

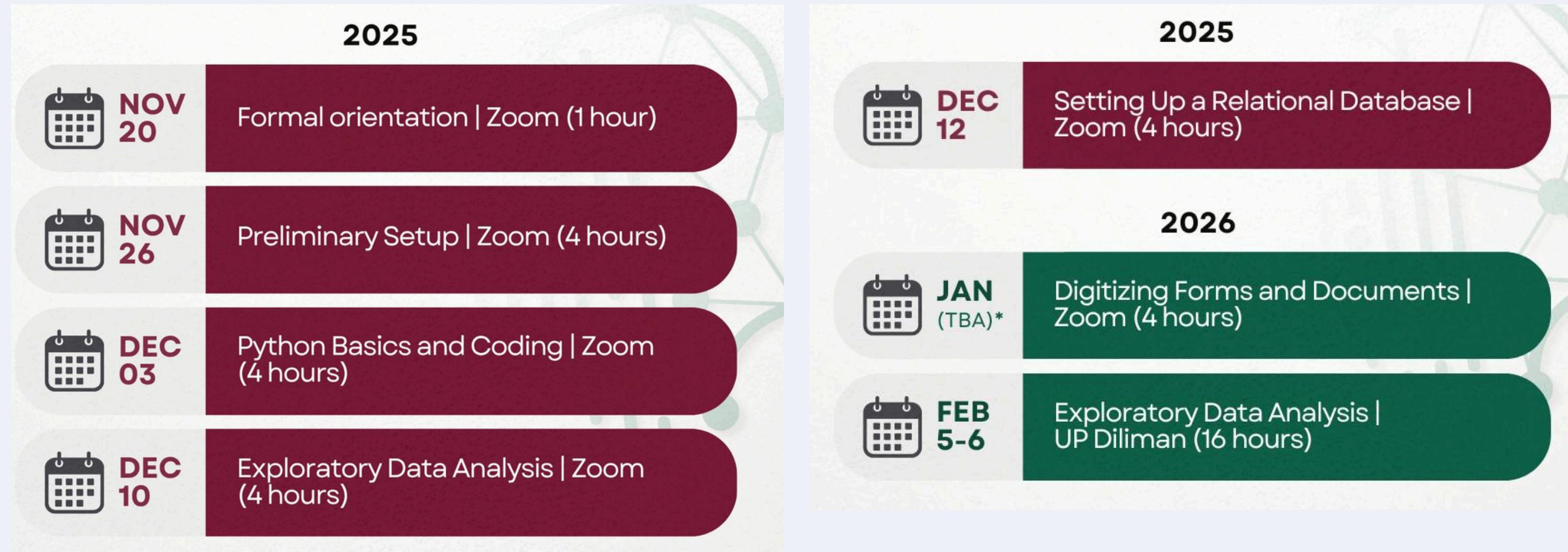
# **PREPARATORY LEARNING SESSION: PRELIMINARY SETUP**

**BIG GOAL FOR THIS WORKSHOP SERIES:**

## **Dx Labs: AI for Institutional Transformation in UP**

Source: [ovpdx.up.edu.ph](http://ovpdx.up.edu.ph)

# BIG GOAL FOR THIS WORKSHOP SERIES:



Source: [ovpdx.up.edu.ph](http://ovpdx.up.edu.ph)

# BIG GOAL FOR THIS WORKSHOP SERIES:

2025		2025	
	<b>NOV 20</b>	Formal orientation   Zoom (1 hour)	
	<b>NOV 26</b>	Preliminary Setup   Zoom (4 hours)	
	<b>DEC 03</b>	Python Basics and Coding   Zoom (4 hours)	
	<b>DEC 10</b>	Exploratory Data Analysis   Zoom (4 hours)	
2025		2026	
	<b>DEC 12</b>	Setting Up a Relational Database   Zoom (4 hours)	
	<b>JAN (TBA)*</b>	Digitizing Forms and Documents   Zoom (4 hours)	
	<b>FEB 5-6</b>	Exploratory Data Analysis   UP Diliman (16 hours)	

Source: [ovpdx.up.edu.ph](http://ovpdx.up.edu.ph)

# **GOAL FOR THE FIRST DAY**

To ease participants into programming.

# **GOAL FOR THE FIRST DAY**

To ease participants into programming.

**NOTE:**

- We don't need to learn everything.
- We don't need to code from scratch.

# SPECIFIC GOALS

By the end of this session, the participants will:

- have their development tools installed
- understand how to use AI coding assistants
- write basic Python code



# WHY PYTHON?

- One of the most popular programming language
- Simple syntax: easy to learn
- open source
- general-purpose
- great interactive environment

---

 Use Python for...

» More

---

**Web Development:** [Django](#) , [Pyramid](#) , [Bottle](#) , [Tornado](#) , [Flask](#) , [Litestar](#) , [web2py](#)

---

**GUI Development:** [tkInter](#) , [PyGObject](#) , [PyQt](#) , [PySide](#) , [Kivy](#) , [wxPython](#) , [DearPyGui](#)

---

**Scientific and Numeric:** [SciPy](#) , [Pandas](#) , [IPython](#)

---

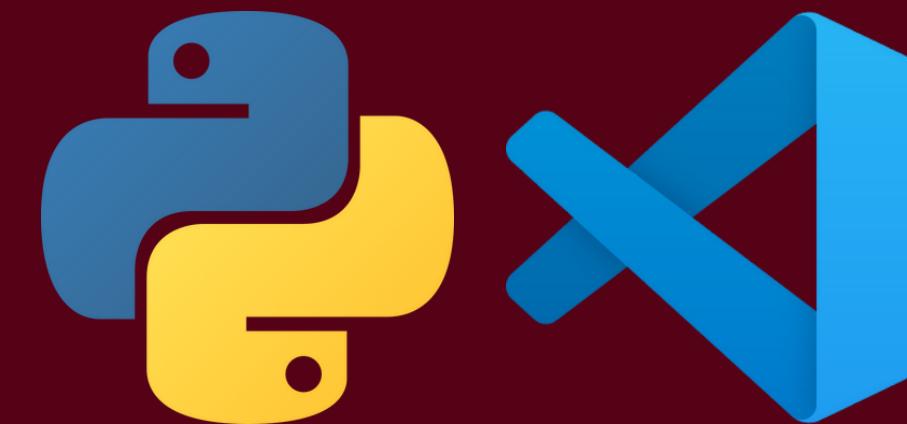
**Software Development:** [Buildbot](#) , [Trac](#) , [Roundup](#)

