

Advanced Algorithms

How many McDonald stores are there on the
Hong Kong island?



According to GPT 5.4, there are 45 McDonald stores on the Hong Kong island.

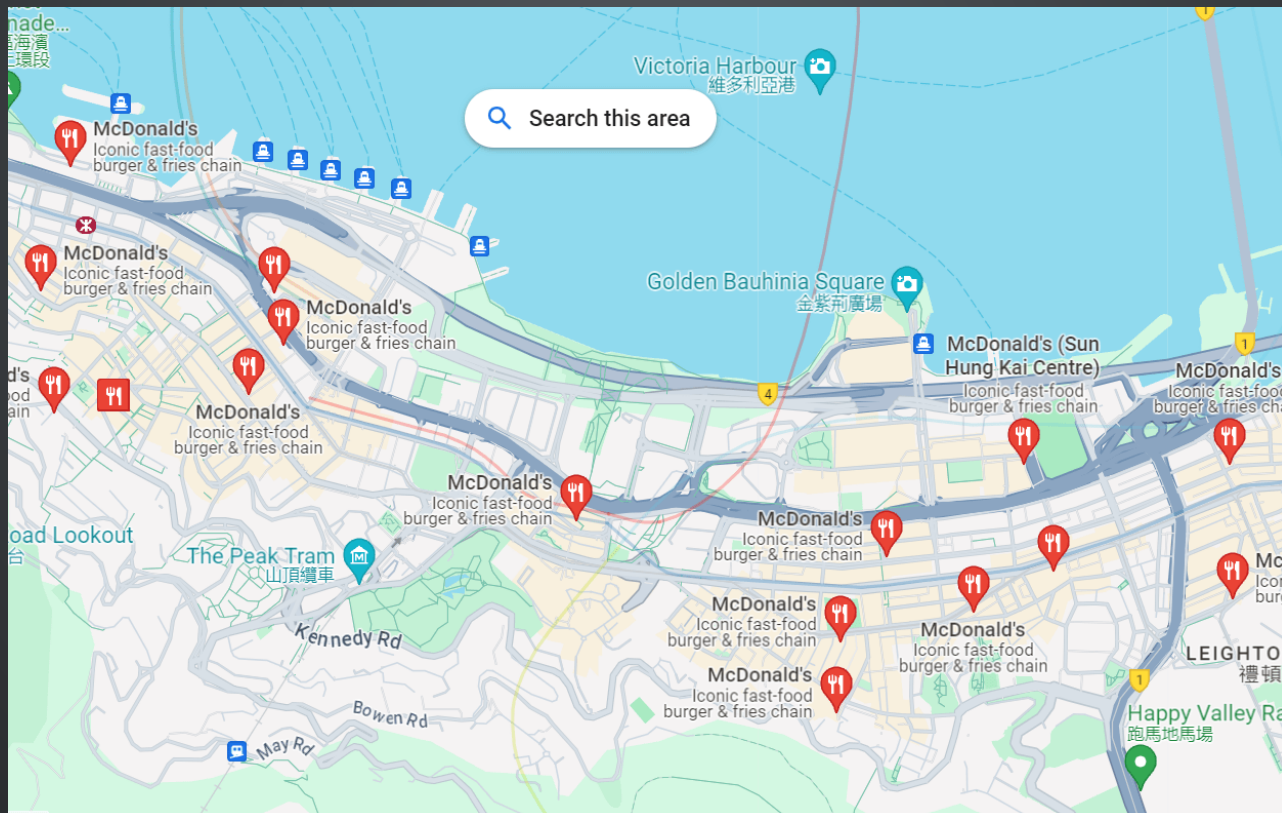
How many McDonald stores are there on Hong Kong island

3 steps completed >

There are **31 McDonald's restaurants** on Hong Kong Island based on the current address list on McDonald's Hong Kong site, counting the branches listed under Central/Western, Wanchai, Eastern, and Southern districts: $9 + 9 + 9 + 4 = 31$. `mcdonalds`

If you were the CEO of McDonald Hong Kong, how would you choose the locations of these stores?

Why is there only one McDonald close to Admiralty station?



Clustering

Imaging that you are designing T-shirts for consumers. You are looking at your consumers' weight and height, which would allow you to decide how many sizes to offer and which size fits a particular individual. Instead of classifying your consumers arbitrarily, you can use data to perform the task more precisely.

SIZE SELECTION. 尺码选择表

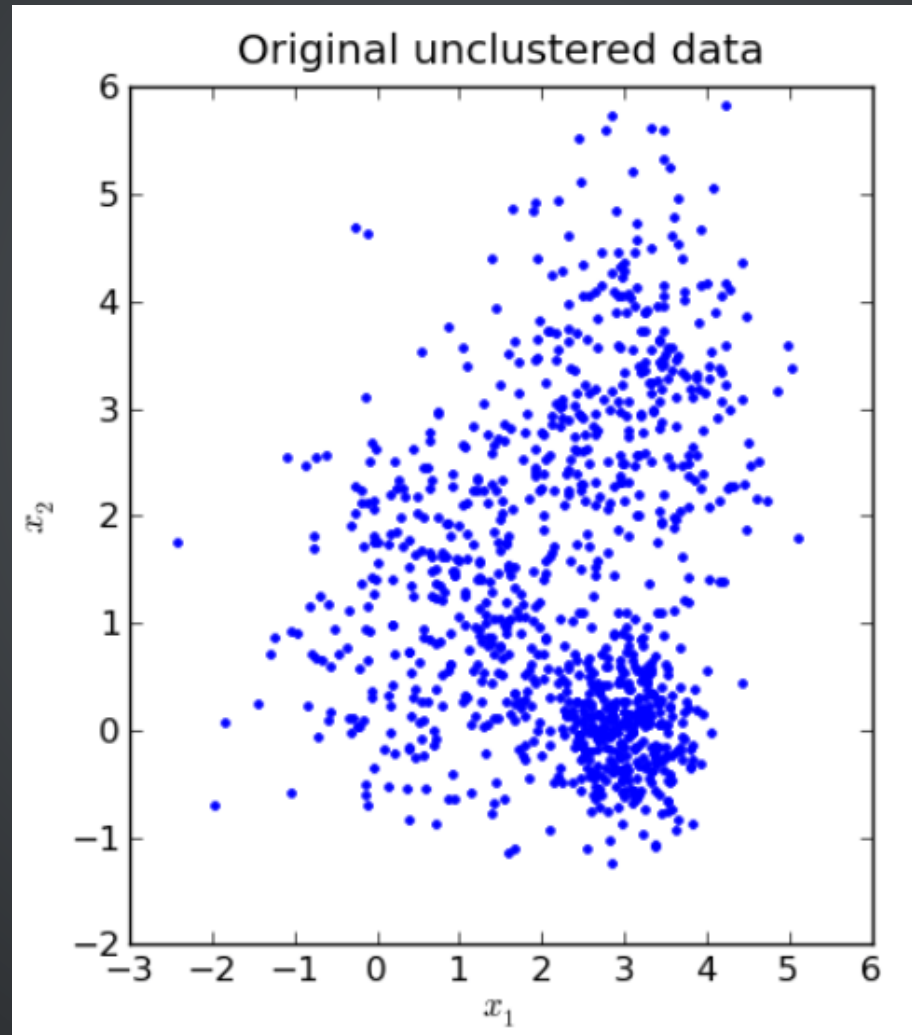
本款衣服为标准版型，喜欢修身的选小一码，喜欢宽松的选大一码

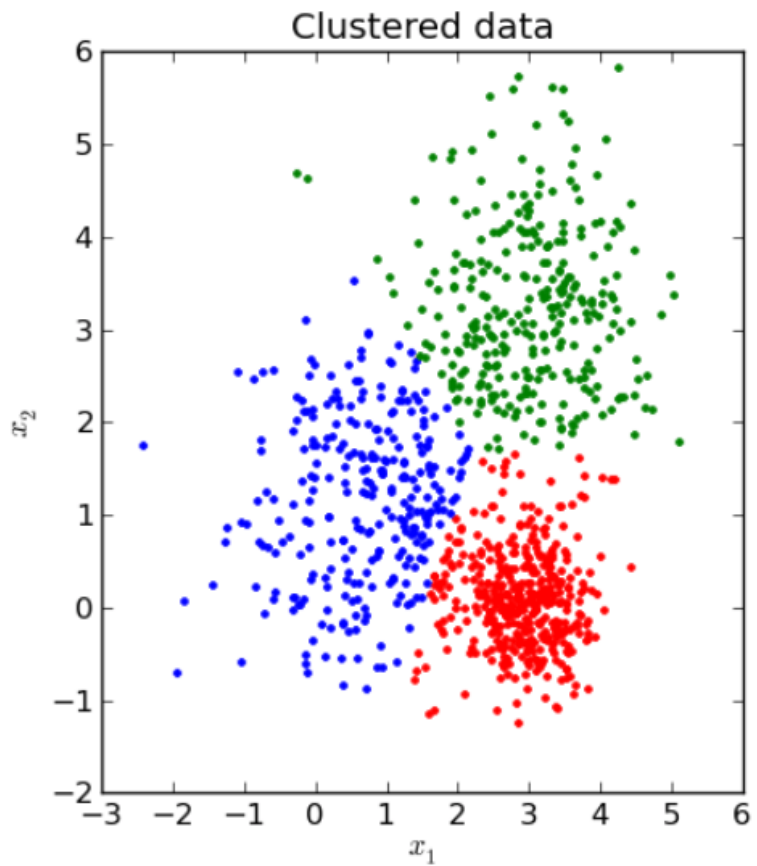
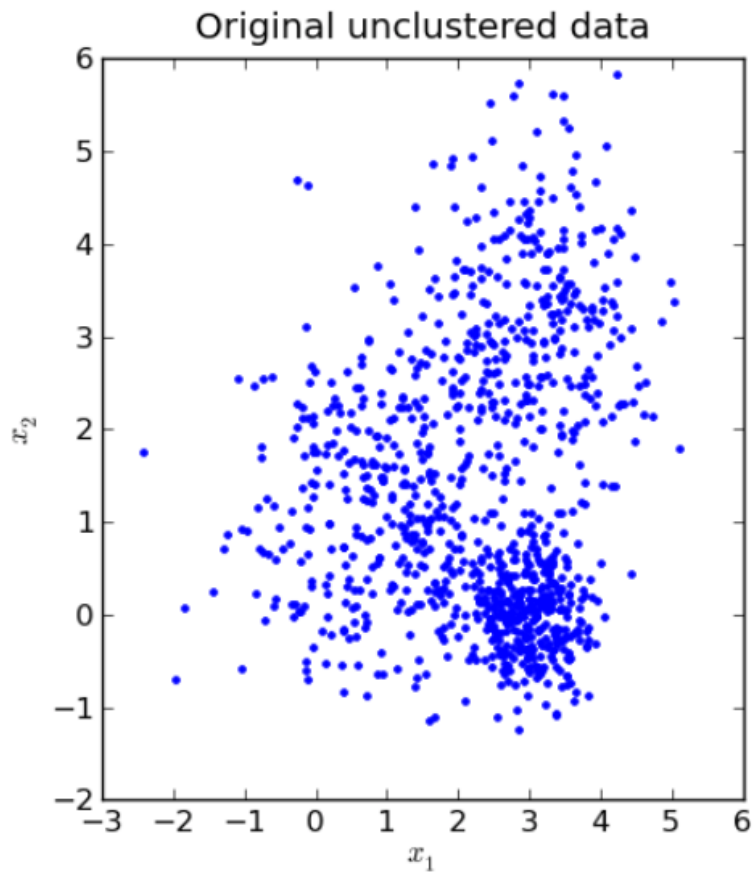
身高 (cm) \ 体重 (斤)	95	105	115	125	135	145	155	165	175	185	195	205
165	S											
170												
175												
180												
185												
190												
195												

Why offer 7 sizes? Are these sizes optimal?

