

Welcome to Colab!

(New) Try the Gemini API

- [Generate a Gemini API key](#)
- [Talk to Gemini with the Speech-to-Text API](#)
- [Compare Gemini with ChatGPT](#)
- [More notebooks](#)

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.



What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to find out more, or just get started below!

✓ Getting started

The document that you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day

86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut 'Command/Ctrl+Enter'. To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week

604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers

or friends, allowing them to comment on your notebooks or even edit them. To find out more, see [Overview of Colab](#). To create a new Colab notebook you can use the File menu above, or use the following link: [Create a new Colab notebook](#).

Colab notebooks are Jupyter notebooks that are hosted by Colab. To find out more about the Jupyter project, see jupyter.org.

✓ Data science

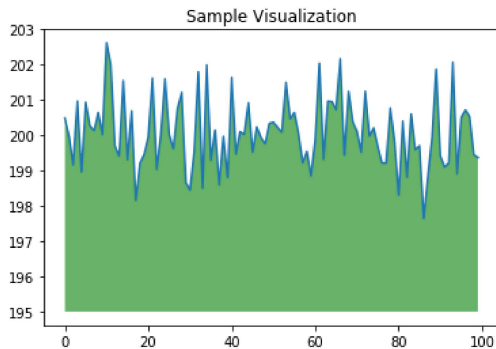
With Colab you can harness the full power of popular Python libraries to analyse and visualise data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualise it. To edit the code, just click the cell and start editing.

```
import numpy as np
from matplotlib import pyplot as plt

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Sample Visualization")
plt.show()
```



You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from GitHub and many other sources. To find out more about importing data, and how Colab can be used for data science, see the links below under [Working with data](#).

✓ Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just [a few lines of code](#). Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including [GPUs and TPUs](#), regardless of the power of your machine. All you need is a browser.

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the [machine learning examples](#) below.

✓ More resources

Working with notebooks in Colab

- [Overview of Colaboratory](#)
- [Guide to markdown](#)
- [Importing libraries and installing dependencies](#)

- [Saving and loading notebooks in GitHub](#)
- [Interactive forms](#)
- [Interactive widgets](#)

Working with data

- [Loading data: Drive, Sheets and Google Cloud Storage](#)
- [Charts: visualising data](#)
- [Getting started with BigQuery](#)

Machine learning crash course

These are a few of the notebooks from Google's online machine learning course. See the [full course website](#) for more.

- [Intro to Pandas DataFrame](#)
- [Linear regression with tf.keras using synthetic data](#)

Using accelerated hardware

- [TensorFlow with GPUs](#)
- [TensorFlow with TPUs](#)

```
from queue import Queue

graph = {0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}
print("The adjacency List representing the graph is:")
print(graph)

def bfs(graph, source):
    Q = Queue()
    visited_vertices = set()
    Q.put(source)
    visited_vertices.update({0})
    while not Q.empty():
        vertex = Q.get()
        print(vertex, end="-->")
        for u in graph[vertex]:
            if u not in visited_vertices:
                Q.put(u)
                visited_vertices.update({u})

print("BFS traversal of graph with source 0 is:")
bfs(graph, 0)

The adjacency List representing the graph is:
{0: [1, 3], 1: [0, 2, 3], 2: [4, 1, 5], 3: [4, 0, 1], 4: [2, 3, 5], 5: [4, 2], 6: []}
BFS traversal of graph with source 0 is:
0-->1-->3-->2-->4-->5-->

graph1 = {
    'A' : ['B','S'],
    'B' : ['A'],
    'C' : ['D','E','F','S'],
    'D' : ['C'],
    'E' : ['C','H'],
    'F' : ['C','G'],
    'G' : ['F','S'],
    'H' : ['E','G'],
    'S' : ['A','C','G']
}

def dfs(graph, node, visited):
    if node not in visited:
        visited.append(node)
        for k in graph[node]:
            dfs(graph,k, visited)
    return visited

visited = dfs(graph1,'D', [])
print(visited)
```

```
['D', 'C', 'E', 'H', 'G', 'F', 'S', 'A', 'B']
```

```

from copy import deepcopy
import numpy as np
import time

def bestsolution(state):
    bestsol = np.array([], int).reshape(-1, 9)
    count = len(state) - 1
    while count != -1:
        bestsol = np.insert(bestsol, 0, state[count]['puzzle'], 0)
        count = (state[count]['parent'])
    return bestsol.reshape(-1, 3, 3)

# checks for the uniqueness of the iteration(it).
def all(checkarray):
    set=[]
    for it in set:
        for checkarray in it:
            return 1
    else:
        return 0

# number of misplaced tiles
def misplaced_tiles(puzzle,goal):
    mscost = np.sum(puzzle != goal) - 1
    return mscost if mscost > 0 else 0

def coordinates(puzzle):
    pos = np.array(range(9))
    for p, q in enumerate(puzzle):
        pos[q] = p
    return pos

# start of 8 puzzle evaluation, using Misplaced tiles heuristics
def evaluate_misplaced(puzzle, goal):
    steps = np.array([('up', [0, 1, 2], -3),('down', [6, 7, 8], 3),('left', [0, 3, 6], -1),('right', [2, 5, 8], 1)],
        dtype = [('move', str, 1),('position', list),('head', int)])

    dtstate = [('puzzle', list),('parent', int),('gn', int),('hn', int)]

    costg = coordinates(goal)

    # initializing the parent, gn and hn, where hn is misplaced_tiles function call
    parent = -1
    gn = 0
    hn = misplaced_tiles(coordinates(puzzle), costg)
    state = np.array([(puzzle, parent, gn, hn)], dtstate)

    #priority queues with position as keys and fn as value.
    dtpriority = [('position', int),('fn', int)]

    priority = np.array([(0, hn)], dtpriority)

    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        # sort priority queue using merge sort,the first element is picked for exploring.
        priority = np.delete(priority, 0, 0)
        puzzle, parent, gn, hn = state[position]
        puzzle = np.array(puzzle)

        blank = int(np.where(puzzle == 0)[0])

        gn = gn + 1
        c = 1
        start_time = time.time()
        for s in steps:
            c = c + 1
            if blank not in s['position']:
                openstates = deepcopy(puzzle)
                openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank]

                if ~(np.all(list(state['puzzle']) == openstates, 1)).any():

```

```

end_time = time.time()
if (( end_time - start_time ) > 2):
    print(" The 8 puzzle is unsolvable \n")
    break

hn = misplaced_tiles(coordinates(openstates), costg)
# generate and add new state in the list
q = np.array([(openstates, position, gn, hn)], dtype=state)
state = np.append(state, q, 0)
# f(n) is the sum of cost to reach node
fn = gn + hn

q = np.array([(len(state) - 1, fn)], dtype=priority)
priority = np.append(priority, q, 0)

if np.array_equal(openstates, goal):
    print(' The 8 puzzle is solvable \n')
    return state, len(priority)

return state, len(priority)

# initial state
puzzle = []

puzzle.append(2)
puzzle.append(8)
puzzle.append(3)
puzzle.append(7)
puzzle.append(1)
puzzle.append(4)
puzzle.append(0)

```