---

**System Administration:** [Ansible](#) , [Salt](#) , [OpenStack](#) , [xonsh](#)

Source: <https://www.python.org/>

# SETTING UP THE DEVELOPMENT ENVIRONMENT



# SETTING UP OUR TOOLS

- Refer to our [pre-workshop setup guide](#)



## Pre-Workshop Setup Guide

Installing Python 3, Visual Studio Code (VS Code) and Github  
(For both Windows and macOS users)

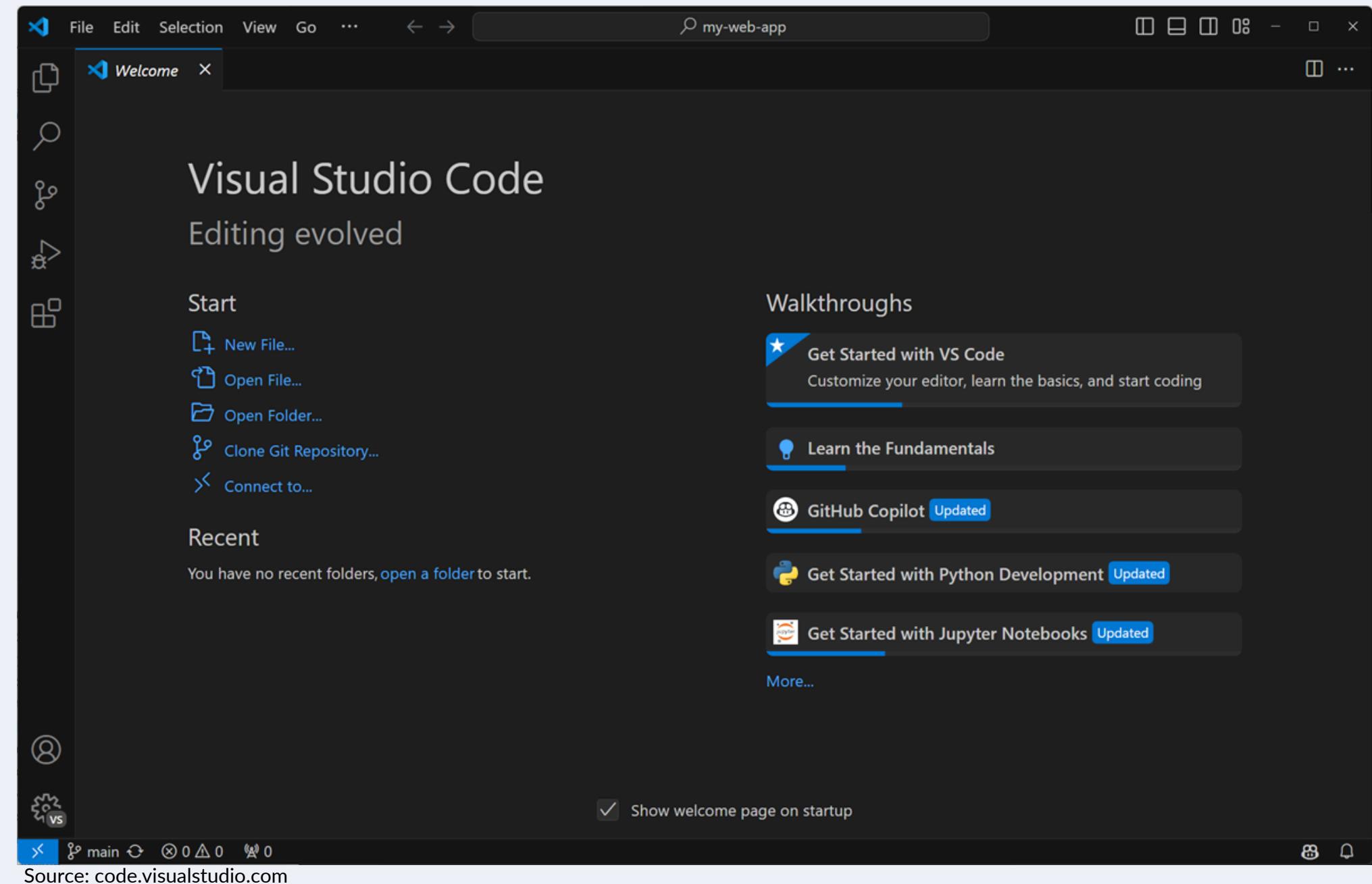
This guide will help you get ready for our Python for Administrative Staff Workshop. Please follow these steps before the training session so you're ready to start coding right away!