Image credit: taobao.com

Here is consumer data. How would you classify them into groups?





The K -means Algorithm

The K -means algorithm an EM (expectation-maximization) algorithm commonly used for classifying objects.

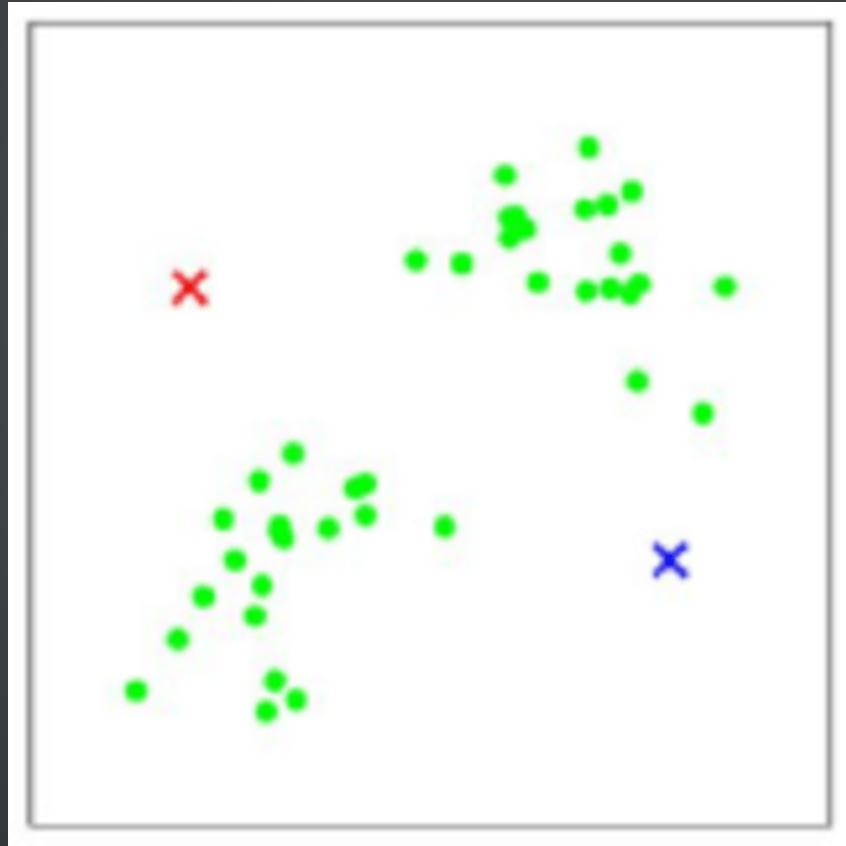
Input: A number of observations $(X_1, X_2, \dots, X_n)$ and k , the number of groups to be classified

Output: k mutually exclusive and collectively exhaustive groups containing all observations

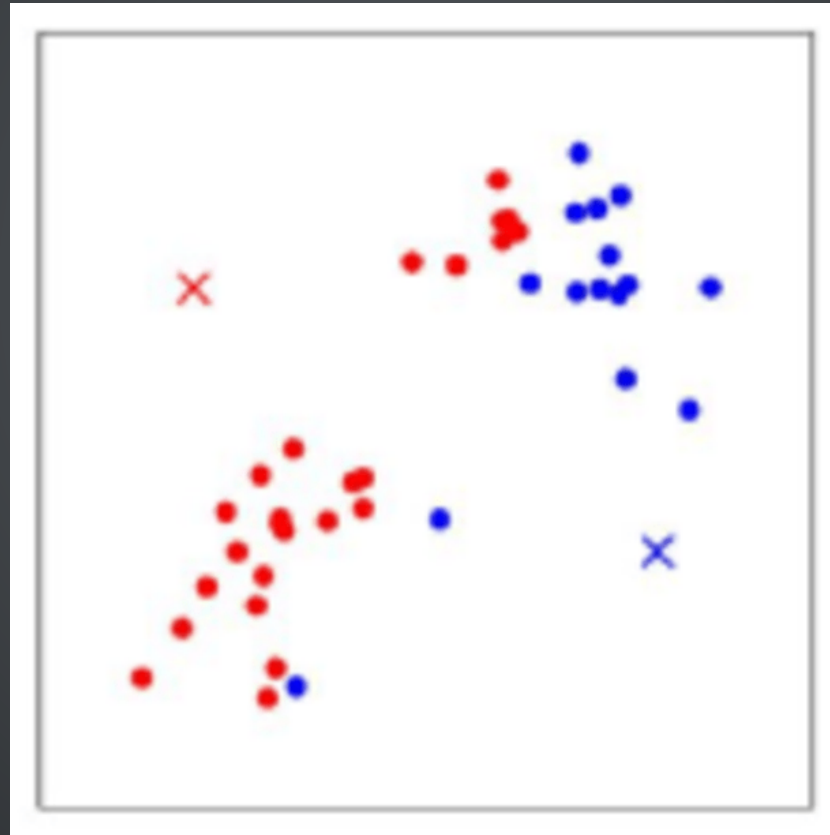
Classify the following observations into $k = 2$ groups:



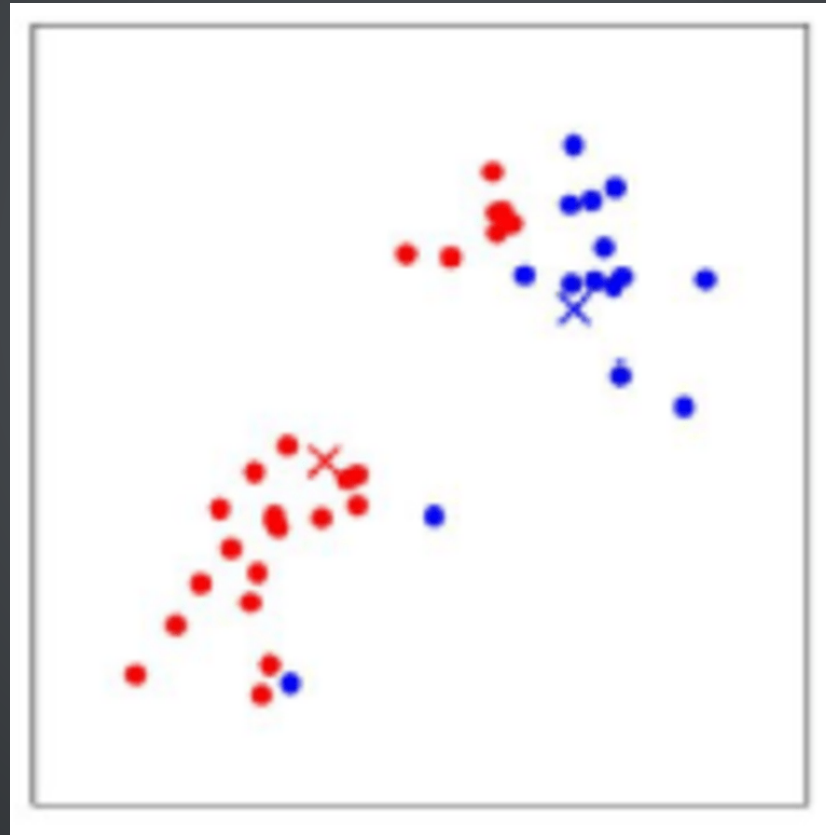
Step 1: Random choose $k = 2$ “centers” for your clusters.



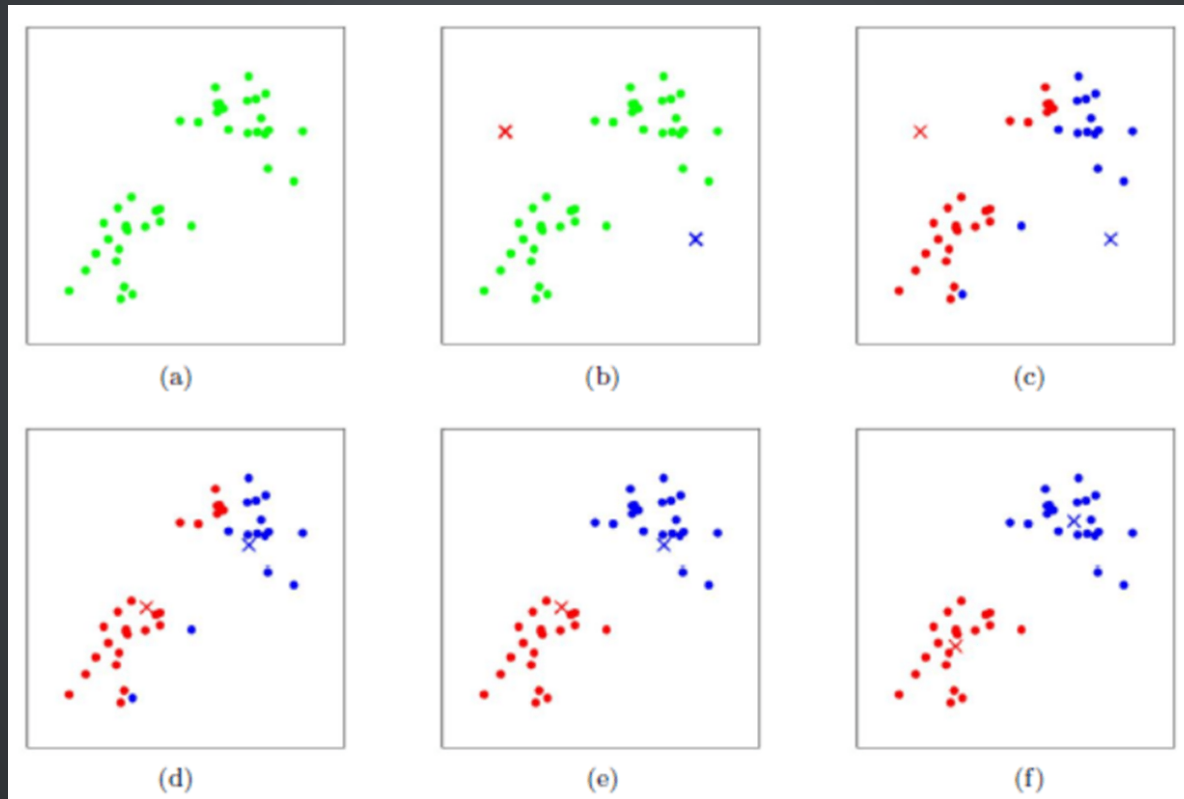
Step 2: Assign each observation to the nearest center.



