

Cluster counting

CONTENTS

[Summary](#)[Learning objectives](#)[‘Clusters’ in 1 dimension](#)[Challenge](#)[Suggested steps](#)[Check your answers](#)[Example output](#)

Summary

You will develop a Python script that generates a random bit string and calculates the average size of clusters of 1s within the string. A "cluster" is defined as a contiguous group of 1s, and the script must account for circular boundaries (i.e., the bit string is wrapped around, so the last bit connects to the first). You will break the problem down into functions to generate the random bit string, identify clusters, and calculate the average cluster size.

Learning objectives

By the end of this lab, you should be able to:

- Generate random bit strings – Use Python libraries to create a string of random bits of a specified length.
- Understand and implement circular boundaries – Handle cases where the bit string wraps around, treating the first and last bits as adjacent.
- Identify clusters of contiguous elements – Use looping and conditional logic to identify contiguous clusters of 1s within a bit string.
- Use modular programming techniques – Break down a problem into smaller, reusable functions (e.g., bit string generation, cluster identification, and average size calculation).
- Calculate averages – Use basic arithmetic to compute the average size of clusters.

'Clusters' in 1 dimension

In a random bit string of length N (e.g. $N = 10$), consider the example string:

```
console output
```

```
1010111101
```

In such a string, we often find consecutive sequences (clusters) of 1s. These clusters are defined as contiguous segments of 1s, and the size of a cluster is simply the number of 1s in that contiguous segment.

Assuming zero-indexing (where the first bit in the string is at index 0) we notice that:

- Element 2 contains a 1, but it is not adjacent with any other 1s, so it forms a cluster of size 1.
- Elements 4, 5, 6, and 7 all contain 1s and are adjacent to one another, forming a cluster of size 4.
- Elements 9 and 0 contain 1s, and since we assume the bit string boundary is wrapped (i.e., the string is circular), they form a cluster of size 2.

So, in total, we have 3 distinct clusters of 1s in the string:

- A cluster of size 1 at index 2.
- A cluster of size 4 at indices 4-7.
- A cluster of size 2 at indices 9 and 0 (wrapped around).

Challenge

Your task is to write a Python script named `clusters.py` that does the following:

- Generates a random bit string of length n . The string should consist of 0s and 1s. Set n in a variable at the start of a `main()` function (for example, $n = 20$) and pass it to the functions that need it as an argument. Do not use global variables: every variable should live inside a function.
- Identifies all clusters of contiguous 1s in the generated bit string, accounting for the circular nature of the string.

- Calculates the average cluster size of 1s in the bit string. The average cluster size is defined as the total number of 1s divided by the number of distinct clusters of 1s.
- Handles the two special cases: a string with no 1s has no clusters (report “No clusters” instead of dividing by zero), and a string with no 0s is a single cluster of size n.
- The script should make use of functions to modularize the code (e.g., generating the bit string, identifying clusters, calculating the average cluster size).
- Feel free to implement the solution using any approach, but ensure that you break down the functionality into clear reusable functions.

SUGGESTED STEPS

- Write and test the function that generates the random bit string.
- Find the clusters without any wrapping first: walk along the string, start a new cluster when you meet a 1 and close the cluster when you meet a 0.
- Then deal with the wrap. Hint: if you begin your walk just after a 0, and use the % operator to bring the index back to 0 at the end of the string, no cluster can be cut in two by your starting point.
- Calculate the average cluster size and print the results.

CHECK YOUR ANSWERS

Before running your script on random strings, test your cluster function on strings whose answers you know:

- 1010111101 has 3 clusters, of sizes 1, 4 and 2 (average 2.33)
- 00110111 has 2 clusters, of sizes 2 and 3 (average 2.5)
- 1001 has 1 cluster of size 2, which wraps around
- 0000 has no clusters
- 1111 has 1 cluster of size 4

EXAMPLE OUTPUT

Your output does not need to match this exactly, but it should show the bit string, the clusters found and the average cluster size:

```
8 of 8 tests passed
-----
Bit string length = 20
00000000000000000000
No clusters
-----
Bit string length = 20
11111111111111111111
Number of clusters: 1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Average cluster size = 20.0
-----
Bit string length = 20
01111111111111111111
Number of clusters: 1
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Average cluster size = 19.0
-----
Bit string length = 20
11101101100011000001
Number of clusters: 4
[4, 5]
[7, 8]
[12, 13]
[19, 0, 1, 2]
Average cluster size = 2.5
-----
Bit string length = 20
01100011010000110011
Number of clusters: 5
[1, 2]
[6, 7]
[9]
[14, 15]
[18, 19]
Average cluster size = 1.8
-----
```