Before proceeding, the participants are assumed to have successfully installed the needed tools following the pre-workshop setup guide

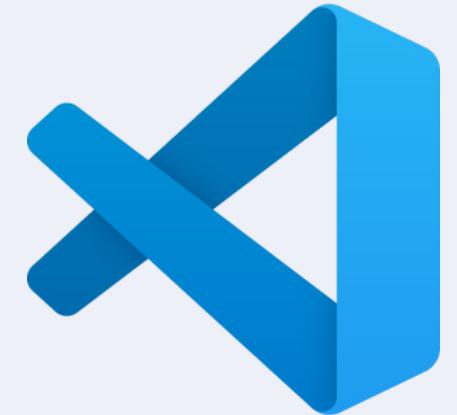
# MICROSOFT VISUAL STUDIO CODE



- This is the main app we will use to write, edit and run our Python codes



# MICROSOFT VISUAL STUDIO CODE



The screenshot shows the Microsoft Visual Studio Code interface with various components labeled:

- A Activity Bar**: The left sidebar containing the Explorer, Search, Problems, and Timeline sections.
- B Primary Side Bar**: The bottom-left bar with file navigation icons.
- C Editor Groups**: Two side-by-side code editors showing `arrays.ts` and `package.json`.
- D Panel**: The bottom center panel showing a list of files in the current directory.
- E Status Bar**: The bottom right bar displaying file path, line/col, tab size, encoding, and file type.

**Code Editor Content (arrays.ts):**

```
src > vs > base > common > TS arrays.ts > ...
1  /*
2   * Copyright (c) Microsoft Corporation. All rights reserved.
3   * Licensed under the MIT License. See License.txt in the project root for more information.
4   */
5
6  import { CancellationToken } from 'vs/base/common/cancellation';
7  import { CancellationError } from 'vs/base/common/errors';
8  import { ISlice } from 'vs/base/common/sequence';
9  import { findFirstIdxMonotonicOrArrLen } from './arraysFind';
10
11 /**
12  * Returns the last element of an array.
13  * @param array The array.
14  * @param n Which element from the end (default is zero).
15  */
16 export function tail<T>(array: ArrayLike<T>, n: number = 0): T {
17     return array[array.length - (1 + n)];
18 }
19
20 export function tail2<T>(arr: T[]): [T[], T] {
21     if (arr.length === 0) {
22         throw new Error('Invalid tail call');
23     }
}
```

**Panel Content (List of files):**

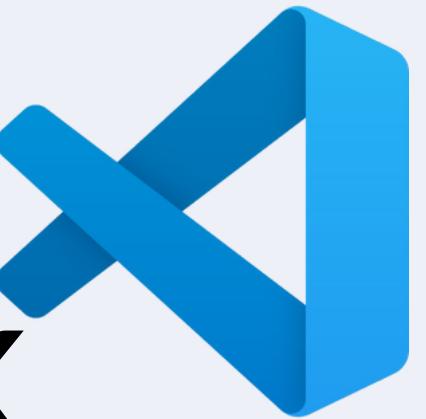
-a----	2/8/2024 2:44 PM	9749 package.json
-a----	2/8/2024 2:44 PM	3185 product.json
-a----	12/20/2023 11:33 AM	6939 README.md
-a----	12/20/2023 11:33 AM	2807 SECURITY.md
-a----	2/8/2024 2:44 PM	147681 ThirdPartyNotices.txt
-a----	12/20/2023 11:34 AM	763 tsfmt.json
-a----	2/8/2024 2:45 PM	466744 yarn.lock

Source: [code.visualstudio.com](https://code.visualstudio.com)

# MICROSOFT VISUAL STUDIO CODE



1. Launch VS Code.
2. Create a new folder on your desktop, e.g. **DxLabs\_Week01**
3. Inside VS Code, open this folder (**File --> Open Folder**).
4. Create a new file named **test\_notebook.ipynb** (Python Notebook).



# INTRODUCTION TO PYTHON NOTEBOOK

- rich text + runnable code
- ideal for documentation and sharing

## Running Code

First and foremost, the Jupyter Notebook is an interactive environment for writing and running code. The notebook is capable of running code in a wide range of languages. However, each notebook is associated with a single kernel. This notebook is associated with the IPython kernel, therefore runs Python code.

### Code cells allow you to enter and run code

Run a code cell using `Shift-Enter` or pressing the button in the toolbar above:

In [1...]

```
a = 10
```

In [2...]

```
print(a)
```

```
10
```

There are two other keyboard shortcuts for running code:

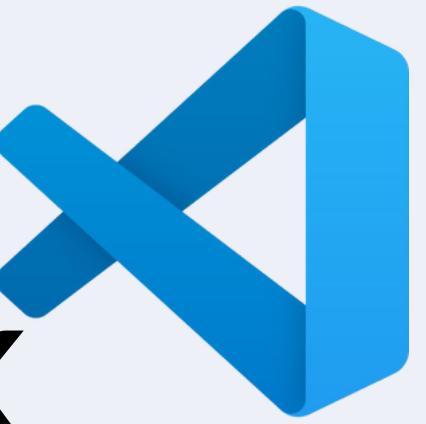
- `Alt-Enter` runs the current cell and inserts a new one below.
- `Ctrl-Enter` run the current cell and enters command mode.

## Managing the Kernel

Code is run in a separate process called the Kernel. The Kernel can be interrupted or restarted.

Try running the following cell and then hit the button in the toolbar above.

Source: <https://www.nbviewer.org/>



# INTRODUCTION TO PYTHON NOTEBOOK

1. In the VS Code Editor, click + Code.
2. In the resulting cell, type the ff:
  - `print("Hello, World!")`
3. Press ► Run Cell (or Shift+ENTER) to see the result.

A screenshot of the Microsoft Visual Studio Code interface, specifically showing a Jupyter Notebook environment. The top bar shows the file path "test\_notebook.ipynb > print("Hello, World!")". Below the menu bar, there are buttons for "Generate", "+ Code", "+ Markdown", "Run All", "Restart", "Clear All Outputs", "Jupyter Variables", "Outline", and three dots. A toolbar on the right includes icons for "Python 3.14.0", "Cell", "Run", "Kernel", "Help", and "Close". The main area displays a single code cell containing the Python command `print("Hello, World!")`. The cell has a green header indicating it is ready to run. The status bar at the bottom right shows "Python".