Step 3: Update the location the centers, which is given by the average location of all points in the corresponding cluster.



Repeat the above process again and again until the centers no longer change.



The K -means algorithm:

1. Select cluster centers $c_1, \dots, c_k \in \mathbb{R}^d$ arbitrarily.
2. Assign every $x \in \mathcal{X}$ to the cluster $\mathcal{C}_i$ whose cluster center c_i is closest to it, i.e., $\|x - c_i\| \leq \|x - c_j\|$ for all $j \neq i$.
3. Set $c_i = \frac{1}{|\mathcal{C}_i|} \sum_{x \in \mathcal{C}_i} x$.
4. If clusters or centers have changed, goto 2. Otherwise, terminate.

https://www.youtube.com/embed/R2e3Ls9H_fc?enablejsapi=1

Performing *K*-means with Python:

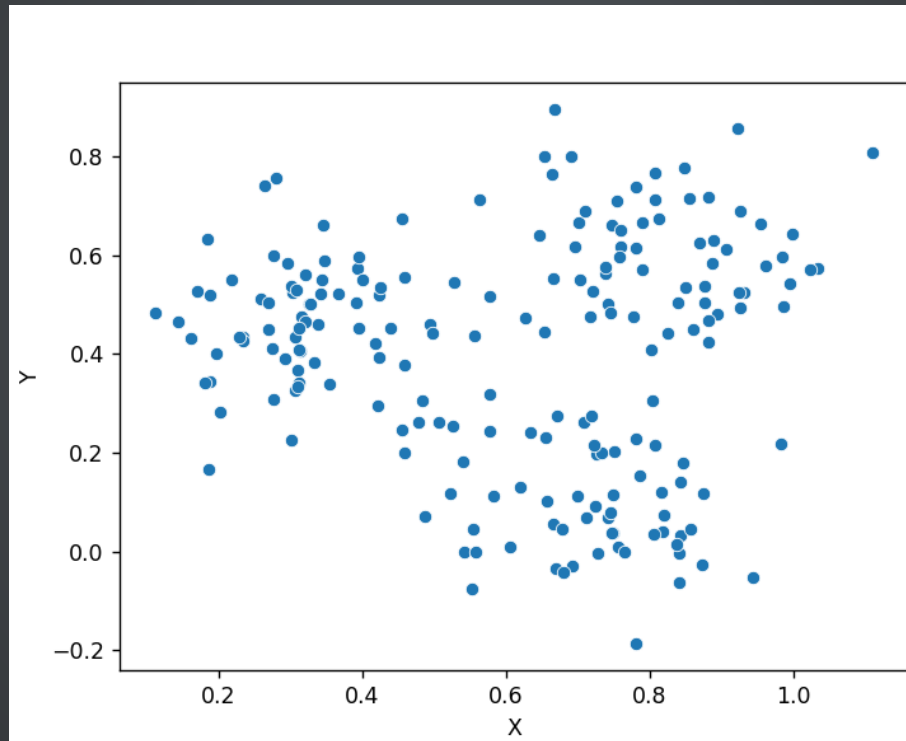
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from sklearn.cluster import KMeans
5 from sklearn.metrics import silhouette_score
6 data = pd.read_csv('https://ximarketing.github.io/data/clustering.csv')
7 print(data)
```

	X	Y
0	0.627123	0.473972
1	0.562636	0.714202
2	0.295934	0.583077
3	0.815801	0.121524
4	0.345354	0.662115

The input is a CSV file containing the X and Y coordinates of some individuals.

```
1 sns.scatterplot(data = data, x = 'X', y = 'Y')  
2 plt.show()
```

Let's visualize these locations!



How many groups do they belong to?

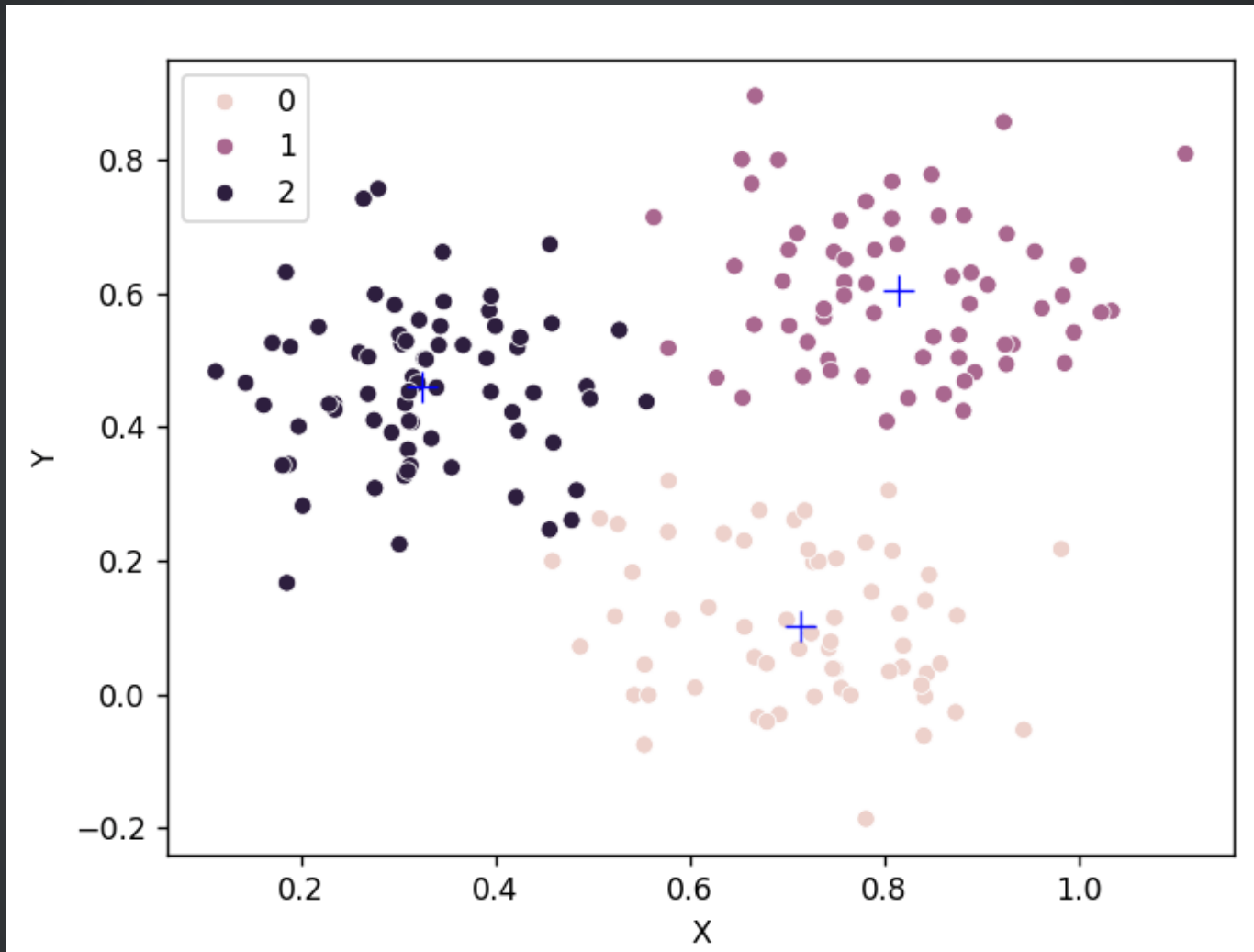
```
1 kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
2 kmeans.fit(data)
3 sns.scatterplot(x = data['X'], y = data['Y'], hue = kmeans.labels_)
4 centers = kmeans.cluster_centers_
5 sns.scatterplot(x = centers[:, 0], y = centers[:, 1],
6                 color = 'blue', s=100, marker='+')
7 plt.show()
```

Now, we use the KMeans function to classify the observations. If you don't understand the code, ask AI:

What is the meaning of random state in python's kmeans?

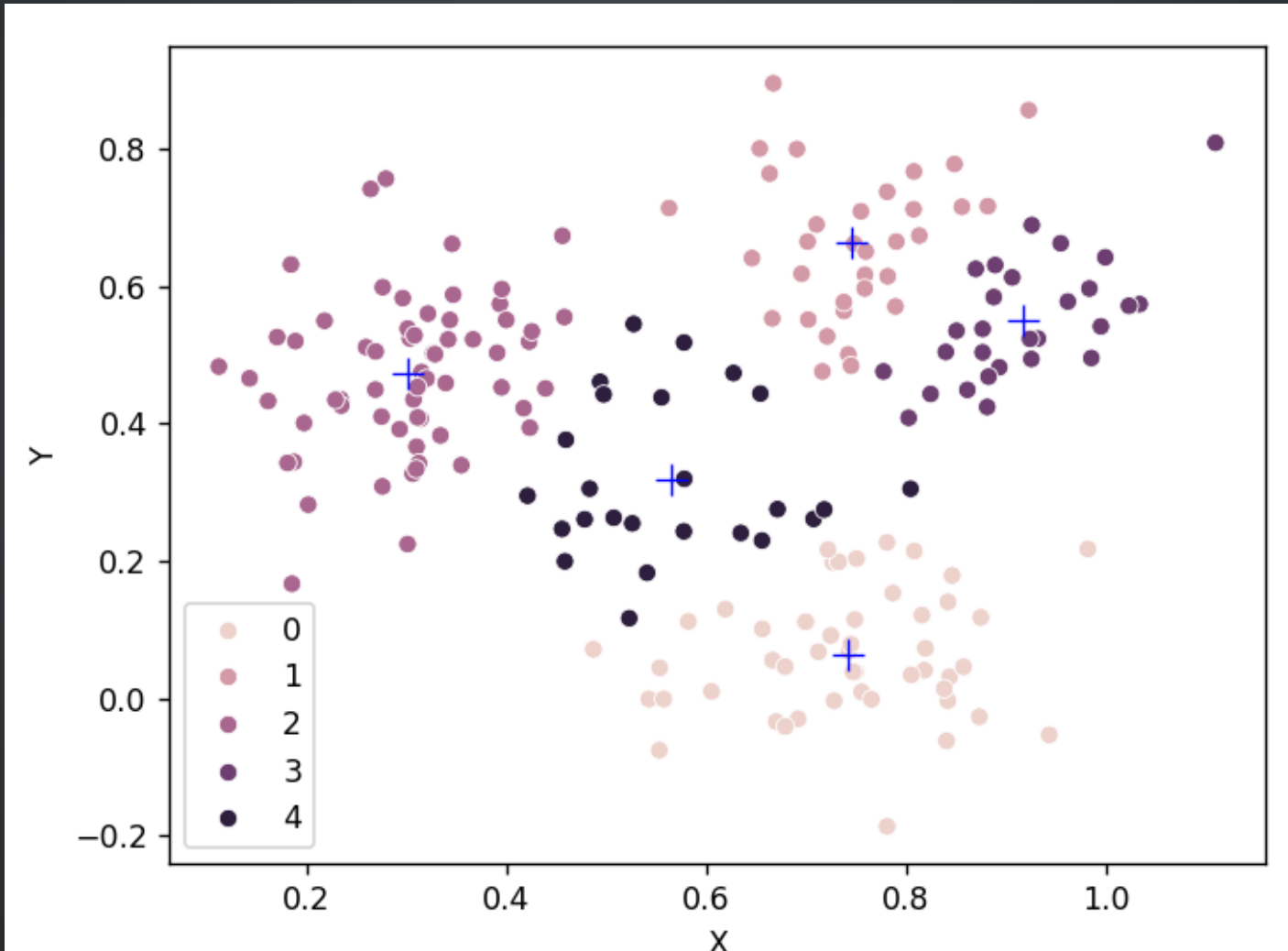
Reviewed 15 sources >

In Python's scikit-learn `KMeans`, `random_state` controls the random numbers used when choosing the initial cluster centers. If you set it to a fixed integer like `0` or `42`, you make the initialization deterministic, so the same data and settings will usually give the same result each time you run the code. [scikit-learn +1](#)



Does it look good?

Let's try five groups instead...



Exercise

Use AI to generate a desktop APP which demonstrates the K-means algorithm

Question:

What is the optimal number of clusters?

We use a measure called “Silhouette Score.”

It is done in the following way:

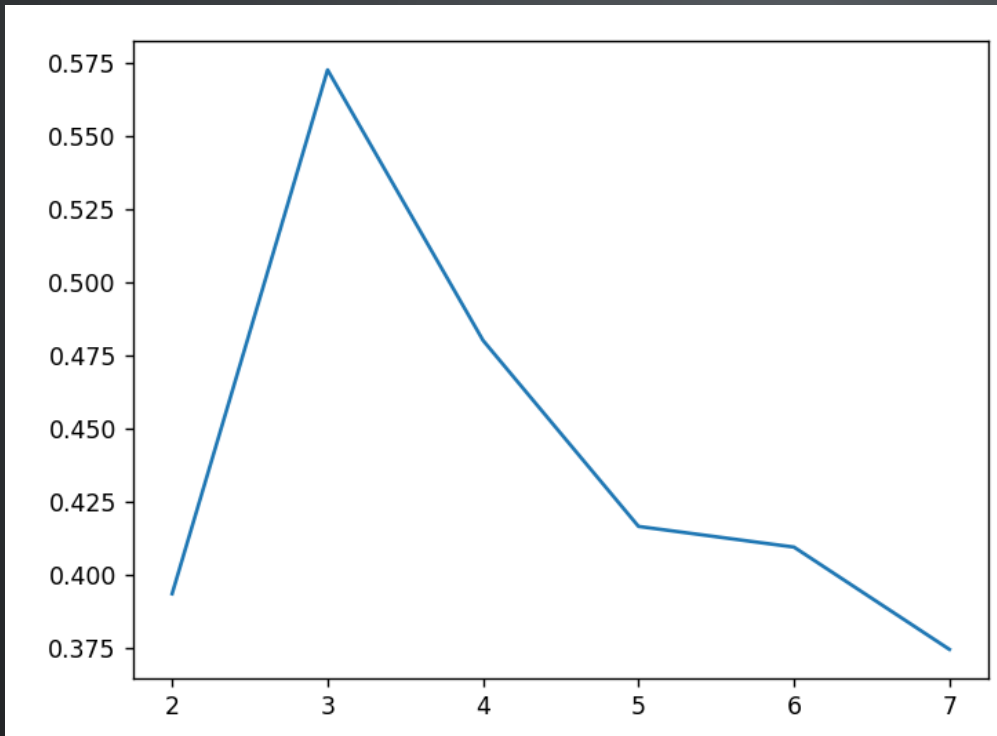
1. For each data point, calculate the average distance between the data point and all other points in the same cluster. This is called the intra-cluster distance.
2. For each data point, calculate the average distance between the data point and all other points in the nearest cluster. This is called the nearest-cluster distance.
3. Calculate the silhouette score for each data point as $(\text{nearest-cluster distance} - \text{intra-cluster distance}) / \text{nearest-cluster distance}$.
4. The overall silhouette score is the average of the silhouette scores for all data points.

We use a measure called “**Silhouette Score**.”

The idea is simple: Each point should be as close as possible to points in the same cluster, and as far as possible from points in other clusters.

It should be value between 0 and 1. The greater the score is, the better the algorithm performs.

