample input 1 : 120 0.3
#sample input 2 : 50 0.3
```

OUTPUT :



(EXTRA)

C. Team management wants to pick the top 7 batsmen for the next game. Team management has data of all of their available 11 batsman details. Help the team management to select the best 6 batsmen (need to make a strategy for selecting the top 6 players). The format of all the data is shown below. The first column is showing the playerID, Remaining 10 columns are having data of run scored by a

player in 10 innings. If a player is not selected in playing 11, then NA is mentioned. PlayerID must be described format. Generate all the data randomly.

```
#!/usr/bin/env python3

#total,highest,avg =>Histogram

import random
from random import randrange
import array as arr

import array as arr
#a = arr.array('i', [])
matrix = []

#Generating unique ids through set
set_of_ids = set()
while len(set_of_ids)<11:
  set_of_ids.add(random.randint(1, 999999))

# Here -1 is for NA.

#avg_and_max_of_all = []

#ID i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 Total avg Max <50 (50,100) (100,150) (150,200)

#generating 10 innings data
for i in range(11):          # A for loop for row entries
  a = ['P' + str(set_of_ids.pop()).zfill(6)]
  count = 0;
  count_less_than_50 = 0
  count_less_than_100 = 0
  count_less_than_150 = 0
  count_less_than_200 = 0
  total_score = 0;
  max_score = 0;
  matches_played = 0;
  for j in range(1,11):      # A for loop for column entries
```

```
        number = random.randint(-1, 200)
        while (count>=2 and number==-1):
            number = random.randint(-1, 200)

        if(number != -1):
            if(number<50):
                count_less_than_50 += 1
            elif(number<100):
                count_less_than_100 += 1
            elif(number<150):
                count_less_than_150 += 1
            else:
                count_less_than_200 += 1

            total_score += number
            matches_played += 1
            if(max_score<number):
                max_score = number
        else:
            count+=1
        a.append(number)

    a.append(total_score)    #appending total
    a.append(round(total_score/matches_played,1)) #appending avg
    a.append(max_score) #appending max
    a.append(count_less_than_50)
    a.append(count_less_than_100)
    a.append(count_less_than_150)
    a.append(count_less_than_200)

# avg_and_max = []
# avg_and_max.append(total_score/matches_played)
# avg_and_max.append(max_score)
# avg_and_max_of_all.append(avg_and_max)
    matrix.append(a)



#Printing data
print("ID       I1   I2   I3   I4   I5   I6   I7   I8   I9   I10   | Total   Avg     Max <50 <100 <150 <200")
for i in range(11):
```

```
 for j in range(18):
        if (matrix[i][j]==-1):
             print("NA", end = "   ")
        else:
             if(j==12):
                  print(str(matrix[i][j]).zfill(5), end = "  ")
             elif(j==11):
                  print(" | ",str(matrix[i][j]).zfill(4), end = "   ")
             else:
                  print(str(matrix[i][j]).zfill(3), end = "  ")
 print()

#Printing avg and max data
#print("\n\nAVERAGE   |   MAX")
#for i in range(11):
# print(str(round(avg_and_max_of_all[i][0],1)).zfill(5),end = "        ")
# print(str(round(avg_and_max_of_all[i][1],2)).zfill(3),end = "")
# print()

#Top 6 players :
print("\n\nPlayers in their avg. score's increasing order :\n")
matrix = sorted(matrix, key = lambda a_entry:a_entry[11])
print("ID       I1   I2   I3   I4   I5   I6   I7   I8   I9   I10   | Total   Avg     Max <50 <100 <150 <200")
for i in range(11):
 for j in range(18):
        if (matrix[i][j]==-1):
             print("NA", end = "   ")
        else:
             if(j==12):
                  print(str(matrix[i][j]).zfill(5), end = "  ")
             elif(j==11):
                  print(" | ",str(matrix[i][j]).zfill(4), end = "   ")
             else:
```

```
                print(str(matrix[i][j]).zfill(3), end = "  ")
print()

#Top 6 players :
print("\nTop players : \nID        Avg.")
for i in range(5,11):
 print(matrix[i][0] , "  " , str(matrix[i][12]).zfill(4))
```

OUTPUT :

```
ID      I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  | Total  Avg    Max <50 <100 <150 <200
P388480 146 024 117 076 069 034 115 175 127 024  |  0907  090.7  175 003 002 004 001
P542184 188 110 046 149 004 136 081 034 041 154  |  0943  094.3  188 004 001 003 002
P867852 055 086 129 025 194 102 134 160 029 095  |  1009  100.9  194 002 003 003 002
P953881 042 170 068 159 070 096 104 125 180 140  |  1154  115.4  180 001 003 003 003
P594608 042 046 029 163 045 193 075 040 183 144  |  0960  096.0  193 005 001 001 003
P880308 166 103 009 098 130 168 037 146 107 136  |  1100  110.0  168 002 001 005 002
P305045 176 045 010 175 077 027 083 055 151 031  |  0830  083.0  176 004 003 000 003
P234358 141 081 186 134 037 148 093 152 056 160  |  1188  118.8  186 001 003 003 003
P789209 108 109 107 146 051 060 129 157 083 190  |  1140  114.0  190 000 003 005 002
P996090 101 015 142 121 187 057 179 157 175 092  |  1226  122.6  187 001 002 003 004
P512605 174 121 136 030 040 142 038 040 084 137  |  0942  094.2  174 004 001 004 001


Players in their avg. score's increasing order :

ID      I1  I2  I3  I4  I5  I6  I7  I8  I9  I10  | Total  Avg    Max <50 <100 <150 <200
P305045 176 045 010 175 077 027 083 055 151 031  |  0830  083.0  176 004 003 000 003
P388480 146 024 117 076 069 034 115 175 127 024  |  0907  090.7  175 003 002 004 001
P512605 174 121 136 030 040 142 038 040 084 137  |  0942  094.2  174 004 001 004 001
P542184 188 110 046 149 004 136 081 034 041 154  |  0943  094.3  188 004 001 003 002
P594608 042 046 029 163 045 193 075 040 183 144  |  0960  096.0  193 005 001 001 003
P867852 055 086 129 025 194 102 134 160 029 095  |  1009  100.9  194 002 003 003 002
P880308 166 103 009 098 130 168 037 146 107 136  |  1100  110.0  168 002 001 005 002
P789209 108 109 107 146 051 060 129 157 083 190  |  1140  114.0  190 000 003 005 002
P953881 042 170 068 159 070 096 104 125 180 140  |  1154  115.4  180 001 003 003 003
P234358 141 081 186 134 037 148 093 152 056 160  |  1188  118.8  186 001 003 003 003
P996090 101 015 142 121 187 057 179 157 175 092  |  1226  122.6  187 001 002 003 004

Top players :
ID        Avg.
P867852   100.9
P880308   110.0
P789209   114.0
P953881   115.4
P234358   118.8
P996090   122.6
```

Run Succeeded | Time 78 ms | Peak Memory 7.6M