# WITHIN PYTHON NOTEBOOK

- A cell is a place where we can write in our code or text.
- Two main types of cells:
  - Code cell
  - Markdown cell

A screenshot of a Jupyter Notebook interface. At the top, there's a menu bar with options like 'Generate', '+ Code', '+ Markdown', 'Run All', 'Restart', 'Clear All Outputs', 'Jupyter Variables', 'Outline', and 'Python 3.14.0'. Below the menu is a toolbar with buttons for 'Generate', '+ Code', and '+ Markdown'. The main area shows a code cell containing the Python command `print("Hello, World!")`. To the right of this cell, a yellow callout bubble contains the text 'this is a code cell'. A blue arrow points from this callout towards the code cell. The bottom right corner of the interface says 'Python'.

# WITHIN PYTHON NOTEBOOK

Raw markdown cell

```
# This is a header (level 1)
## This is a header (level 2)
```

For **bold** text

For *italicized* text

For `bulleted list`:

1. item 1
2. item 2

Rendered markdown cell  
(after pressing SHIFT+ENTER)

This is a header (level 1)

This is a header (level 2)

For **bold** text

For *italicized* text

For **bulleted list**:

1. item 1
2. item 2

# WITHIN PYTHON NOTEBOOK

- Two types of mode:
  - Edit mode - we can type in our codes or markdown text.
  - Command mode - perform commands like adding a cell above or below an active cell, changing the type of cell, etc
- Command to Edit
  - Click inside the cell
  - Or Click on a cell then press ENTER
- Edit to Command
  - Press ESC

EDIT MODE



A screenshot of a Jupyter Notebook interface. A single code cell is visible, containing the Python command `print("Hello, World!")`. The cell has a blue vertical selection bar on its left and a blue horizontal bar at the top. The text is in a monospaced font. The word "print" is in brown, and the string "Hello, World!" is in red. To the left of the cell, there are two small icons: a downward arrow with a right arrow to its right, and a bracket icon [ ].

# WITHIN PYTHON NOTEBOOK

- Code cells can be executed multiple times and in any order
- An execution number is associated to each executed code cell  
This is used to remember which code cells are executed first.
- When a code cell is not yet executed, it will have no execution number.

```
print("Hello, World!")  
✓ 0.0s  
... Hello, World!
```

This is a header (level 1)

This is a header (level 2)

For bold text

For italicized text

For **bulleted list**:

1. item 1
2. item 2

```
print("Hello, Universe!")  
✓ 0.0s  
... Hello, Universe!
```

# WITHIN PYTHON NOTEBOOK

- Code cells can be executed multiple times and in any order
- An execution number is associated to each executed code cell  
This is used to remember which code cells are executed first.
- When a code cell is not yet executed, it will have no execution number.



# WITHIN PYTHON NOTEBOOK

Python notebook specific options





# INTRODUCING COPILOT

- An AI tool that suggests code as you type
- Built on large AI models trained on public code
- Auto-complete lines and entire functions
- Generate code from comments or prompts
- Recommend fixes and improvements



# INTRODUCING COPILOT

Verify your Copilot Usage  
by clicking the Copilot  
icon in the status bar

The screenshot shows a sidebar for 'Copilot Usage' with the following data:

Category	Percentage
Inline Suggestions	19.6%
Chat messages	16%

Below the chart, it says "Allowance resets December 13, 2025." The sidebar also includes sections for 'Workspace Index' (Locally indexed) and 'Inline Suggestions' (checkboxes for All files, Python, Next edit suggestions). At the bottom, there are 'Snooze' and 'Hide suggestions for 5 min' buttons. The status bar at the bottom of the sidebar shows "Spaces: 4 LF () Cell 7 of 7" with a red circle around the Copilot icon (a robot head).



# INTRODUCING COPILOT

via inline suggestions

this is our code

A screenshot of a code editor interface. On the left, there's a dropdown menu icon and a closing bracket icon. In the center, a suggestion box is open, containing the text '# choose a random name from a list' in green and 'import random' in purple. A red arrow points from the text 'this is our code' to the suggestion box. Another red arrow points from the text 'this is a suggestion also called *ghost code*' to the word 'random' in the suggestion box.

```
# choose a random name from a list
import random
```

this is a suggestion  
also called *ghost code*



# INTRODUCING COPILOT

via inline suggestions

this is our code

A screenshot of a code editor interface. A blue rectangular suggestion box is open at the bottom of the screen, containing the following code:

```
# choose a random name from a list
import random
```

Two red arrows point to the suggestion box: one from the text "this is our code" above it, and another from the text "this is a suggestion also called *ghost code*" below it.

this is a suggestion  
also called *ghost code*

Press TAB to accept  
Press ESC to reject



# INTRODUCING COPILOT

via inline suggestions

This is a sample code suggestion

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

Note!

You might see different suggestions because large language models are nondeterministic.



# INTRODUCING COPILOT

via inline suggestions

If you see multiple suggestions, hover over the ghost text to see navigation controls, or use ALT+] and ALT+[ to cycle through options.

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")