```
1 K = range(2, 8)
2 fits = []
3 score = []
4 for k in K:
5     model = KMeans(n_clusters=k, random_state=0, n_init='auto').fit(data)
6     fits.append(model)
7     score.append(silhouette_score(data, model.labels_, metric='euclidean'))
8 sns.lineplot(x = K, y = score)
9 plt.show()
```



3 is the optimal number!

The complete code is here:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from sklearn.cluster import KMeans
5 from sklearn.metrics import silhouette_score
6 data = pd.read_csv('https://ximarketing.github.io/data/clustering.csv')
7 print(data)
8 sns.scatterplot(data = data, x = 'X', y = 'Y')
9 plt.show()
10 kmeans = KMeans(n_clusters = 5, random_state = 0, n_init='auto')
11 kmeans.fit(data)
12 sns.scatterplot(x = data['X'], y = data['Y'], hue = kmeans.labels_)
13 centers = kmeans.cluster_centers_
14 sns.scatterplot(x = centers[:, 0], y = centers[:, 1],
15                 color = 'blue', s=100, marker='+')
16 plt.show()
17 K = range(2, 8)
18 fits = []
19 score = []
20 for k in K:
21     model = KMeans(n_clusters=k, random_state=0, n_init='auto').fit(data)
22     fits.append(model)
23     score.append(silhouette_score(data, model.labels_, metric='euclidean'))
24 sns.lineplot(x = K, y = score)
25 plt.show()
```

The k -means algorithm can also be used for image compression. How could this be done?

How would you compress this [image](#)?



Image Compression

An image has millions of pixels. Each pixel is represented by its RGB color, e.g., (127, 176, 96).

For each pixel, you need to save a lot of information! There are $256 \times 256 \times 256 = 16,777,216$ possible color combinations for each pixel. This requires a lot of storage.

Image Compression

The idea is that we do not need so many colors! Some colors are very similar and we can combine them into a single color. For instance, consider two pixels, $(45, 80, 97)$ and $(46, 79, 98)$. Their colors are very much similar and we can combine them into a single color.

Image Compression

If we combine similar colors and come up with only 20 possible colors for the images, the size of the image will be much smaller! Previous, each pixel is represented by a vector (r_i, g_i, b_i) , where $0 \leq r_i, b_i, g_i \leq 255$. Now, each pixel is represented by simply one number $0 \leq x_i \leq 19$, where $x_i = 0$ denotes the first color, and so on.

The algorithm

- Each pixel is represented by a point in the 3D space, (r_i, g_i, b_i) .
- Classify the points into k clusters, call them cluster $j = 1, 2, \dots, k$.
- Calculate the centroid (i.e., center) of each cluster.
- Replace the color of each pixel with the color of the corresponding centroid, and when saving the image, only save the corresponding cluster number j .

