

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

```
2.#!/usr/bin/env python3
3.
4.import math
5.
6.def cal_gaussian(s,x,m):
7.    answer = float(1)
8.    answer /= (s*math.sqrt(2*math.pi))
9.    answer *= math.exp((-1/2)*((x-m)/s)**2)
10.    return answer
11.
12.s = float(0)
13.while(s<1):
14.    s = float(input("Enter s : "))
15.    if(s<1):
16.        print("Value of s must be greater than zero.")
17.x = float(input("Enter x : "))
18.m = float(input("Enter m : "))
19.print("The answer is : " , cal_gaussian(s, x, m))
```

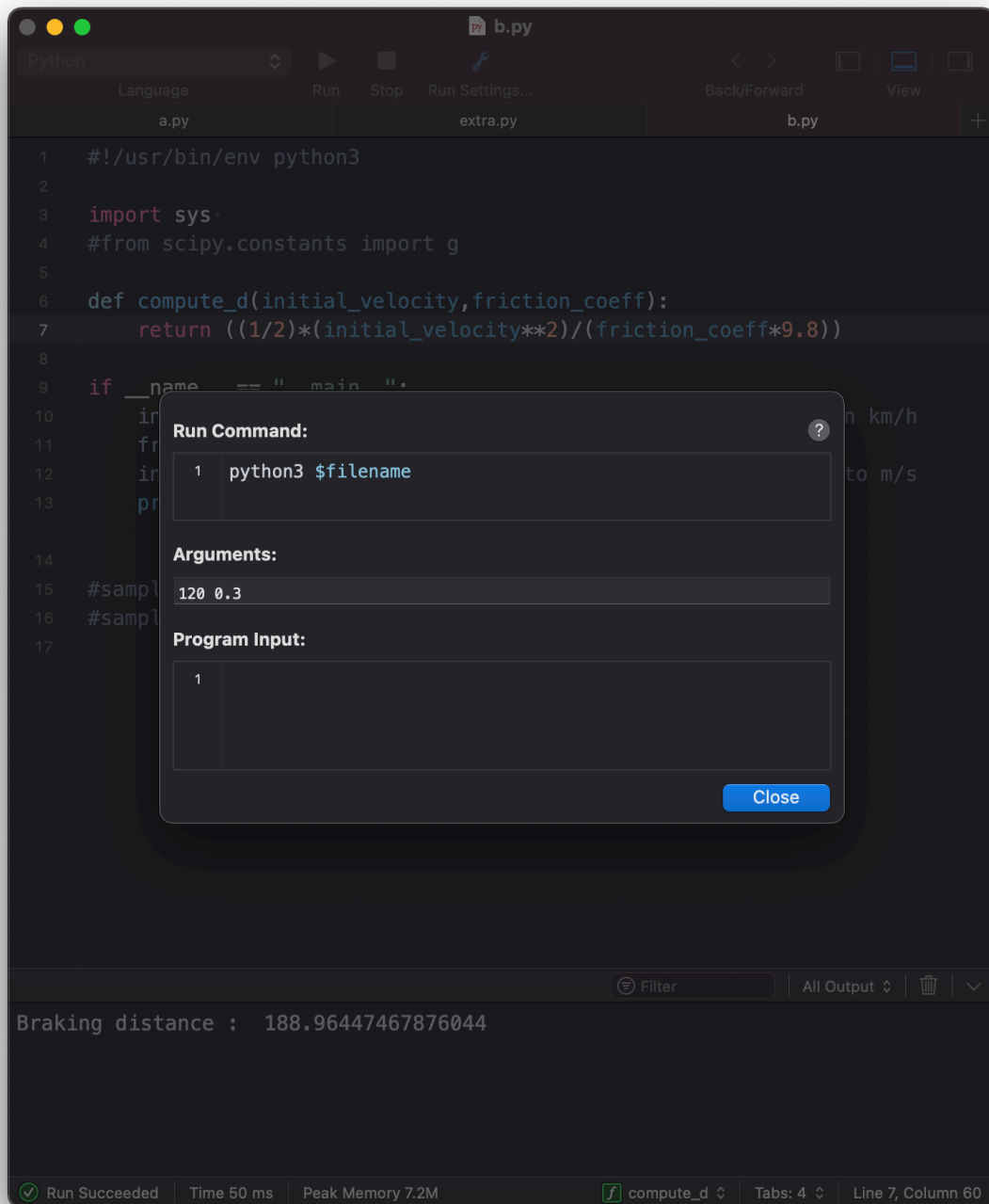
OUTPUT :

```
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 $\mu = 0.3$ (μ is dimensionless).