# loop over random names and print them
```

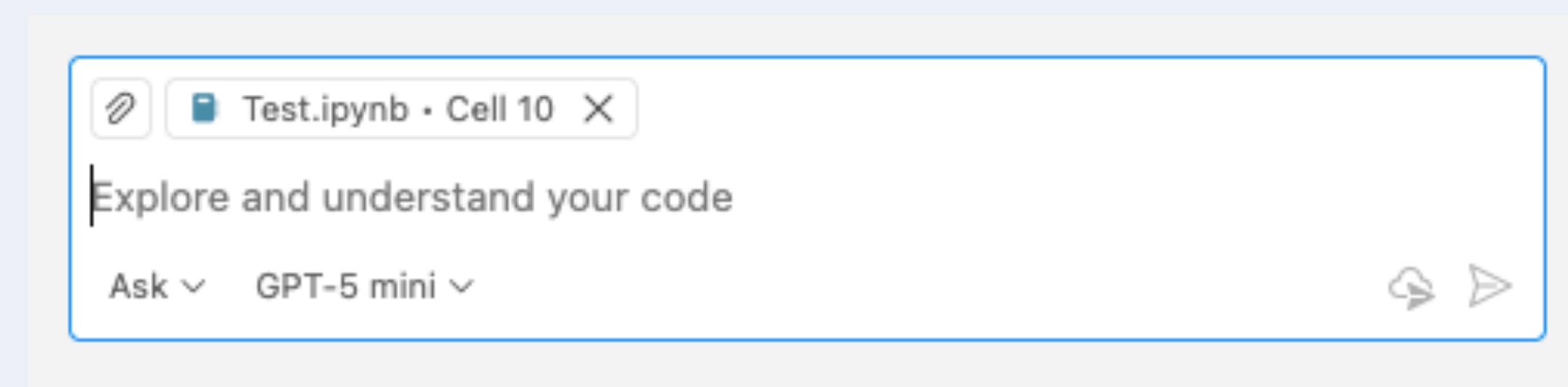
< 1/2 > Accept Tab Accept Word ⌘ ⌘ ...



# INTRODUCING COPILOT via Chat

Open the Chat view by pressing SHIFT+CMD+I or by selecting the chat icon in the VS Code title bar.

The Chat view enables you to have an ongoing conversation with the AI, making it easier to refine your requests and get better results.





# INTRODUCING COPILOT

via inline chat

We can also use copilot for targeted improvements to specific code sections within a file



# INTRODUCING COPILOT

via inline chat

Highlight the code section you want to improve, then press CMD+I to access the inline chat

```
# choose a random name from a list

import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

Ask or edit in context

Auto ▾



# INTRODUCING COPILOT

via inline chat

the phrase we typed in  
the inline chat



highlighted in green are  
the suggested code edit

```
# choose a random name from a list
create a timer before printing random name

Ask or edit in context

Accept Close Accept & Run ⌂ ▾

import random
import time
names = ["Alice", "Bob", "Charlie", "Diana"]

# countdown timer before printing the random name
countdown = 3
for i in range(countdown, 0, -1):
    print(f"Printing random name in {i}...")
    time.sleep(1)

random_name = random.choice(names)
```



# INTRODUCING COPILOT

via inline chat

press Accept to adopt  
the code edit

# choose a random name from a list

⟳ create a timer before printing random name

Ask or edit in context

Accept Close Accept & Run ⌂ ⌂

```
import random
import time
names = ["Alice", "Bob", "Charlie", "Diana"]
```

# countdown timer before printing the random name

```
countdown = 3
for i in range(countdown, 0, -1):
    print(f"Printing random name in {i}...")
    time.sleep(1)
```

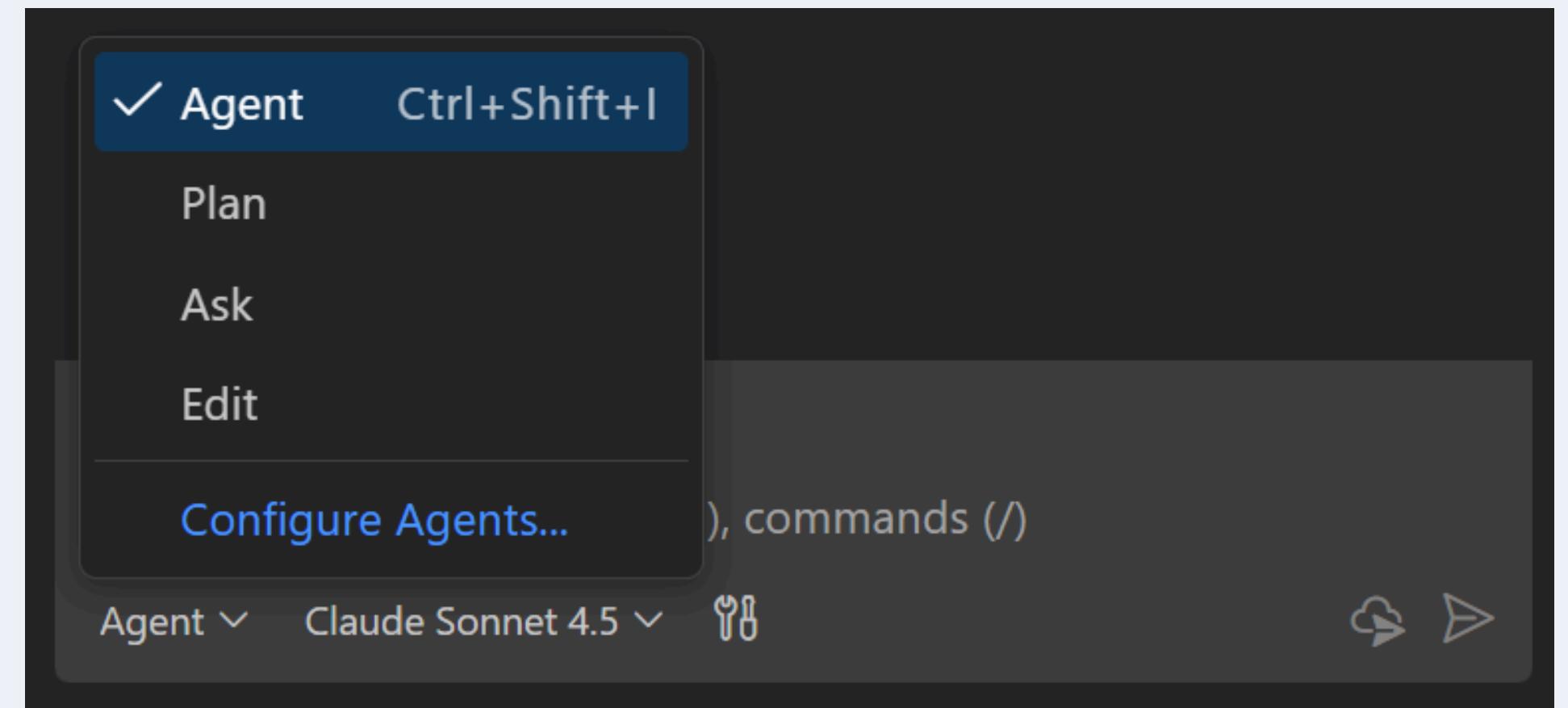
```
random_name = random.choice(names)
```



# INTRODUCING COPILOT

via agent

Agents are VS Code's strongest AI tool, autonomously planning and applying complex changes across files from a simple prompt.