Original Image



Compressed Image with 5 Colors



Original Image



Compressed Image with 10 Colors



Original Image



Compressed Image with 20 Colors



Original Image



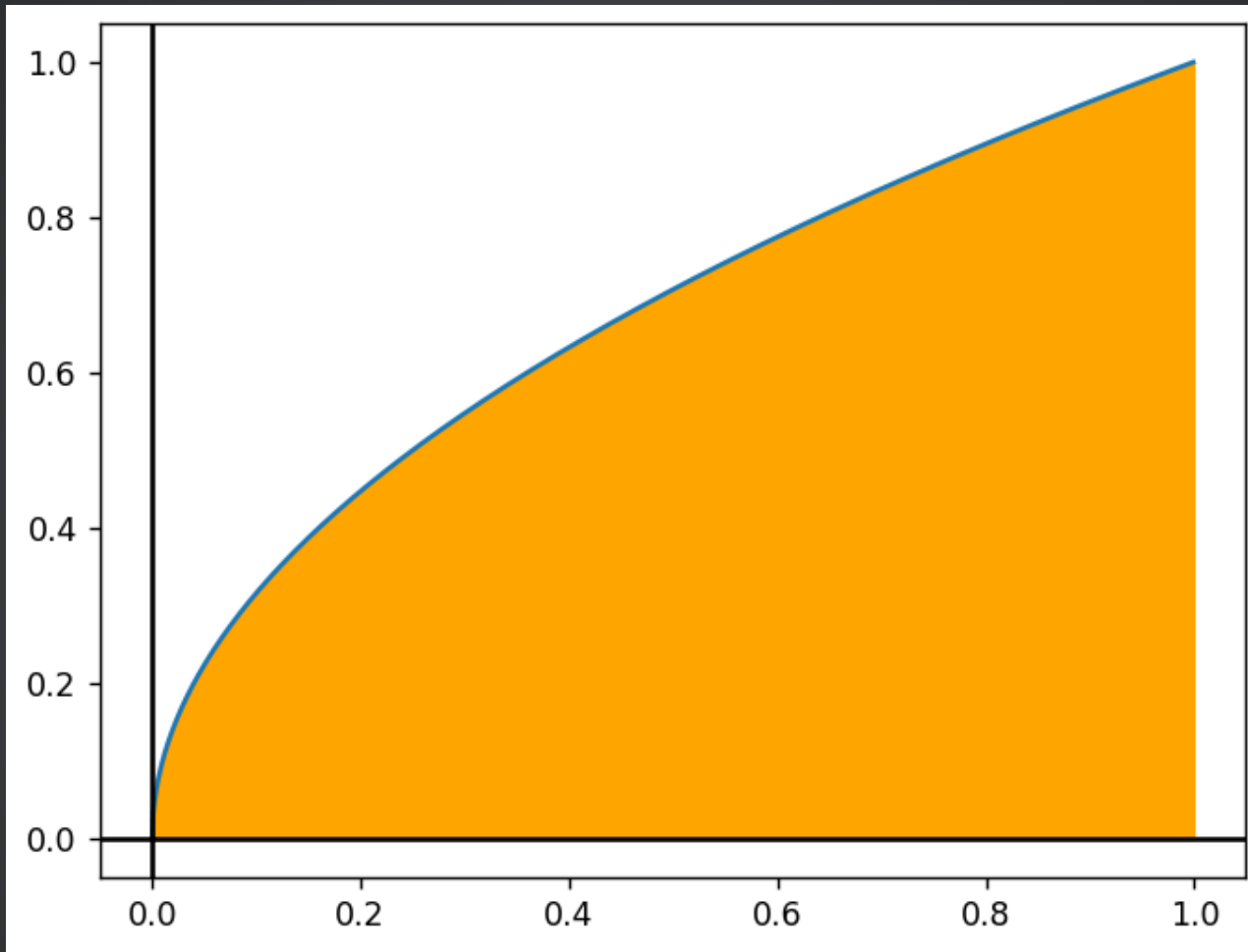
Compressed Image with 50 Colors



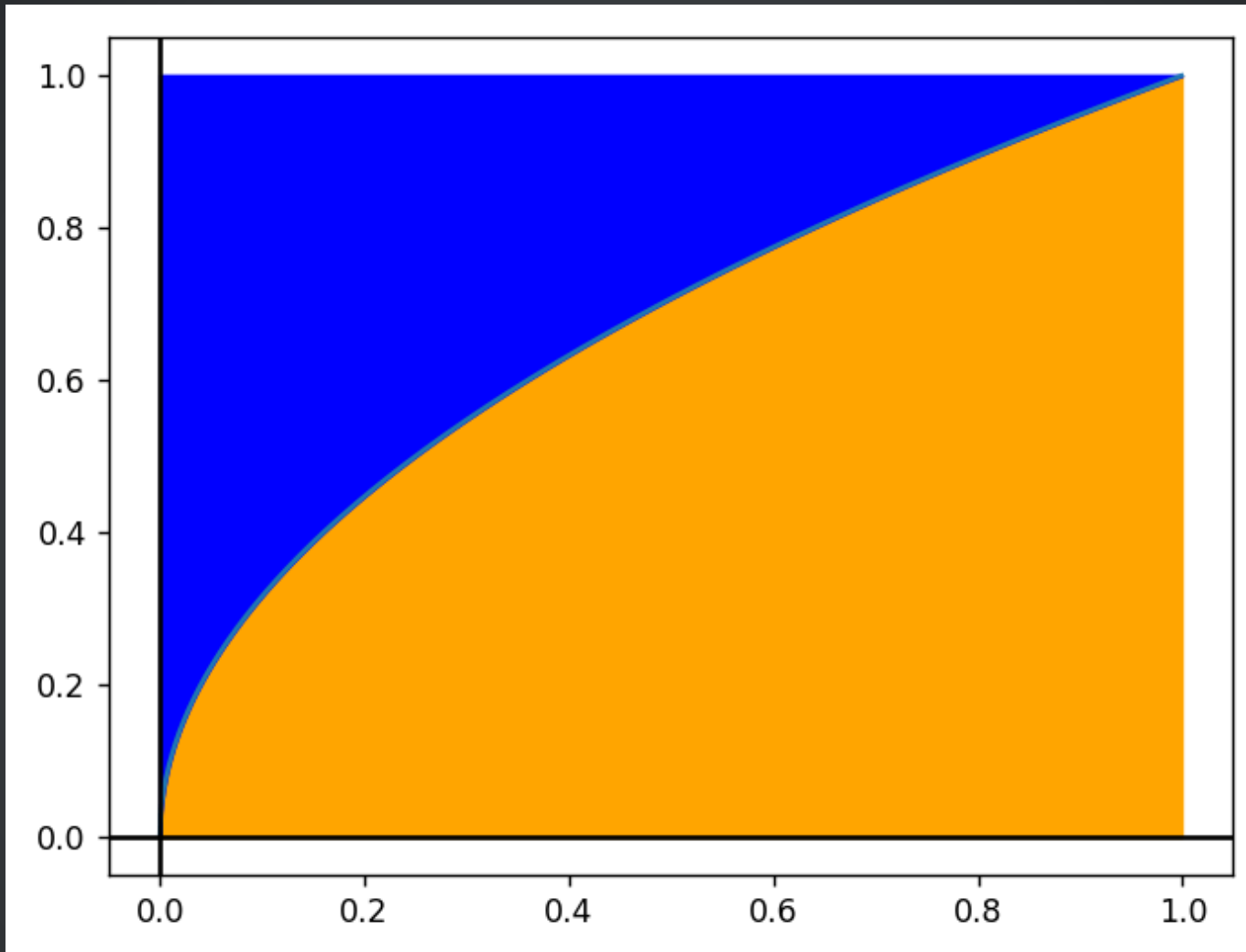
Calculate the following integral:

$$\int_0^1 \sqrt{x} dx$$

Monte Carlo Algorithm

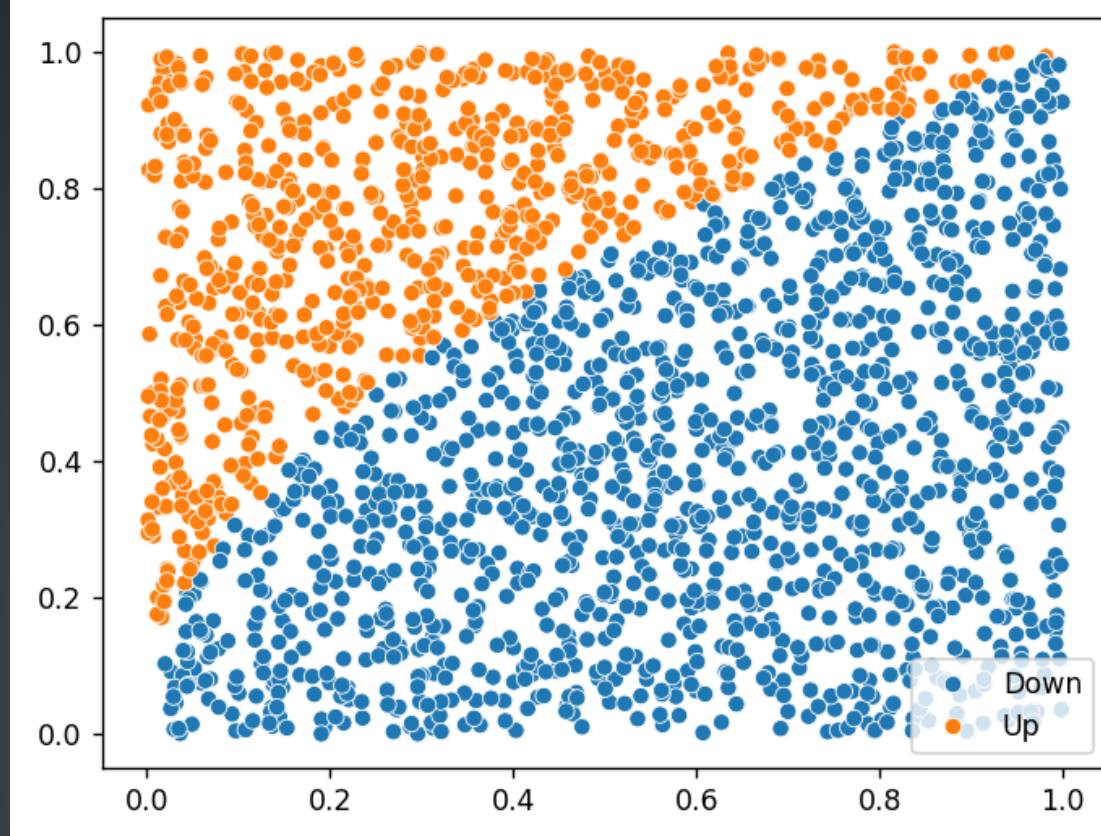


Integral is the area below the line $f(x) = \sqrt{x}$



The area size of blue + orange = 1

We want to calculate the fraction of the orange area



The idea is as follows. We random draw a large number of points in the area $[0, 1] \times [0, 1]$, and count how many points are above $\sqrt{x}$. The size can be calculated as follows:

$$\text{size of blue area} \approx \frac{\text{number of blue points}}{\text{total number of points}} \times 1$$

Algorithm:

1. Randomly choose K points that are uniformly distributed over $[0, 1] \times [0, 1]$. Denote these points by $(x_1, x_2), \dots, (x_K, y_K)$.
2. Calculate how many points satisfy $y_i < \sqrt{x_i}$, denote by count_below.
3. The size of the area is count_below / K .

The complete code is here:

```
1 import random
2 import math
3 K = 10000
4 count_below = 0
5 for i in range(0, K):
6     x = random.uniform(0, 1)
7     y = random.uniform(0, 1)
8     if y <= math.sqrt(x):
9         count_below = count_below + 1
10 print (count_below/K)
```

Exercise: Calculate the value of π .

Exercise: Calculate the value of π .

In 1733, the French mathematician Buffon first came up with a probabilistic method to calculate π numerically. He threw needles to parallel lines, and found that the probability that a needle crosses a line is a function of π . Based on that, he calculated the value of π .

In Chinese, this is known as “布丰投针”.

<https://www.youtube.com/embed/kazgQXaeOHk?enablejsapi=1>

The code for calculating π :

```
1 import random
2 K = 100000
3 count_inside = 0
4 for i in range(0, K):
5     x = random.uniform(-1, 1)
6     y = random.uniform(-1, 1)
7     if x**2+y**2<1:
8         count_inside = count_inside + 1
9 print(count_inside/K*4)
```

Exercise

Use AI to generate a desktop APP which demonstrates how to calculate π numerically.

Dynamic Programming

Question

You are climbing a staircase. It takes 10 steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

A More General Question

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

You can ask GPT.

Yes, this problem can be solved using dynamic programming.

Here is the solution:

This problem is similar to the Fibonacci number problem. If we denote $f(n)$ as the number of distinct ways to climb to the top, we have the following relationship:

$$f(n) = f(n-1) + f(n-2)$$

The base cases are $f(1) = 1$ and $f(2) = 2$, as there is one way to climb one step and two ways to climb two steps.

Therefore, we can calculate $f(n)$ using a bottom-up approach. Initialize the first and second numbers to 1 and 2, respectively, then use a loop to calculate the following numbers until n .

