Python Fundamentals and Google Colab

Summer Institute in Computational Social Science

(SICSS-Lingnan, June 16 to June 27, 2025 | Hong Kong SAR)

Teaching Assistant: Juliana Yuncg (Qi Xuan Yuncg)

2025-06-13 (Pre-Camp Training)

- Welcome and Overview
- Getting Started with Google Colab
- Introduction to Python Basics
- Loops and Logic
- Working with DataFrames and CSVs
- Library Imports and Writing Functions
- Debugging and Error Handling
- Wrap-Up and Learning Beyond the Classroom

SICSS-LU

01

Welcome and Overview

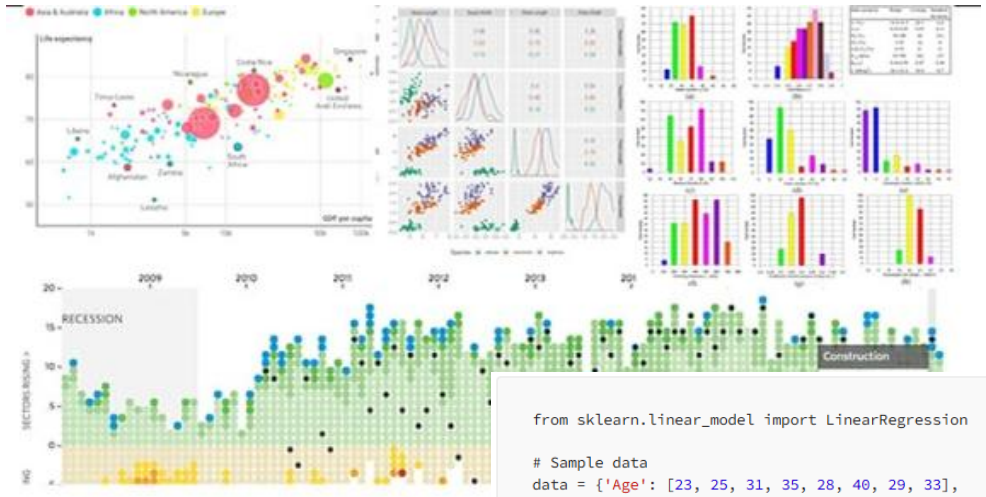
Welcome and Overview

What You'll Learn Today

- How to use **Google Colab**
  - Run code, upload files, mount Google Drive, runtime behavior
- Understand **basic Python syntax**
  - Print statements, indentation, comments, common errors
- Work with **variables, data types, and logic**
  - Strings, numbers, loops, conditionals
- Use **external data with pandas**
  - Import CSVs, manipulate DataFrames, export results
- Write your own **functions** and import libraries
  - Learn def, import pandas, import numpy
- Extend learning with **AI tools like ChatGPT**
  - Prompt templates and self-study strategies

Why Learn Python?

- Easy-to-read syntax
- Versatile: Data analysis, automation, AI-Machine Learning
- Large supportive community and open-source libraries



(Okutan, 2024)

```
from sklearn.linear_model import LinearRegression

# Sample data
data = {'Age': [23, 25, 31, 35, 28, 40, 29, 33],
        'Salary': [48000, 52000, 58000, 60000, 53000, 68000, 59000, 62000]}
df = pd.DataFrame(data)

# Linear regression
X = df[['Age']]
y = df['Salary']
model = LinearRegression().fit(X, y)

predicted_salary = model.predict([[30]])
print(f"Predicted Salary for Age 30: {predicted_salary[0]}")
```

GitHub
https://github.com › PyGithub › PyGithub · 翻譯這個網頁

PyGithub/PyGithub: Typed interactions with the GitHub API v3

PyGitHub is a Python library to access the GitHub REST API. This library enables you to manage GitHub resources such as repositories, user profiles, and ...

(Maheta, 2025)

Here is an example of a data compilation script using Python:

```
import pandas as pd

# Read CSV files and concatenate them
df1 = pd.read_csv('file1.csv')
df2 = pd.read_csv('file2.csv')
df_combined = pd.concat([df1, df2])

# Group data and calculate statistics
grouped = df_combined.groupby('category')
stats = grouped.mean()

# Save results to a new CSV file
stats.to_csv('results.csv')
```