# INTRODUCING COPILOT

Agents are VS Code's strongest AI tool, autonomously planning and applying complex changes across files from a simple prompt.

Create a complete random name picker Python web application with the ability to get names as input from user, then select a random name after a 5-second countdown.

- ## ✓ Inspecting files

I'll check the current project files to see what's present and restore or recreate the Flask app if needed. I'll read the folder listing and any existing files.

- ✓ Searched for files matching `**/random_name_picker.py/**`, no matches

- ✓ Executing tool calls with updates...

Now I'll create a TODO plan, then add the Flask app files (template + static) and docs.

- ✓ Created 3 todos

app-DV +26 -1

index.html +29 -1

JS app is +82 -1

# style.css +33 -1

三 requirements.txt +3 -1

① README.md +28 -1

# SUMMARY (SO FAR)

We learned about:

- Basic navigation on Visual Studio Code
- How to work on Jupyter notebooks
- How to use Github Copilot

# PYTHON BASICS

OBJECTIVE: Learn how Python syntax  
works through simple examples

# HELLO WORLD

- **print()**  
print the output of  
whatever is inside the  
open and close  
parentheses
- “Hello, World”  
and example of a data  
type called a **string**.



A screenshot of a terminal window. At the top, there's a toolbar with a play button and a dropdown menu. Below that, the command `print("Hello, World!")` is typed in. To the left of the command, the number `[1]` is displayed. To the right, a green checkmark and the time `0.0s` are shown. At the bottom of the window, the output of the command is displayed: `... Hello, World!`

# HELLO WORLD

- Python's way to recognize plain text
- Any characters enclosed in quotations (either single or double quotation) will be treated by Python as a string.

```
▶ v
      print("Hello, World!")
[1] ✓ 0.0s
...
... Hello, World!
```

# HELLO WORLD

- What happens if we mistyped the code?

```
[5]     print("Hello, World!")
      ⚡  0.0s
...
...   Cell In[5], line 1
      print("Hello, World!")
      ^
_IncompleteInputError: incomplete input

[8]     print(Hello, World!)
      ⚡  0.0s
...
...   Cell In[8], line 1
      print(Hello, World!)
      ^
SyntaxError: invalid syntax

▷ v [7]     prin("Hello, World!")
      ⚡  0.1s
...
...
-----                                         Traceback (most recent call last)
NameError
Cell In[7], line 1
----> 1 prin("Hello, World!")

NameError: name 'prin' is not defined
```

# HELLO WORLD

- We can have multiple commands in one code cell.



A screenshot of a Jupyter Notebook cell. The cell contains the following Python code:

```
print("Hello, Philippines")
print("and")
print("Hello, World!")
```

The cell has a status bar indicating [10] and 0.0s. Below the cell, the output is shown:

... Hello, Philippines  
and  
Hello, World!

# COMMENTS

- starts with #
- No effect on the execution of codes



A screenshot of a terminal window illustrating comments in code. The window shows the following text:

```
# this is a comment
print("Hello, World!")
[11] ✓ 0.0s
... Hello, World!
```

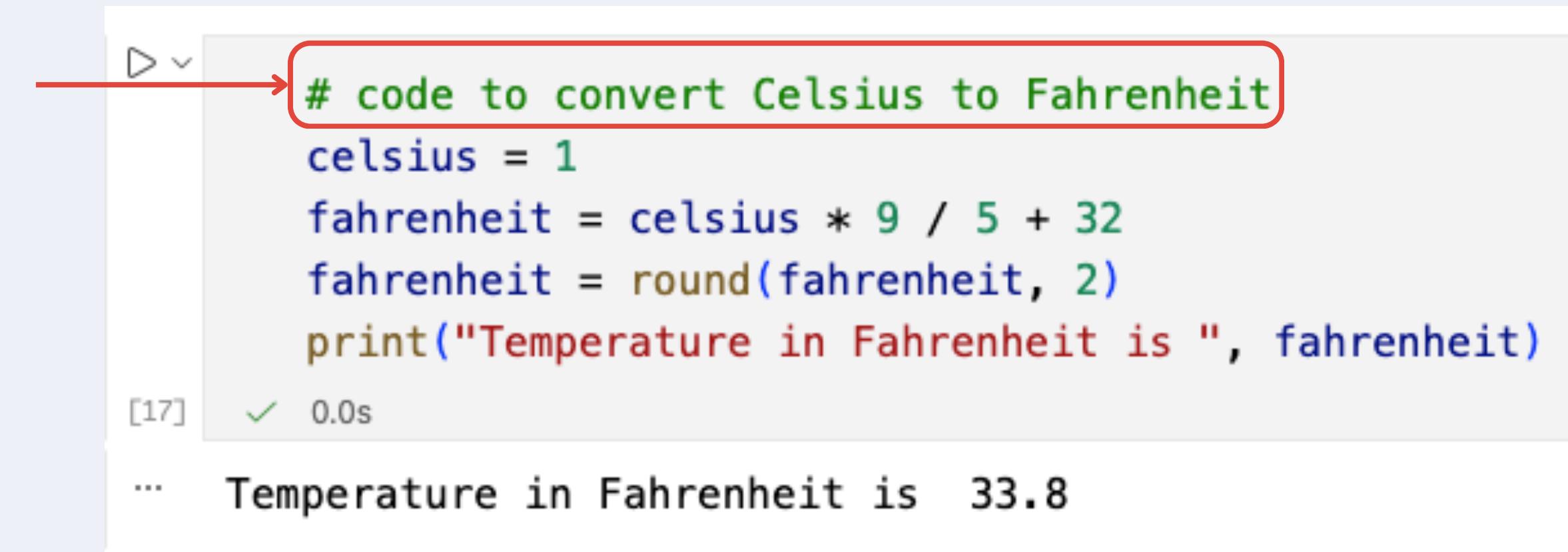
The first line, '# this is a comment', is colored green, indicating it is a comment. The second line, 'print("Hello, World!")', is colored blue and red, indicating it is executable code. The output '[11] ✓ 0.0s' is shown below the code, followed by an ellipsis and the printed output 'Hello, World!'. The terminal has a light gray background and a white input field.