A More General Question

$n = 1$: Only 1 way to climb up; we write $f(1) = 1$.

$n = 2$: Only 2 ways to climb up; we write $f(2) = 2$.

$n = 3$: You can reach stair 3 by jumping from stair 1 or stair 2, and the solution is $f(3) = f(2) + f(1) = 3$.

In general: $f(n) = f(n - 1) + f(n - 2)$.

Algorithm:

Set $f(1) = 1, f(2) = 2$.

For $i = 3, 4, \dots, n$, we have

$$f(n) = f(n - 1) + f(n - 2)$$

Output $f(n)$.

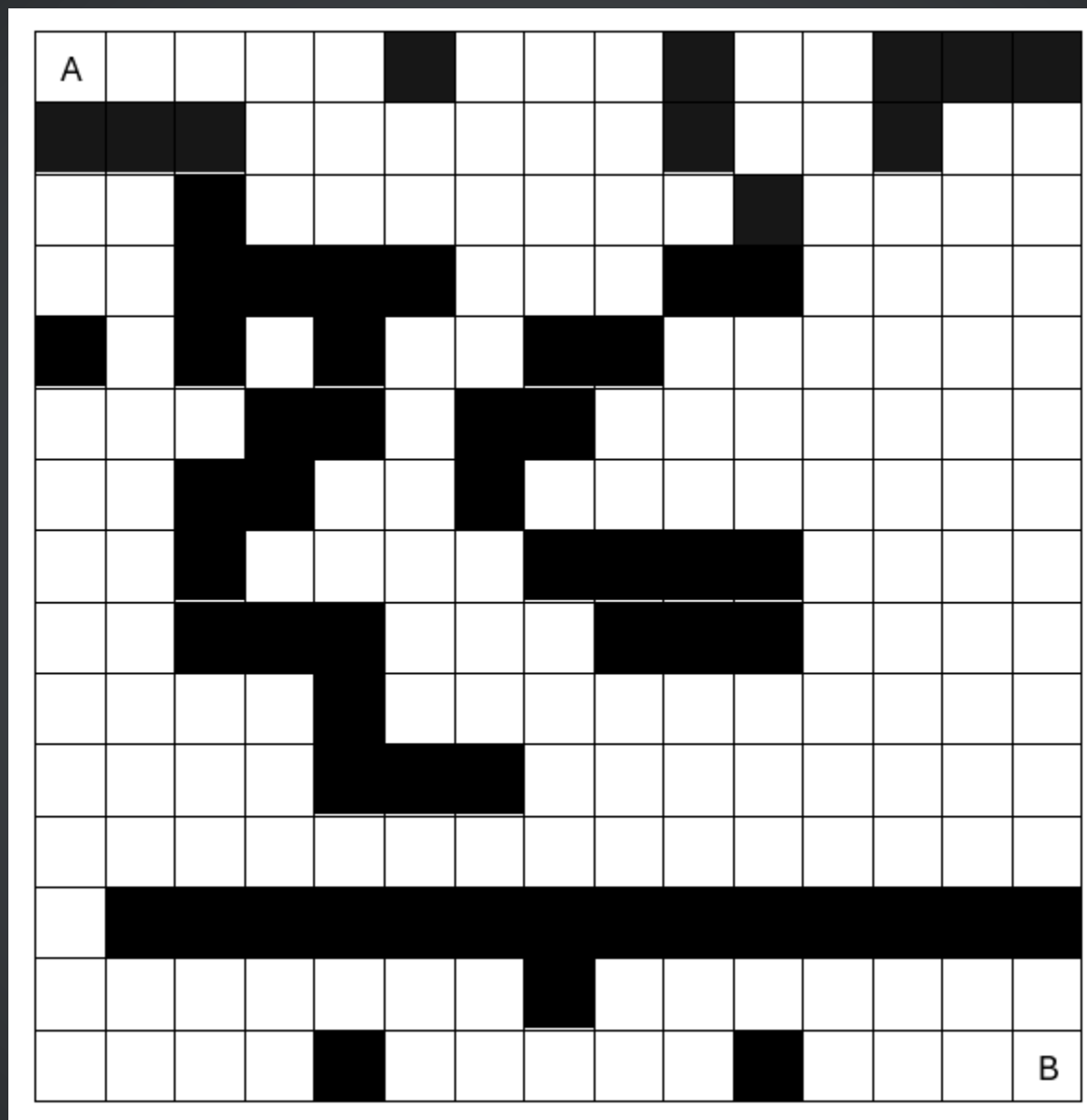
<https://www.youtube.com/embed/tOYZcy2IzJA?enablejsapi=1>

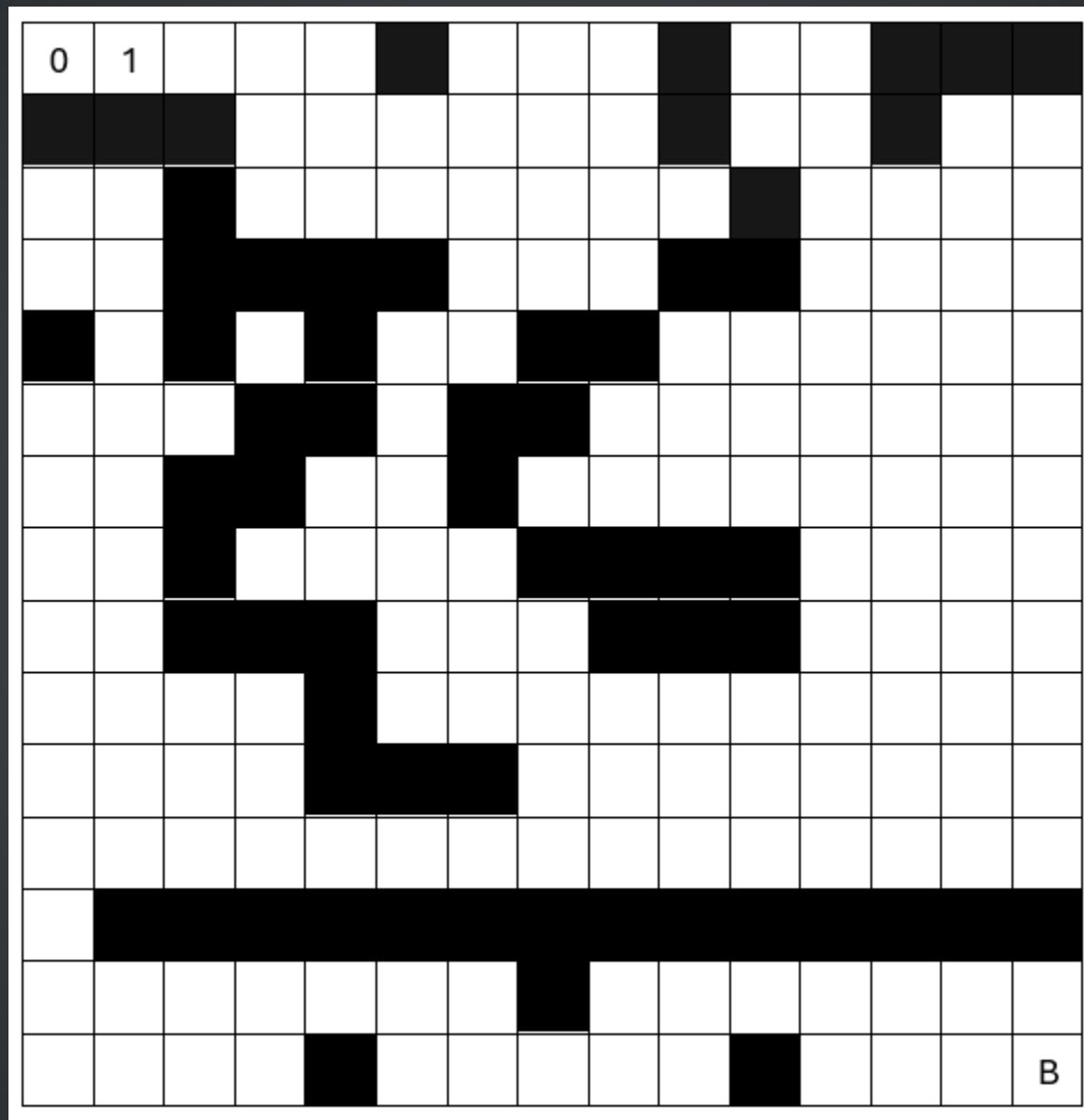
Exercise: Answer the question for $n = 10$.

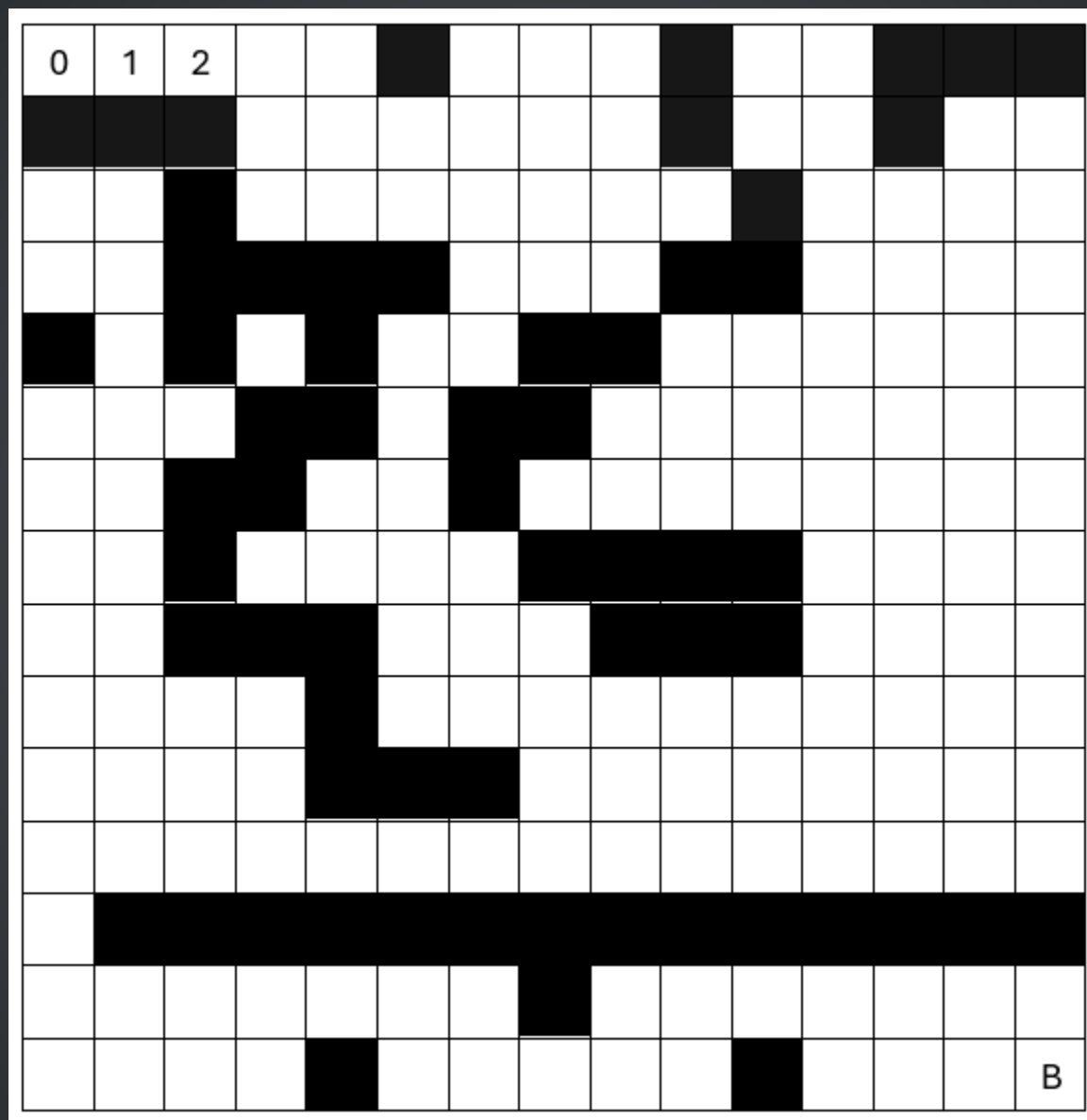
Sample Code:

```
1 list = []
2 list.append(1)
3 list.append(2)
4 for n in range(2, 10):
5     list.append(list[n-1] + list[n-2])
6 print(list[9])
```