02

Getting Started with Google Colab

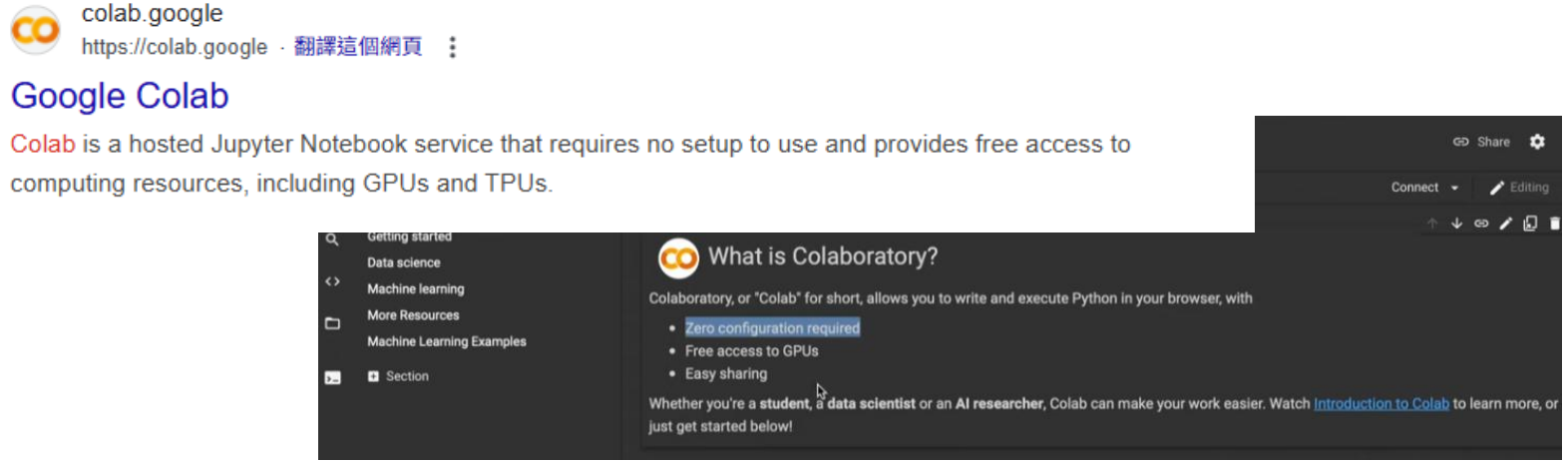
Getting Started with Google Colab

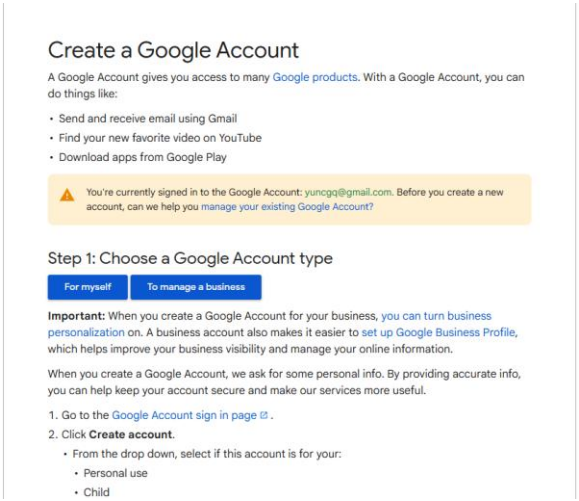
What is Google Colab?

- Cloud-based environment for writing and executing Python code
- Based on Jupyter Notebook but pre-installed and free to use
- Requires a Google account, no local setup needed

Key Benefits

- Zero installation required
- Collaborate and share notebooks easily
- Access to free GPUs for computation
- Seamlessly integrates with Google Drive





Google Colab Tutorial for Beginners | Get Started with Google Colab

Reference Notes

https://www.youtube.com/watch?v=RLYoEyIHL6A (Ozgon, 2021)

Reference Notes

Create a Google Account

https://support.google.com/accounts/answer/27441?hl=en&co=GENIE.Platform%3DAndroid (Google, 2025)

Getting Started with Google Colab

Opening a Notebook

Option 1: Create from Colab interface: colab.research.google.com

Option 2: Open .ipynb file from your Google Drive

Option 3: Upload from your computer

Reference Notes:

How to Open Google Colab Notebook (.ipynb) from Google Drive | (colab.research.google.com)

https://www.youtube.com/watch?v=ly3XB2e99R4

(Thinks, 2022)

Getting Started with Google Colab

Navigating the Colab Interface

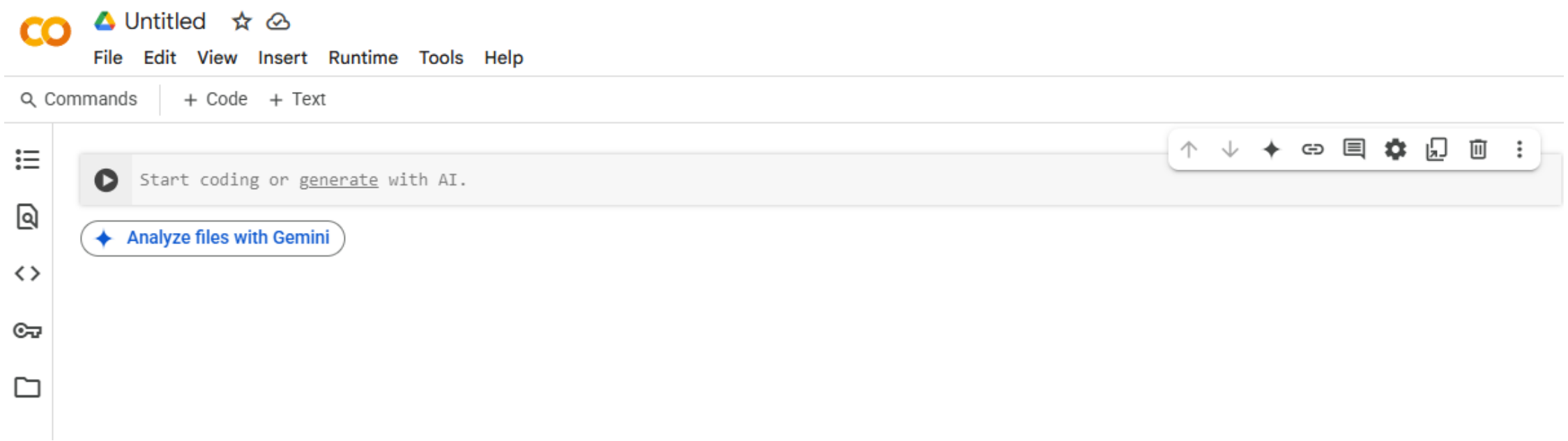
- Toolbar: File, Edit, View, Insert, Runtime, Tools
- "+ Code" to add Python code
- "+ Text" to add markdown (headers, notes)
- Run button ▶ or Shift + Enter to execute cells

Colab Settings

- Change themes (dark/light)
- Enable Table of Contents under "View"
- Manage sessions, clear runtime memory

Managing Runtimes

- Colab runs on a **temporary virtual machine**
  - If runtime disconnects or times out, uploaded data is lost
- **Tip:** Mount Google Drive to save and reload files across sessions



Getting Started with Google Colab

File Access in Colab

❌ Common Mistake : File not uploaded or linked

```
[3] pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```
[5] import pandas as pd
```

```
[18] # This will fail in Colab if the file isn't uploaded or linked
     df = pd.read_csv("D:\\my_data.csv")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-18-abe1beb854fd> in <cell line: 0>()
      1 # This will fail in Colab if the file isn't uploaded or linked
----> 2 df = pd.read_csv("D:\\my_data.csv")

4 frames
/usr/local/lib/python3.11/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    871         if ioargs.encoding and "b" not in ioargs.mode:
    872             # Encoding
--> 873             handle = open(
    874                 handle,
    875                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'D:\\my_data.csv'
```

Getting Started with Google Colab

Correct Way: Mount Google Drive

GPT prompt: I saved the my_data.csv at the following path of google colab:https://drive.google.com/drive/my-drive, what is the python code to read the file at google colab

```
[12] from google.colab import drive
     drive.mount('/content/drive')

     import pandas as pd

     # Example path (replace with your file's path)
     file_path = '/content/drive/MyDrive/my_data.csv'

     # Read the CSV file
     df = pd.read_csv(file_path)

     # Verify the data (optional)
     print(df.head())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
      name  age  score  passed
0    Alice   24     85    True
1      Bob   27     90    True
2  Charlie   22     78   False
3    Diana   32     92    True
4    Ethan   29     88    True
```

Introduction to Python Basic Syntax

Why Python — Especially for R or SPSS Users?