# WORKING ON FORMULAS

```
▶ v
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
[17] ✓ 0.0s
...
... Temperature in Fahrenheit is 33.8
```

# WORKING ON FORMULAS

comment, no effect in code



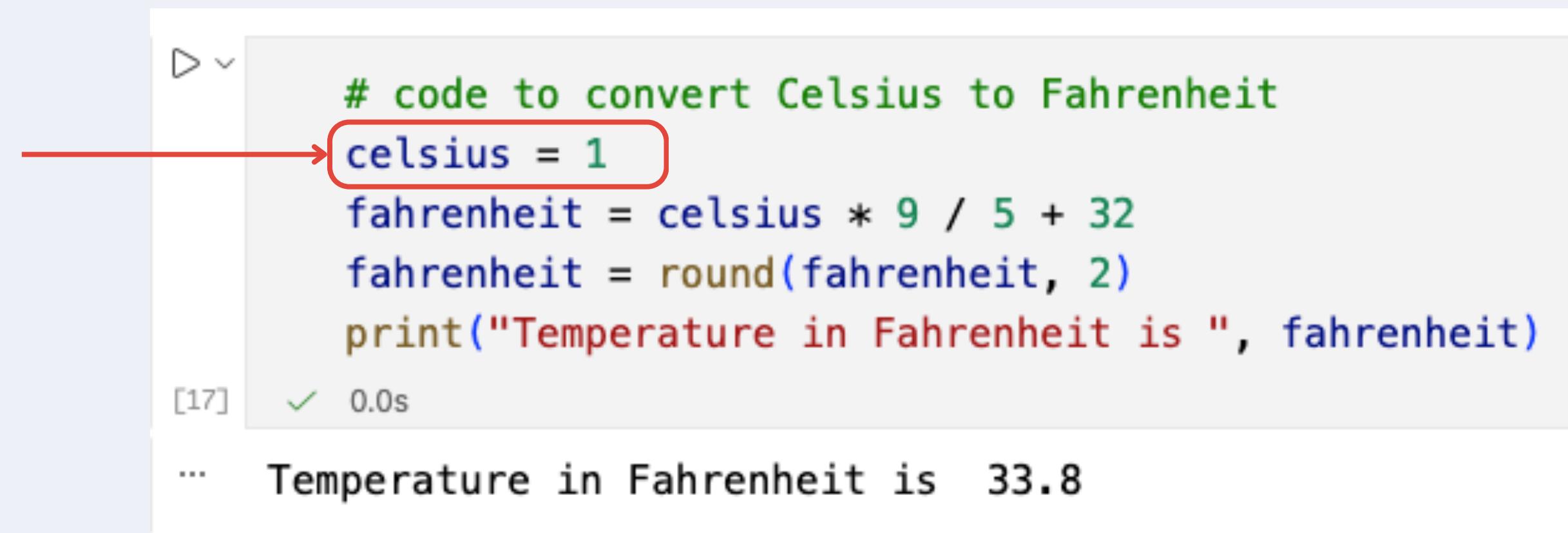
```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
```

[17] ✓ 0.0s

... Temperature in Fahrenheit is 33.8

# WORKING ON FORMULAS

declare a variable named  
celsius  
set its value to 1



The screenshot shows a Jupyter Notebook cell with the following code:

```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
```

The variable `celsius = 1` is highlighted with a red box and an arrow points from the text "set its value to 1" to it. The output of the cell is:

[17] ✓ 0.0s

... Temperature in Fahrenheit is 33.8

# WORKING ON FORMULAS

- A variable is a named value (e.g. name, age, speed, area).
- We can assign a value to a variable using an assignment statement represented by the equal symbol (=).

## WARNING:

- The symbol (=) means assignment (not equality). Assignment is not symmetric. Python assignment always go left.

# WORKING ON FORMULAS

- Why use variables?

```
1 print("Juan is a student")
2 print("Juan is 16 years old")
3 print("Juan likes programming")
```

Juan is a student  
Juan is 16 years old  
Juan likes programming

```
1 name = "Juan"
2 print(name, "is a student")
3 print(name, "is 16 years old")
4 print(name, "likes programming")
```