```
1. #!/usr/bin/env python3
2.
3. import sys
4. from scipy.constants import g
5.
6. def compute_d(initial_velocity, friction_coeff):
7.     return ((1/2)*(initial_velocity**2)/(friction_coeff*g))
8.
9. if __name__ == "__main__":
10.    initial_velocity = float(sys.argv[1])    #to be enetered
        in km/h
11.    friction_coeff = float(sys.argv[2])
12.    initial_velocity = initial_velocity/3.6 #converting it
        into m/s
13.    print("Braking distance : ", compute_d(initial_velocity,
        friction_coeff))
14.
15. #sample input 1 : 120 0.3
16. #sample input 2 : 50 0.3
```

OUTPUT :

The screenshot shows a Python IDE window with a file named `b.py`. The code in the editor is as follows:

```
1  #!/usr/bin/env python3
2
3  import sys
4  #from scipy.constants import g
5
6  def compute_d(initial_velocity,friction_coeff):
7      return ((1/2)*(initial_velocity**2)/(friction_coeff*9.8))
8
9  if __name__ == "__main__":
10     ir
11     fr
12     ir
13     pr
14
15     #sampl
16     #sampl
17
```

A modal dialog box is open in the center of the IDE, titled "Run Command:". It contains three sections:

- Run Command:** A text field with the command `python3 $filename`.
- Arguments:** A text field with the arguments `120 0.3`.
- Program Input:** A text field with the input `1`.

A "Close" button is located at the bottom right of the dialog box.

Below the dialog box, the output of the program is displayed in a terminal-like window:

```
Braking distance : 188.96447467876044
```

The status bar at the bottom of the IDE indicates that the run was successful, with a time of 50 ms and a peak memory usage of 7.2M. The current cursor position is at line 7, column 60.

(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.

```
1.#!/usr/bin/env python3
2.
3. #total, highest, avg =>Histogram
4.
5. import random
6. from random import randrange
7. import array as arr
8.
9. import array as arr
10. #a = arr.array('i', [])
11. matrix = []
12.
13. #Generating unique ids through set
14. set_of_ids = set()
15. while len(set_of_ids)<11:
16.     set_of_ids.add(random.randint(1, 999999))
17.
18. # Here -1 is for NA.
19.
20. #avg_and_max_of_all = []
21.
22. #ID i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 Total avg Max <50
    (50,100) (100,150) (150,200)
23.
24. #generating 10 innings data
25. for i in range(11):          # A for loop for row entries
26.     a = ['P' + str(set_of_ids.pop()).zfill(6)]
27.     count = 0;
28.     count_less_than_50 = 0
29.     count_less_than_100 = 0
30.     count_less_than_150 = 0
31.     count_less_than_200 = 0
32.     total_score = 0;
33.     max_score = 0;
34.     matches_played = 0;
35.     for j in range(1,11):    # A for loop for column
        entries
```

```

36.         number = random.randint(-1, 200)
37.         while (count>=2 and number== -1):
38.             number = random.randint(-1, 200)
39.
40.         if(number != -1):
41.             if(number<50):
42.                 count_less_than_50 += 1
43.             elif(number<100):
44.                 count_less_than_100 += 1
45.             elif(number<150):
46.                 count_less_than_150 += 1
47.             else:
48.                 count_less_than_200 += 1
49.
50.             total_score += number
51.             matches_played += 1
52.             if(max_score<number):
53.                 max_score = number
54.         else:
55.             count+=1
56.         a.append(number)
57.
58.     a.append(total_score)      #appending total
59.     a.append(round(total_score/matches_played,1))
        #appending avg
60.     a.append(max_score) #appending max
61.     a.append(count_less_than_50)
62.     a.append(count_less_than_100)
63.     a.append(count_less_than_150)
64.     a.append(count_less_than_200)
65.
66. # avg_and_max = []
67. # avg_and_max.append(total_score/matches_played)
68. # avg_and_max.append(max_score)
69. # avg_and_max_of_all.append(avg_and_max)
70.     matrix.append(a)
71.
72.
73.
74. #Printing data
75. print("ID      I1  I2  I3  I4  I5  I6  I7  I8  I9  I10 |
      Total Avg      Max <50 <100 <150 <200")
76. for i in range(11):

```

```

77. for j in range(18):
78.     if (matrix[i][j]==-1):
79.         print("NA", end = " ")
80.     else:
81.         if(j==12):
82.             print(str(matrix[i][j]).zfill(5), end = "
")
83.         elif(j==11):
84.             print(" | ",str(matrix[i][j]).zfill(4),
end = " ")
85.         else:
86.             print(str(matrix[i][j]).zfill(3), end = "
")
87. print()
88.
89.#Printing avg and max data
90.#print("\n\nAVERAGE | MAX")
91.#for i in range(11):
92.# print(str(round(avg_and_max_of_all[i][0],1)).zfill(5),end
= " ")
93.# print(str(round(avg_and_max_of_all[i][1],2)).zfill(3),end
= "")
94.# print()
95.
96.#Top 6 players :
97.print("\n\nPlayers in their avg. score's increasing order :
\n")
98.matrix = sorted(matrix, key = lambda a_entry:a_entry[11])
99.print("ID      I1  I2  I3  I4  I5  I6  I7  I8  I9  I10 |
Total Avg      Max <50 <100 <150 <200")
100.for i in range(11):
101. for j in range(18):
102.     if (matrix[i][j]==-1):
103.         print("NA", end = " ")
104.     else:
105.         if(j==12):
106.             print(str(matrix[i][j]).zfill(5), end = "
")
107.         elif(j==11):
108.             print(" | ",str(matrix[i][j]).zfill(4),
end = " ")
109.         else:

```

```

110.                print(str(matrix[i][j]).zfill(3), end = "
    ")
111. print()
112.
113. #Top 6 players :
114. print("\nTop players : \nID          Avg.")
115. for i in range(5,11):
116.     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