Feature	Python	R
General-purpose	✅ Yes	❌ Mostly statistical
Web scraping	✅ Strong (requests, BeautifulSoup)	✅ But less native
Machine learning	✅ Dominant (scikit-learn, TensorFlow)	Possible, but fewer mainstream libraries
Social media APIs / automation	✅ Broad use	❌ Limited
Readability & integration	✅ Beginner-friendly, integratable into apps	❌ More math/stat style

R is great for statistics. Python is great for stitching data analysis into research workflows, scraping data, building tools, and doing automation (e.g. office automation).

Why Python vs. C / Java?

Feature	Python	C / Java
Syntax simplicity	✅ Very simple	❌ Verbose
Runtime safety	✅ Dynamic, flexible	✅ But strict & compiled
Learning curve	✅ Gentle	❌ Steeper
Use in data science	✅ First choice	❌ Not typical

Python trades speed for simplicity — that's why it's used in teaching, prototyping, and research

Introduction to Python Basic Syntax

Python is Simple and Readable

- Does not need curly braces or semicolons
  - Indentation = Structure (no {} like in C or Java)
- Case-sensitivity
- Dynamically Typed

print() Function

- Displays output in the console
- Syntax: print("Hello, world!")
- Can print strings, numbers, or variables

Text (String) vs Numbers

- Strings: "Hello" or 'World'
- Numbers: 123, 4.56
- Use + to concatenate strings: "Hi " + "there!"

Comments in Python

- Use # for single-line comments
- Use triple quotes ''' or """ for multiline documentation

Common Mistakes

- Missing quotation marks
- Forgetting to close parentheses
- Mixing strings and integers without type conversion

Variables, Data Types, and Transformations

What Is a Variable?

- A variable stores a value for later use
- Syntax: variable_name = value
- No need to declare type in advance

Common Data Types

- int: Integer (e.g. 900)
- float: Decimal (e.g. 4.5)
- str: String/Text (e.g. "Apple")
- bool: Boolean (True / False)

Naming Rules

- Case-sensitive
- Use _ to separate words (e.g. phone_price)
- Can't start with a number

Type Checking & Type Conversion

- Check type: type(variable)
- Convert:
  - str(900) → "900"
  - int("900") → 900
  - float(5) → 5.0

Why Conversion Matters?

- Strings and numbers can't be combined directly
- Prevents common errors in printing and calculations

Loops and Logic

Building Logic with Conditionals and Loops

Why Use if Statements?

Concept Overview

Conditionals allow your program to **make decisions** based on different inputs, variables, or situations.

Just like in daily life:

"If I have an umbrella, I'll go out."

"If the product is affordable, I'll buy it."

Python mimics this logic using if, else, and elif.

Real-World Analogy

Situation	Logic in Your Head	Python Version
If it's raining, bring umbrella	if raining: bring umbrella	if is_raining: take_umbrella()
If the price is low, buy it	if price < budget: buy item	if price < budget: buy()
If age > 18, grant access	if age > 18: allow entry	if age > 18: grant_access()

if Statements-Code Examples

if Statements

Code Example 1: Budget Check

```
[ ]  price = 800
     budget = 1000

     if price <= budget:
            print("You can buy this item.")
     else:
            print("Sorry, it's too expensive.")
```

```
You can buy this item.
```

if Statements-Code Examples

Code Example 2: Age Gate

```
[ ]  age = int(input("Enter your age: "))

     if age < 18:
            print("You're a children.")
     else:
            print("You're an adult.")
```

```
Enter your age: 8
You're a children.
```

Note: We use int() to convert input into a number.

Common Mistake

```
# This will break!
age = input("Enter your age: ")

if age < 18:
       print("You're a student.")    # ❌ TypeError: '<' not supported between instances of 'str' and 'int'
```

```
Enter your age: 89
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-c0138bf0be04> in <cell line: 0>()
      2 age = input("Enter your age: ")
      3
----> 4 if age < 18:
      5     print("You're a student.")  # ❌ TypeError: '<' not supported between instances of 'str' and 'int'

TypeError: '<' not supported between instances of 'str' and 'int'
```

Fix:

```
[ ]  age = int(input("Enter your age: "))
```

```
Enter your age: 89
```

Using if / elif / else to Handle Multiple Conditions

Concept Overview

Sometimes a simple **yes/no (if/else)** isn't enough.

You need to **check multiple conditions** one after another.

That's where elif ("else if") comes in.

Real-World Analogy

Think of classifying people based on age:

Age Range	Group
Under 18	Children
18–64	Working adult
65+	Senior citizen

Instead of writing many if statements, use one if, multiple elifs, and one final else.

if / elif / else – Code Examples

Code Example 1: Age Classification

```
[ ]  age = int(input("Enter your age: "))

     if age < 18:
         print("Student")
     elif age < 65:
         print("Working adult")
     else:
         print("Senior")
```

```
Enter your age: 65
Senior
```

Explanation:

if age < 18: → only executes if the condition is True

elif age < 65: → only checks if the first condition was False

else: → catches all other remaining cases

Example Outputs:

Input: 15 → Output: Student

Input: 30 → Output: Working adult

Input: 70 → Output: Senior

if / elif / else – Code Examples

Code Example 2: Grade System

```
[ ]  score = int(input("Enter your exam score: "))

     if score >= 90:
         print("Grade: A")
     elif score >= 80:
         print("Grade: B")
     elif score >= 70:
         print("Grade: C")
     elif score >= 60:
         print("Grade: D")
     else:
         print("Grade: F")
```

```
Enter your exam score: 5
Grade: F
```

Common Mistake

```
[ ]  # This won't work logically
     if age < 18:
         print("Student")
     if age < 65:
         print("Working adult")    # This will also run if age is under 18
```

Tips

Statement	Meaning
`if`	First condition (checked no matter what)
`elif`	Additional condition (only checked if all above are False)
`else`	Final fallback (runs if all previous are False)

Always use elif if your cases are mutually exclusive (non-overlapping).