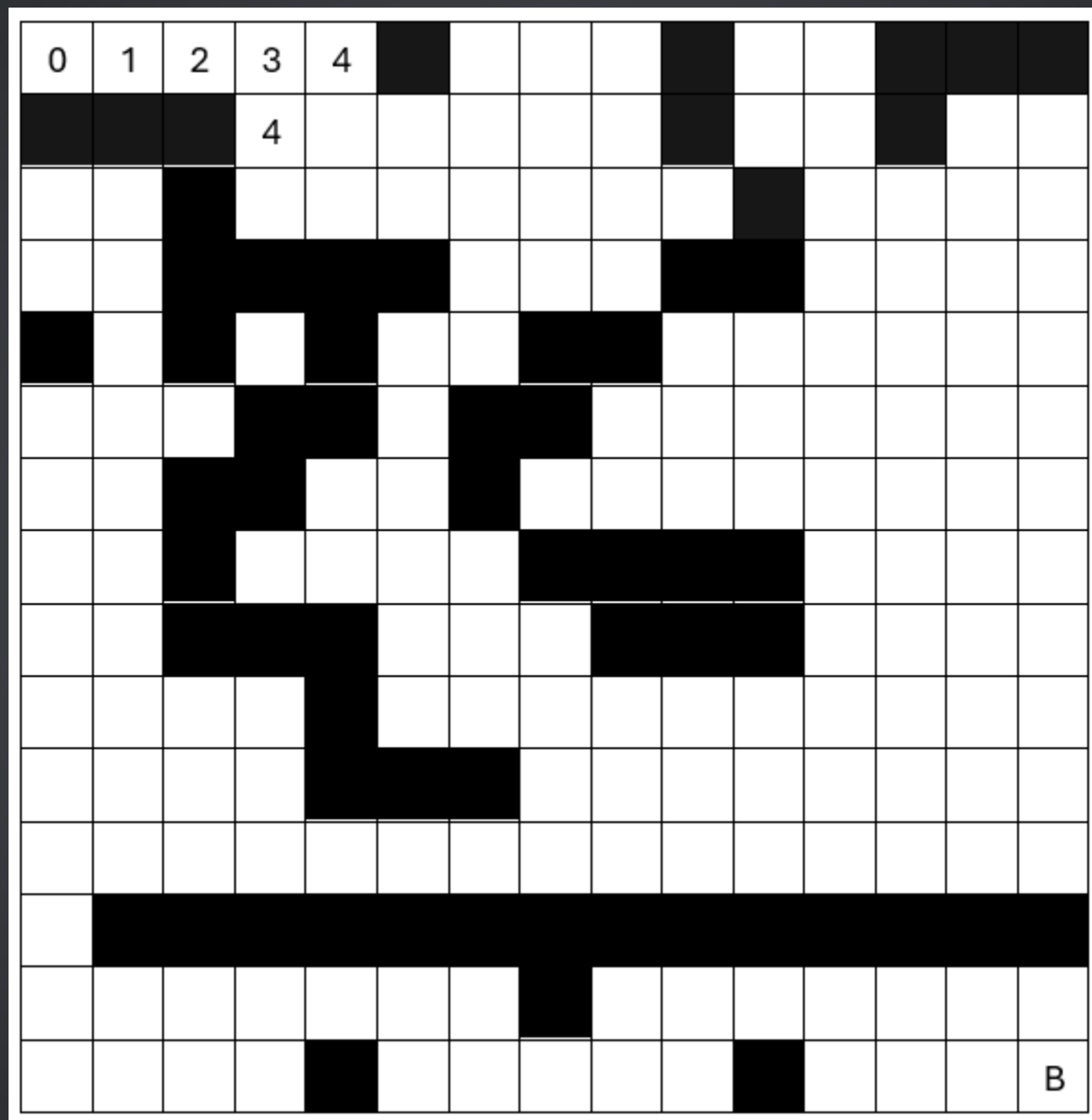
The answer is 89.