Tips: ChatGPT Prompt

"Write a Python program that prints the weather category: Cold (<10), Warm (10–25), Hot (>25). Use if/elif/else."

for Loops — Best for Counting

Concept Overview

A for loop in Python allows you to **repeat an action a specific number of times**, or **iterate over items** in a sequence (like a list or string).

The most common use case for beginners is **counting** — and that's done using the built-in range() function.

Syntax Breakdown

python 复制 编辑

```python
for variable in range(stop):
    # do something
```

- `variable` : changes each time the loop runs
- `range(stop)` : gives a sequence of numbers from 0 up to (but not including) `stop`

Real-World Analogy:: "Handing Out Exam Papers"

Scenario:

You're a teacher handing out 5 exam papers, one to each student in a row.

Each time you move to the next student, you:

1. Count the student (1, 2, 3, ...)
2. Hand them a paper
3. Repeat until all 5 students have received one

for Loops — Best for Counting

Real-World Analogy:: “Handing Out Exam Papers”

Scenario:

You’re a teacher handing out 5 exam papers, one to each student in a row.

Each time you move to the next student, you:

1. Count the student (1, 2, 3, …)
2. Hand them a paper
3. Repeat until all 5 students have received one

How This Relates to a for Loop in Python:

```
[1]  for i in range(5):
         print("Handed paper to student", i + 1)
```

```
Handed paper to student 1
Handed paper to student 2
Handed paper to student 3
Handed paper to student 4
Handed paper to student 5
```

range(5) is the number of students

i is the student’s index (starts at 0, so we use i + 1 for natural counting)

The loop automates the repeated action of handing out a paper

Message: "A for loop is like doing the same task again and again — like handing out papers, putting stickers on folders, or checking names off a list. Instead of repeating code 5 times, you let the loop do it for you."

for Loops — Best for Counting

Code Example 1: Count from 0 to 4

```
[1] for i in range(5):
        print("Count:", i)
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

range(5) creates the list [0, 1, 2, 3, 4]

i takes on each value in the list, one at a time

Code Example 2: Print Even Numbers Between 1 and 10

```
[2] for number in range(2, 11, 2):
        print(number)
```

```
2
4
6
8
10
```

range(start, stop, step):

Start at 2, stop before 11, step by 2

Code Example 3: Loop Through a List

```
[3] fruits = ["apple", "banana", "cherry"]

    for fruit in fruits:
        print("I like", fruit)
```

```
I like apple
I like banana
I like cherry
```

while Loops — Best for Conditions

Concept Overview:

A **while loop** repeats a block of code **as long as a specified condition is True**.

Unlike a for loop (which knows ahead of time how many times to repeat), a while loop **doesn't know the exact number of repetitions**.

It keeps going until the condition becomes False.

Key Idea:

Use while when you're **waiting for something to change**, or **checking a condition dynamically**.

Real-World Analogy: Everyday while Examples

Situation	Logic
Keep walking while it's not raining	while not raining:
Keep printing until the printer runs out of paper	while paper > 0:
Ask password until it's correct	while password != correct:

while Loops — Best for Conditions

Code Example: Basic Counter

```
[4]  i = 0
     while i < 5:
         print("Count:", i)
         i += 1
```

```
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

Explanation: i = 0: starting value

while i < 5: loop continues as long as i is less than 5

i += 1: increments i so we eventually break the loop

Common Mistake: Infinite Loop

```
[ ]  i = 0
     while i < 5:
         print("Count:", i)
         # Forgot i += 1
```

This runs forever because i never changes → i < 5 stays True.

Always ensure the condition will eventually become False.

GPT Exercise Prompt:

Write a while loop that asks for a password until the correct one is entered:

```
[5]  password = ""
     while password != "secret123":
         password = input("Enter password: ")

     print("Access granted.")
```

```
Enter password: 123
Enter password: 123
Enter password: secret123
Access granted.
```

range() Function Basics

Concept Overview: What is range()?

The **range()** function in Python is used to generate a sequence of numbers. It is most commonly used in **for loops** to control how many times the loop runs.

Syntax:

```python
range(start, stop, step)
```

Parameter	Description
`start`	(Optional) The number to start from (default = 0)
`stop`	The number to stop before (not inclusive)
`step`	(Optional) The amount to increment by (default = 1)

`range()` returns a virtual list of numbers—it doesn't create the full list in memory, which makes it efficient even for large ranges.

Real-World Analogy: Numbered Tickets in a Queue

Imagine you're at a service center. A ticket dispenser hands out queue numbers in sequence.

- If it uses `range(5)`, it prints: 0, 1, 2, 3, 4
- If it uses `range(2, 6)`, it starts from 2 and stops before 6 → 2, 3, 4, 5
- If it uses `range(1, 10, 2)`, it gives you every second number → 1, 3, 5, 7, 9

This mirrors how you use `range()` in loops to control exactly how numbers are counted or skipped.

Another analogy: Think of a bus that stops at every other block. If the blocks are numbered 1 to 9, and the bus stops every 2 blocks, the stops are 1, 3, 5, 7, 9 — just like `range(1, 10, 2)`.

range() Function Basics

Example 1: range(stop)

Prints numbers from 0 to stop - 1

```
[6] for i in range(5):
        print(i)
```

```
0
1
2
3
4
```

Example 2: range(start, stop)

Starts from start and stops before stop

```
[7] for i in range(2, 6):
        print(i)
```

```
2
3
4
5
```

Example 3: range(start, stop, step)

Includes a step size to skip values

```
[8] for i in range(1, 10, 2):
        print(i)
```

```
1
3
5
7
9
```

To view the list generated by range(), you can convert it using list():

```
[9] print(list(range(1, 10, 2)))
    # Output: [1, 3, 5, 7, 9]
```

```
[1, 3, 5, 7, 9]
```

05

Working with DataFrames and CSVs

Handling Real-World Data with pandas and CSVs

Why pandas?

pandas is a powerful library for:

Working with tables (like Excel)	Cleaning, filtering, and analyzing structured data

- Think of a **DataFrame** as a spreadsheet you can program

python 複製 编辑

```python
import pandas as pd
```

Loading Data into a DataFrame

Option 1: Upload CSV file manually in Colab

```
[11] from google.colab import files
     uploaded = files.upload()

     df = pd.read_csv("my_data.csv")
```

```
Choose Files my_data.csv
• my_data.csv(text/csv) - 108 bytes, last modified: 6/5/2025 - 100% done
Saving my_data.csv to my_data.csv
```

Option 2: Load from Google Drive

```
[13] from google.colab import drive
     drive.mount('/content/drive')

     df = pd.read_csv('/content/drive/MyDrive/my_data.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Notes:You must upload or mount — local paths like C:\Users... don't work in Colab!

Exploring DataFrames

```
df.head()          # Show top 5 rows
```

	name	age	score	passed
0	Alice	24	85	True
1	Bob	27	90	True
2	Charlie	22	78	False
3	Diana	32	92	True
4	Ethan	29	88	True

.head() shows the top rows to preview your data

```
[20] df.describe()  # Summary statistics
```

	age	score
count	5.000000	5.000000
mean	26.800000	86.600000
std	3.962323	5.458938
min	22.000000	78.000000
25%	24.000000	85.000000
50%	27.000000	88.000000
75%	29.000000	90.000000
max	32.000000	92.000000

describe() gives stats like mean, min, max

```
[19] df.info()          # Structure and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    5 non-null      object
 1   age     5 non-null      int64
 2   score   5 non-null      int64
 3   passed  5 non-null      bool
dtypes: bool(1), int64(2), object(1)
memory usage: 257.0+ bytes
```

.info() shows column names, null values, data types

Editing and Manipulating Data

python 复制 编辑

```python
df["new_column"] = df["old_column"] * 2      # Add a new column
df.drop("column_name", axis=1, inplace=True) # Remove a column
df["category"] = df["score"] > 70            # Add Boolean column
```

pandas allows flexible column creation, filtering, and cleanup.

Step 0: Upload and Load CSV

```
[22] from google.colab import files
     uploaded = files.upload()

     import pandas as pd
     df = pd.read_csv("my_data.csv")
     print("Original DataFrame:")
     df.head()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving my_data.csv to my_data (3).csv
Original DataFrame:
```

	name	age	score	passed
0	Alice	24	85	True
1	Bob	27	90	True
2	Charlie	22	78	False
3	Diana	32	92	True
4	Ethan	29	88	True

Explanation:

files.upload() opens a file picker in Colab — students can upload their my_data.csv file.

We read it into a pandas DataFrame named df.

df.head() shows the first 5 rows so we can inspect the original data.

Editing and Manipulating Data

Step 1: Add a New Column score_doubled

```
[23] df["score_doubled"] = df["score"] * 2
     print("After adding score_doubled:")
     df.head()
```

```
After adding score_doubled:
```

	name	age	score	passed	score_doubled
0	Alice	24	85	True	170
1	Bob	27	90	True	180
2	Charlie	22	78	False	156
3	Diana	32	92	True	184
4	Ethan	29	88	True	176

Explanation:

Creates a column score_doubled by doubling each value in score.

Helps illustrate how pandas operates element-wise on columns and creates new data.

Editing and Manipulating Data

Step 2: Drop the Column passed

```
[24]  df.drop("passed",  axis=1,  inplace=True)
      print("After  dropping  'passed'  column:")
      df.head()
```

```
After dropping 'passed' column:
```

	name	age	score	score_doubled
0	Alice	24	85	170
1	Bob	27	90	180
2	Charlie	22	78	156
3	Diana	32	92	184
4	Ethan	29	88	176

Explanation: Removes the passed column from the DataFrame.

axis=1 specifies column removal, and inplace=True updates df directly.

Useful for cleaning out unnecessary or sensitive data.

Code:

python 複製 編輯

```python
df.drop("passed", axis=1, inplace=True)
```

What does each part mean?

- `df` : This is your DataFrame containing the data.
- `.drop()` : This is a pandas method used to remove rows or columns from a DataFrame.
- `"passed"` : This is the name of the column you want to delete.
- `axis=1` : This tells pandas you are dropping a column. If you used `axis=0`, it would mean dropping a row.
- `inplace=True` : This means the change is applied directly to the original DataFrame (`df`). It does not return a new modified DataFrame, it simply updates the existing one.

Editing and Manipulating Data

Step 2: Drop the Column passed

Code:

```python
df.drop("passed", axis=1, inplace=True)
```

What does each part mean?

- `df`: This is your DataFrame containing the data.
- `.drop()`: This is a pandas method used to remove rows or columns from a DataFrame.
- `"passed"`: This is the name of the column you want to delete.
- `axis=1`: This tells pandas you are dropping a column. If you used `axis=0`, it would mean dropping a row.
- `inplace=True`: This means the change is applied directly to the original DataFrame (`df`). It does not return a new modified DataFrame, it simply updates the existing one.

Example comparison:

Without `inplace=True`:

```python
new_df = df.drop("passed", axis=1)
```

This would create a new DataFrame (`new_df`) that excludes the "passed" column, but the original `df` remains unchanged.

With `inplace=True`:

```python
df.drop("passed", axis=1, inplace=True)
```

This updates the original `df` immediately, removing the "passed" column without needing to assign it to a new variable.

Editing and Manipulating Data

Step 3: Add a Boolean Column high_score

```
[27] df["high_score"] = df["score"] > 70
     print("After creating 'high_score' boolean column:")
     df.head()
```

```
After creating 'high_score' boolean column:
```

	name	age	score	score_doubled	high_score
0	Alice	24	85	170	True
1	Bob	27	90	180	True
2	Charlie	22	78	156	True
3	Diana	32	92	184	True
4	Ethan	29	88	176	True

Explanation:

Creates a column high_score with True if score > 70, otherwise False.

Demonstrates how to generate categorical flags based on numeric conditions.

Editing and Manipulating Data

Step 4: Review the Final DataFrame

```
[26] print("Final  DataFrame:")
     df
```

```
Final DataFrame:
```

	name	age	score	score_doubled	high_score
0	Alice	24	85	170	True
1	Bob	27	90	180	True
2	Charlie	22	78	156	True
3	Diana	32	92	184	True
4	Ethan	29	88	176	True

Explanation:

Presents the updated DataFrame with all transformations applied.

Should review column names and values to ensure everything looks correct.

Saving Modified Data

Saving Modified Data