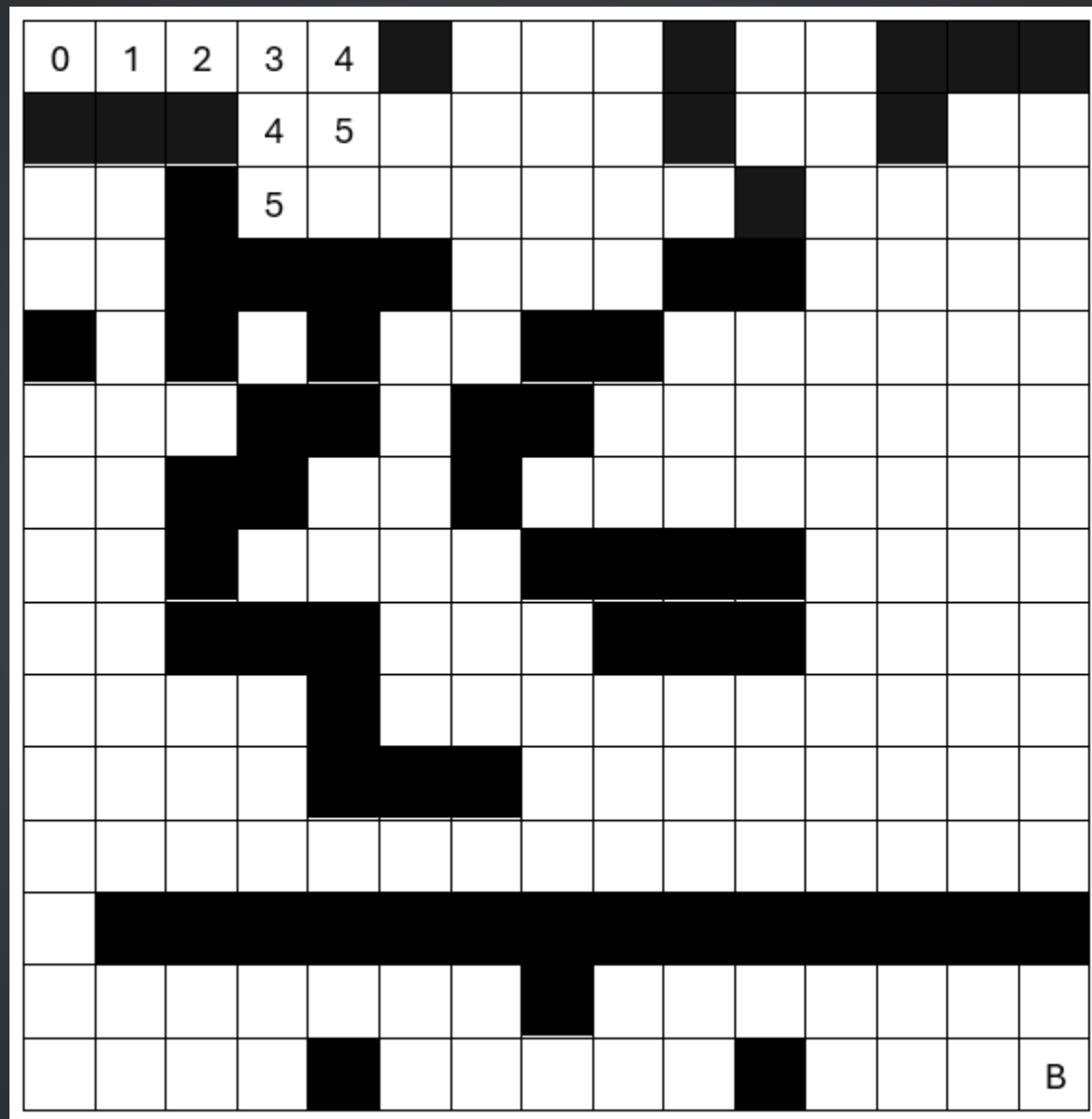


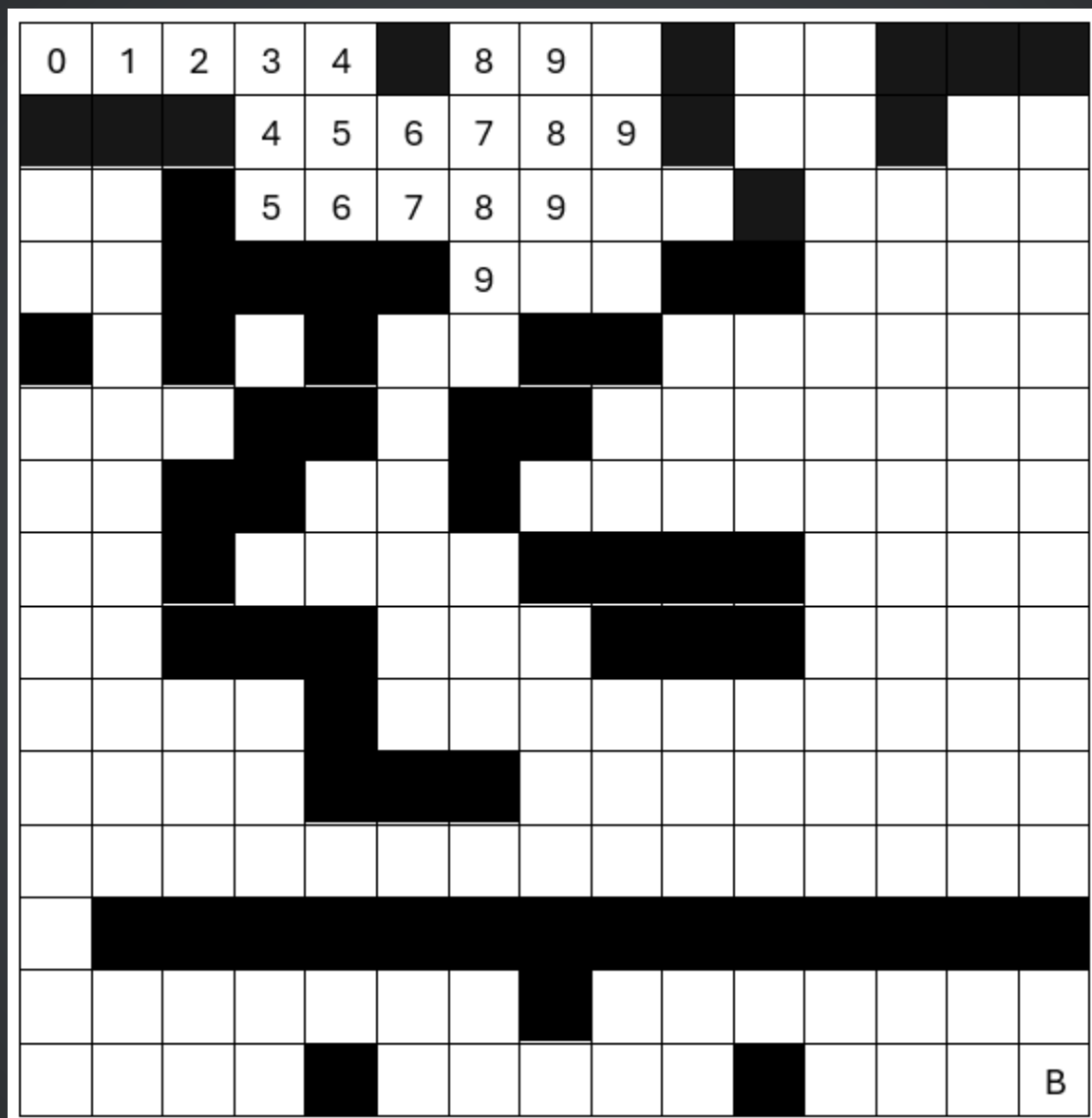












Algorithm:

1. Set the distance of block $(0, 0)$ to 0.
2. In round $n = 0, 1, 2, \dots$, find all blocks whose distance is equal to n . Let (i, j) be such a block. Consider its four neighbors, $(i + 1, j)$, $(i, j + 1)$, $(i - 1, j)$, $(i, j - 1)$. If a neighbor is not a stone and does not have a distance yet, then set its distance to be equal to $n + 1$.
3. Run the algorithm many rounds until the last block $(14, 14)$ has a distance. Output this distance.



This is known as Dijkstra's algorithm.
Its developer, Dijkstra, won the Turing Award in 1972.

Exercise: Write the code yourself

```
1 input = [[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1],
2          [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
3          [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
4          [0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0],
5          [1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
6          [0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
7          [0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
8          [0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
9          [0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
10         [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
11         [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
12         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
13         [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
14         [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
15         [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]]
```

Here, 1 means a stone and 0 means a feasible block

Reference Solution

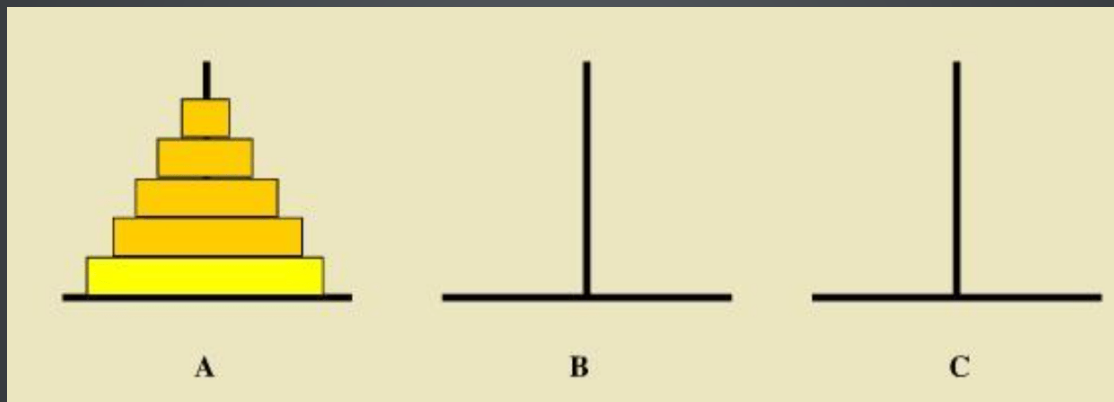
```
1 distance = [[-1] * 15 for _ in range(15)]
2 distance[0][0] = 0
3
4 for round in range(100):
5     for i in range(15):
6         for j in range(15):
7             if distance[i][j] == round:
8                 if ((i>=1) and (distance[i-1][j]==-1) and input[i-1][j]==0):
9                     distance[i-1][j] = round + 1
10                if ((j>=1) and (distance[i][j-1]==-1) and input[i][j-1]==0):
11                    distance[i][j-1] = round + 1
12                if ((i<=13) and (distance[i+1][j]==-1) and input[i+1][j]==0):
13                    distance[i+1][j] = round + 1
14                if ((j<=13) and (distance[i][j+1]==-1) and input[i][j+1]==0):
15                    distance[i][j+1] = round + 1
16
17 print(distance[14][14])
```

The answer is 46.

Exercise

Use AI to generate an interactive APP which demonstrates the algorithm as a game

Tower of Hanoi (汉诺塔)

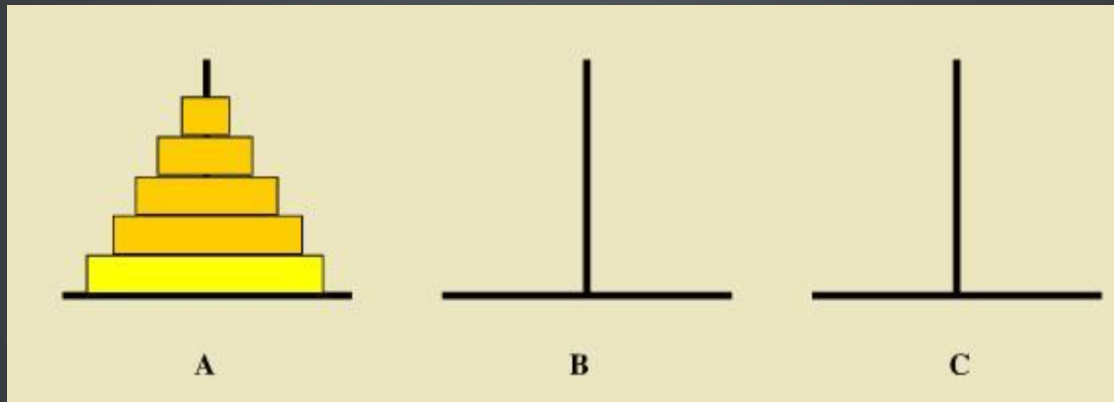


Tower of Hanoi

You want to move n disks from rod A to rod C. Here are the rules:

- Only one disk can be moved at each round.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
- No disk may be placed on top of a disk that is smaller than it.

How to solve the problem?



Let's solve a simpler problem: $n = 1$

- Step 1: Move one disk $A \rightarrow C$.
- Done!

Let's solve a simpler problem: $n = 2$

- Step 1: Move one disk $A \rightarrow B$.
- Step 2: Move one disk $A \rightarrow C$.
- Step 3: Move one disk $B \rightarrow C$.

Imagine that we already know how to move 99 disks from one rod to another rod. Then, how to move 100 disks?

- Step 1: Move the top 99 disks $A \rightarrow B$,
- Step 2: Move the last disk $A \rightarrow C$,
- Step 3: Move the top 99 disks $B \rightarrow C$.

In general, if you are able to move n disks, you should be able to move $n + 1$ disks!

Divide-and-Conquer

Exercise: Write a program which solves the Hanoi problem given any n . The output is a few steps such as $A \rightarrow B$, $B \rightarrow C$, $B \rightarrow A$, etc.

```
1 def tower_of_hanoi(n, source, target, auxiliary):
2     if n == 1:
3         print(f"Move disk 1 from rod {source} to rod
4             {target}")
5         return
6         tower_of_hanoi(n-1, source, auxiliary, target)
7         print(f"Move disk {n} from rod {source} to rod {target}")
8         tower_of_hanoi(n-1, auxiliary, target, source)
9
10 n = 3 # Number of disks
11 source_rod = 'A'
12 target_rod = 'C'
13 auxiliary_rod = 'B'
14
15 print(f"Solving Tower of Hanoi problem for {n} disks:")
16 tower_of_hanoi(n, source_rod, target_rod, auxiliary_rod)
```

Exercise: Can you propose another algorithm which uses the idea of divide-and-conquer?

Searching a Dictionary

A dictionary lists words in alphabetical order. If you are looking for a specific words, do you search page by page to find the words? How can do you it more efficiently?

Searching a Dictionary

A dictionary lists words in alphabetical order. If you are looking for a specific words, do you search page by page to find the words? How can do you it more efficiently?

You start from the middle and check whether the target is (1) on that middle page, (2) before that middle page, or (3) after that middle page.

Searching a Dictionary

function search(word, start_page, end_page):

 If start_page = end_page, return that page.

 Otherwise, let mid_page = (start_page + end_page)/2;

 If the word is on the mid_page, return that page.

 If the word is before the mid_page:

 Return search(word, start_page, mid_page - 1)

 If the word is after the mid_page:

 Return search(word, mid_page + 1, end_page)

Exercise

门萨智商测试题：以左下角的球为起点、右上角的球为终点，选择9个互相连接的球，并将球上的数字相加，能得出的最大值会是多少？



A:457 B:51 C:其他

算法比答案更重要

这是从一道所谓门萨智商测试题修改而来的新题目。我想知道解这种题目的算法到底是什么样的。
除了暴力遍历，还有其他做法吗？

说点什么... 10 3 22

You are moving from the bottom-left ball to the top-right ball. Each round you can move up or move right, so you will pass 9 balls in total. Summing up these 9 numbers, what is the maximum value you can get?

Write an algorithm for it!