```
[28] # Save the modified DataFrame to a new CSV file
     df.to_csv("updated_my_data.csv", index=False)

     # Trigger download in Google Colab
     from google.colab import files
     files.download("updated_my_data.csv")
```

Explanation:

df.to_csv("updated_my_data.csv", index=False)

Saves the DataFrame into a new CSV file named updated_my_data.csv.

The index=False argument excludes the row index from the CSV file.

files.download(...) Launches a browser-based download prompt so you can save the work to a local machine.

06

Library Imports and Writing Functions

Using Libraries (libraries import)

- Instead of reinventing the wheel, you just:

```python
import pandas as pd
import numpy as np
```

Common Libraries

Library	Use Case
pandas	Data wrangling, tables (DataFrames)
numpy	Math operations, arrays
matplotlib	Visualization
os	File paths and folder management

Always use short aliases like pd, np for ease of use.

Using numpy

Concept Overview: Why use NumPy?

NumPy (Numerical Python) is a foundational Python library for performing fast, efficient numerical operations on lists, arrays, and matrices.

It is optimized for scientific computing and is widely used in data science, machine learning, and statistics.

np.mean() is one of many functions NumPy provides for statistical analysis. It calculates the **average** (arithmetic mean) of a list or array of numbers.

Code Example: Using numpy

Code Example: Using numpy

```
[29] import numpy as np

     data = [1, 2, 3, 4, 5]
     mean_val = np.mean(data)
     print("Average is:", mean_val)
```

```
Average is: 3.0
```

np.mean() computes the average of the list

- `import numpy as np`:
  This imports the NumPy library and gives it a short alias `np`, which is the standard convention.
- `data = [1, 2, 3, 4, 5]`:
  This creates a basic Python list containing five integers.
- `np.mean(data)`:
  This calculates the average (mean) of the numbers in the list.
  Formula used internally:
  $(1 + 2 + 3 + 4 + 5)/5 = 15/5 = 3.0$
- `print("Average is:", mean_val)`:
  This prints the result in a readable format.

While you could manually compute the mean using:

python 複製 編輯

```
sum(data) / len(data)
```

NumPy is:

- More efficient for large datasets
- Compatible with NumPy arrays and matrices
- Packed with many more functions (standard deviation, median, etc.)

Writing Functions

Concept Overview: Why Write Functions?

Functions in Python are **reusable blocks of code** designed to perform a specific task. They help structure your programs better and avoid repetition.

Benefits:

Avoid repetition: You write the logic once and reuse it many times.	**Improve readability**: Code becomes easier to understand and maintain.	**Enable testing and reuse**: Functions can be tested in isolation and reused across files or projects.

Functions follow the principle of **"define once, use many times."**

Writing Functions

A. Basic Function Syntax

python 複製 編輯

```python
def greet(name):
    print("Hello, " + name)
```

Explanation:

- `def` : This keyword tells Python you're defining a function.
- `greet` : This is the function name.
- `name` : This is a **parameter** — a placeholder for any value you pass in when calling the function.
- `print(...)` : This is the action the function performs.
- The indented block under the `def` line is the **function body** — it only runs when the function is called.

B. Calling the Function

```
[32] greet("Juliana")    # Output: Hello, Juliana
```

```
Hello, Juliana
```

Explanation:

- This line **calls** or **invokes** the `greet()` function.
- The string `"Juliana"` is passed to the parameter `name`.
- The function prints a greeting with the name inserted.

Key point: **Defining a function does not execute it** — it must be called.

Writing Functions

C. Returning a Value Instead of Printing

```
[33]  def  scale_score(score):
           return  score  /  10

      print(scale_score(87))    #  Output:  8.7
```

```
8.7
```

Explanation:

- `return`: This sends a result back to where the function was called, like a calculator.
- `score / 10`: This calculation is performed inside the function.
- `print(...)`: This shows the returned result.

Use case: Return is better than print when you want to **use the result later** in calculations or other logic.

Writing Functions

What is `.apply()` in a DataFrame?

The `.apply()` function lets you **apply a custom function to each element of a column** (or a row if specified). It's commonly used to:

- Transform data
- Perform calculations
- Apply logic or formatting
- Create new columns based on existing ones

Why use `.apply()` instead of a loop?

- **Cleaner syntax:** No need for writing `for` loops.
- **Faster performance:** pandas is optimized for vectorized operations.
- **More readable and reusable:** Code is easier to maintain.

D. Using Functions with DataFrames (.apply())

Create Data + Apply Custom Function

```
[45] import pandas as pd

     # Step 1: Create a sample dataset
     data = {
            "name": ["Alice", "Bob", "Charlie", "Diana"],
            "score": [78, 85, 92, 66]
     }

     df = pd.DataFrame(data)

     # Step 2: Define a custom function
     def double(x):
            return x * 2

     # Step 3: Apply the function to the 'score' column
     df["score_doubled"] = df["score"].apply(double)

     # Step 4: Show the updated DataFrame
     print(df)
```

```
      name  score  score_doubled
0    Alice     78            156
1      Bob     85            170
2  Charlie     92            184
3    Diana     66            132
```

Writing Functions

```
[36] import pandas as pd

     # Sample data
     data = {
         "name": ["Alice", "Bob", "Charlie"],
         "raw_score": [64, 72, 58]
     }

     df = pd.DataFrame(data)

     # Define the conversion function
     def to_percentage(raw_score):
         return round((raw_score / 80) * 100, 2)

     # Apply the function to create a new column
     df["score_percent"] = df["raw_score"].apply(to_percentage)

     # Display the result
     print(df)
```

```
      name  raw_score  score_percent
0    Alice         64           80.0
1      Bob         72           90.0
2  Charlie         58           72.5
```

Application Example: Converting Exam Scores to Percentages

Scenario:

You are analyzing student exam scores stored in a CSV file. The scores are out of **80 points**, but your school's grading system requires them to be shown as **percentages** out of 100.

You can write a function to scale the scores and use `.apply()` to convert every student's score.

Explanation:

- `to_percentage()` is a reusable function that converts raw scores to percentages.
- `.apply(to_percentage)` runs the function on every value in the `raw_score` column.
- `df["score_percent"]` is a new column that stores the converted percentages.

Why It Matters:

- This technique automates repetitive column-wise transformations.
- It avoids writing for-loops and makes your code cleaner and faster.
- You can easily swap in a new function later without changing the structure of your DataFrame operations.

07

Debugging and Error Handling

Debugging and Error Handling in Python

Why Debugging Matters

Everyone encounters errors — even pros!

Understanding **why code breaks** is key to becoming confident in Python

Common Errors and How to Fix Them

Error Type	Example Trigger	Fix
SyntaxError	Missing parentheses, unmatched quotes	Check line carefully
TypeError	Adding string + int without conversion	Use str() or int()
NameError	Variable not defined	Check variable name spelling
IndentationError	Misaligned code blocks	Use consistent 4 spaces
FileNotFoundError	File path is wrong or not uploaded/mounted in Colab	Upload or mount correctly

Debugging and Error Handling in Python

Example: TypeError

```
[46] age = 25
     # print("Age: " + age)  ❌
     print("Age: " + str(age))    # ✅
```

```
Age: 25
```

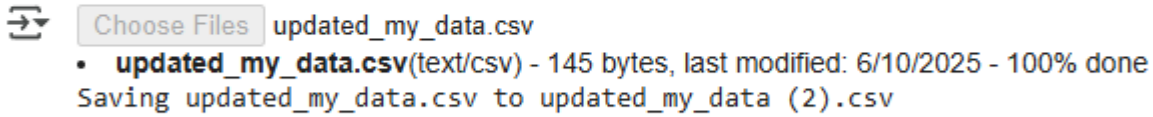
Explanation: You can't use + to combine a string with an integer.

Example: FileNotFoundError

```
[49] # ❌ Won't work in Colab unless uploaded
     df = pd.read_csv("updated_my_data.csv")
```

Fix:

```
[50] from google.colab import files
     files.upload()
     df = pd.read_csv("my_data.csv")    # ✅ Now it works
```



Colab has no access to your C:\ drive unless you upload the file.

Tips for Debugging

- **Read the full error message**: Focus on the last line
- **Use print() statements** to track variable values
- **Run your code step by step**: Don't try to fix the whole script at once
- **Ask AI to help interpret an error**

08

Wrap-Up and Learning Beyond the Classroom

What's Next? Tools, Resources & AI for Python Learning

Key Takeaways from Today

You ran Python code in **Google Colab** — no installation needed

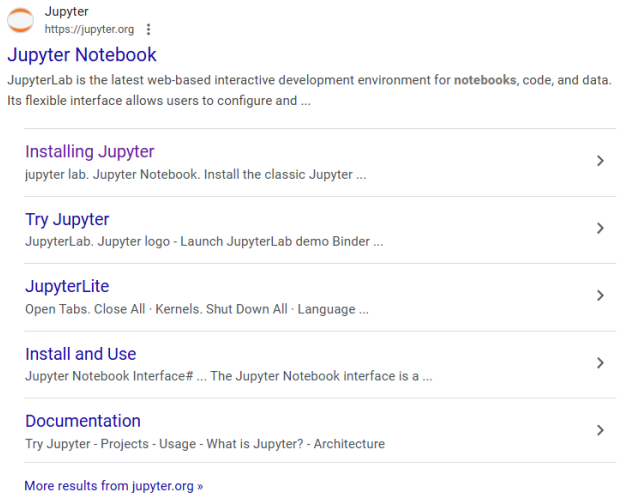
You learned:

- Core Python syntax: print(), variables, strings, numbers
- Loops and conditional logic
- How to import .csv data with pandas
- How to define your own functions
- How to troubleshoot errors and file path issues

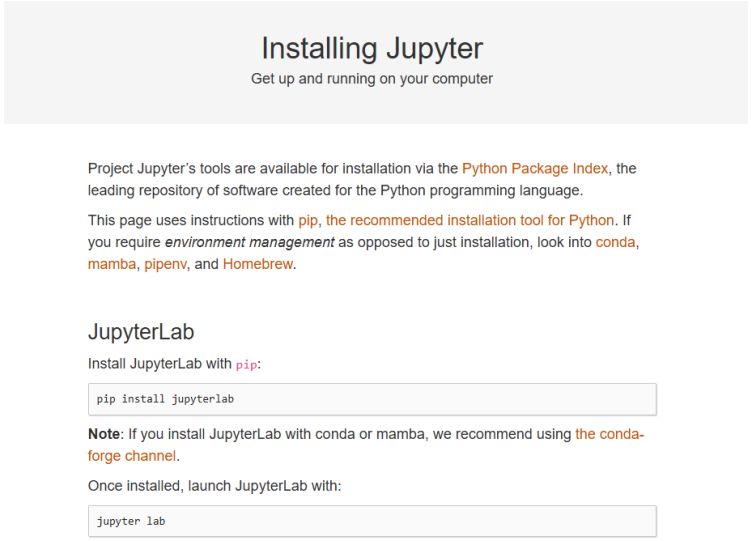
Helpful Tools to Explore Next

Jupyter Notebook

A flexible and powerful tool for creating and sharing documents that contain live code, equations, visualizations, and text.



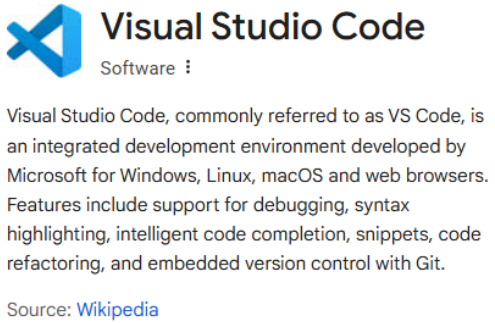
https://jupyter.org/ (Jupyter, 2025)

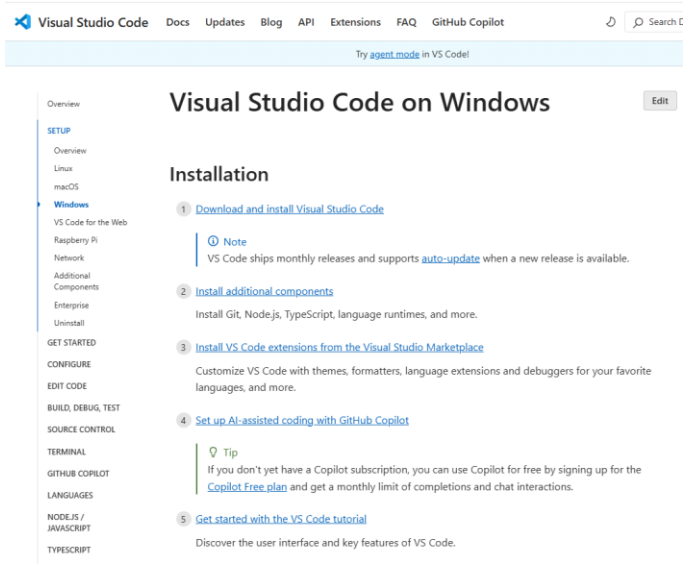


https://jupyter.org/install (Jupyter, 2025)

Visual Studio Code (VS Code)

A lightweight yet powerful source-code editor that supports Python and other languages.





https://code.visualstudio.com/docs/setup/windows (Visual Studio Code, 2025)

How to use ChatGPT for continuous learning

How to ask better prompts:

- Be clear about **what you want** and **what your code is doing**
- Include code **and** error messages when asking for help

Examples:

- "*Summarize this Python code and explain what it does:*" (then paste code)
- "*Generate practice exercises for Python lists.*"
- "*What's the best way to debug a 'TypeError' in Python?*"
- *"Please help me adapt the following python code to my current context* (then paste code from lecture)"

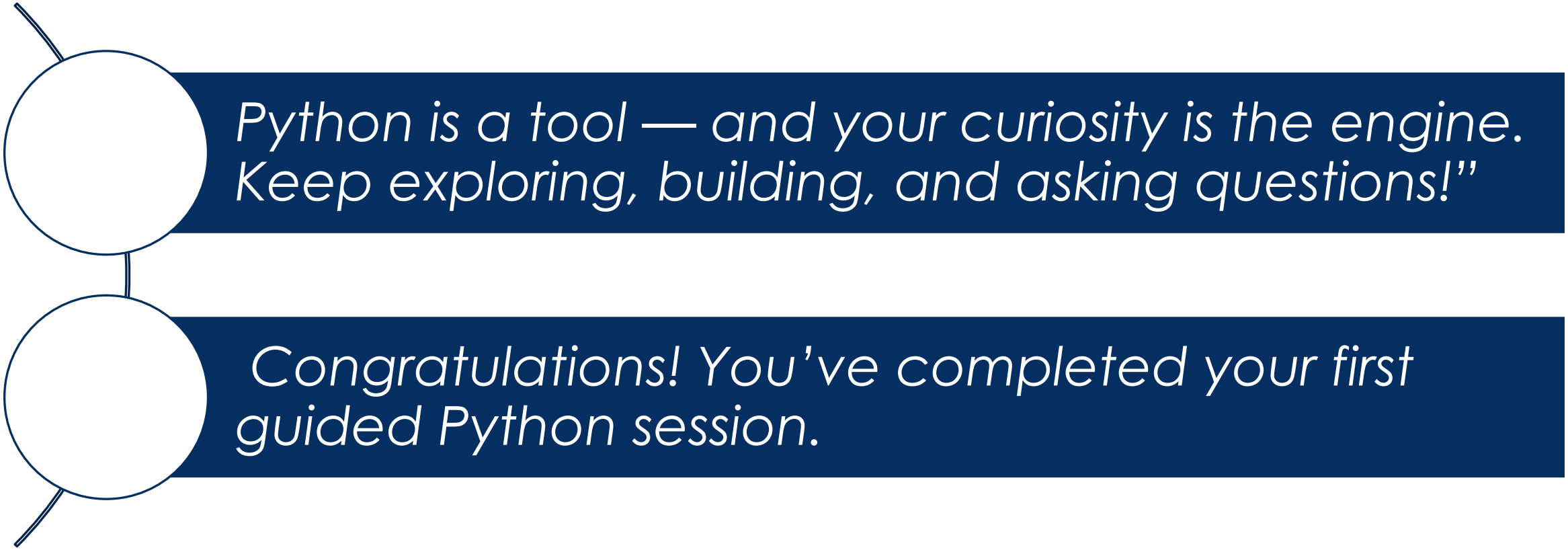
Sharing & Downloading Your Notebook

Download notebook: File → Download → .ipynb or .py

Share notebook via Google Drive link:

- e.g. https://colab.research.google.com/drive/1c85JWtCKRN7qDSvzerCtPJHzBVbSLqkG
- You can email this link or upload it to GitHub

Closing Message



Reference

Google. (2025). *Create a Google Account*. Retrieved from Google: https://support.google.com/accounts/answer/27441?hl=en&co=GENIE.Platform%3DAndroid

Jupyter. (2025). *Installing Jupyter*. Retrieved from Jupyter: https://jupyter.org/install

Jupyter. (2025). *Jupyter:Free software, open standards, and web services for interactive computing across all programming languages*. Retrieved from Jupyter: https://jupyter.org/

Maheta, D. (2025, Feb 28). *Python for Automation: Supercharge Your Productivity*. Retrieved from Bacancy: https://www.bacancytechnology.com/blog/python-for-automation

Okutan, P. (2024, July 4). *How To Perform Statistical Analysis Using Python*. Retrieved from Medium: https://blog.stackademic.com/how-to-perform-statistical-analysis-using-python-de49a894506b

Ozgon, D. (2021). *Google Colab Tutorial for Beginners | Get Started with Google Colab*. Retrieved from YouTube: https://www.youtube.com/watch?v=RLYoEyIHL6A

Thinks, A. (2022). *How to Open Google Colab Notebook (.ipynb) from Google Drive* . Retrieved from YouTube: https://www.youtube.com/watch?v=Iy3XB2e99R4

Visual Studio Code. (2025, Aug 05). *Visual Studio Code on Windows*. Retrieved from Visual Studio Code: https://code.visualstudio.com/docs/setup/windows

THANKS

Juliana Yuncg (Qi Xuan Yuncg)
PhD Student

Contact
Tel: (852) 2616 7676
Fax: (852) 2456 0737
Email: qixuanyuncg@ln.hk

Office
WYL104, Dorothy Y L Wong Building
Department of Sociology and Social Policy
Lingnan University, Tuen Mun, Hong Kong

Summer Institute in Computational Social Science

(SICSS-Lingnan, June 16 to June 27, 2025 | Hong Kong SAR)