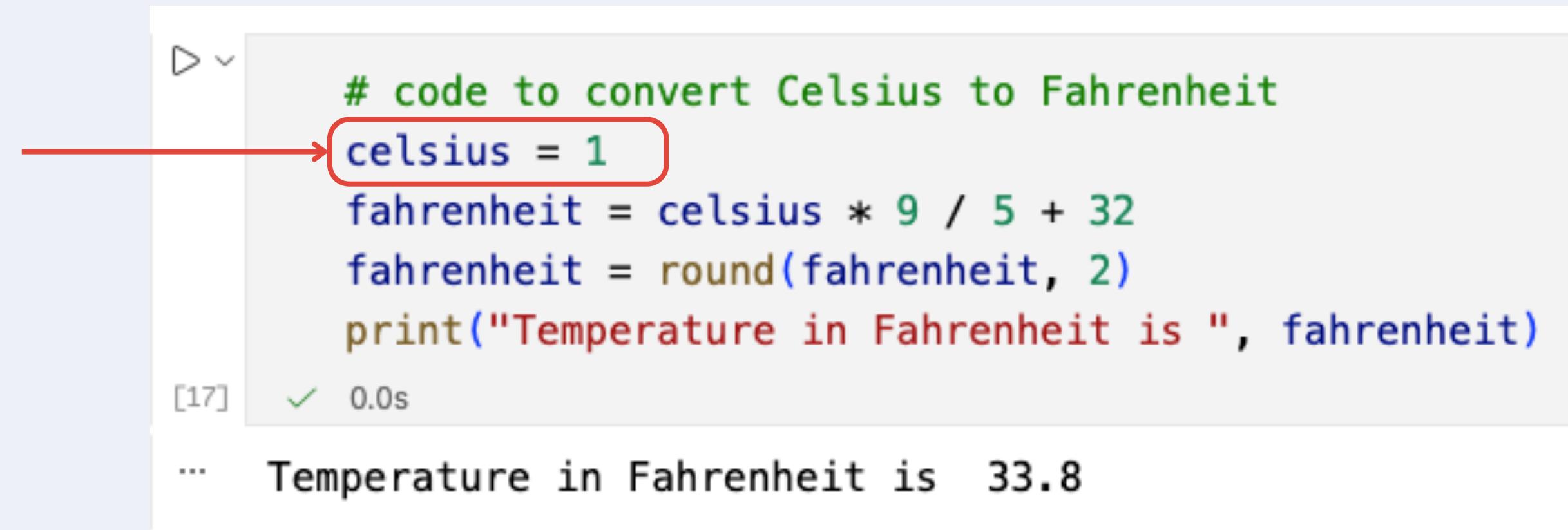
Juan is a student  
Juan is 16 years old  
Juan likes programming

```
1 name = "Joe"
2 print(name, "is a student")
3 print(name, "is 16 years old")
4 print(name, "likes programming")
```

Joe is a student  
Joe is 16 years old  
Joe likes programming

# WORKING ON FORMULAS

declare a variable named  
celsius  
set its value to 1



The screenshot shows a Jupyter Notebook cell with the following code:

```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
```

The variable `celsius` is highlighted with a red box and an arrow points from the text "declare a variable named celsius" to it. The output of the cell is:

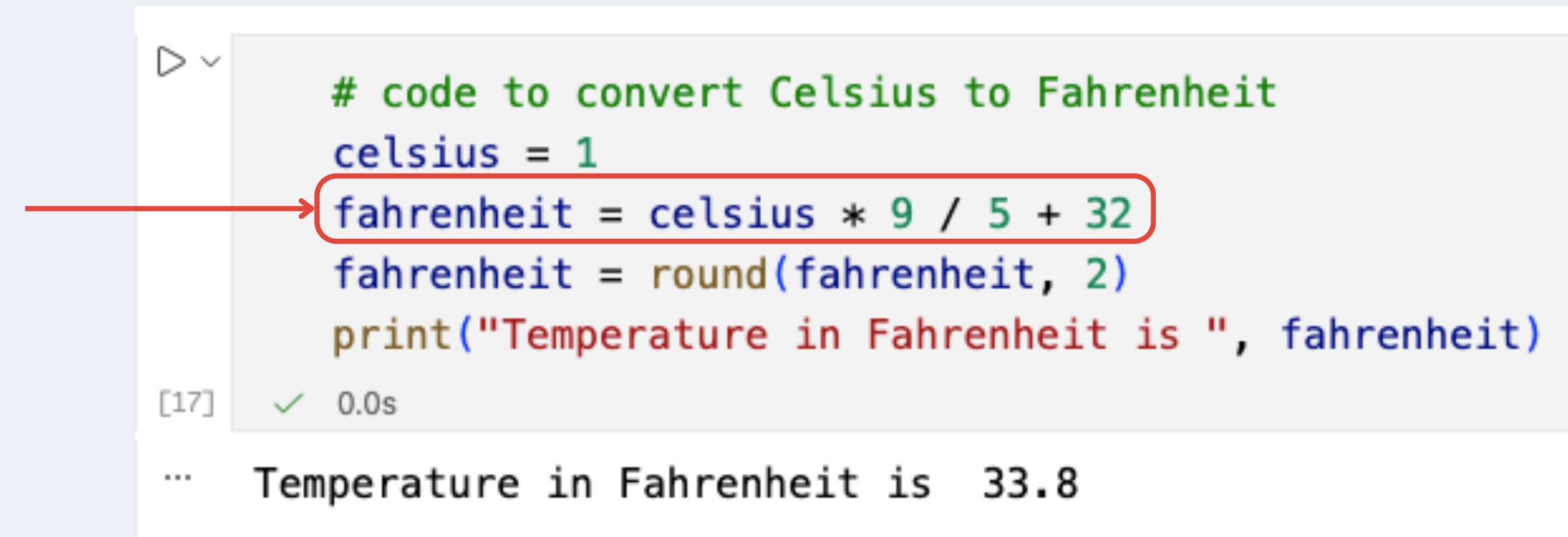
[17] ✓ 0.0s

... Temperature in Fahrenheit is 33.8

# WORKING ON FORMULAS

evaluate the expression on  
the left-side of symbol =

then assign result to  
fahrenheit



The screenshot shows a Jupyter Notebook cell with the following code:

```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
```

The line `fahrenheit = celsius * 9 / 5 + 32` is highlighted with a red box and a red arrow points from the text "evaluate the expression on the left-side of symbol =" to this line.

[17] ✓ 0.0s

... Temperature in Fahrenheit is 33.8

# WORKING ON FORMULAS

round off  
fahrenheit  
to two decimal digits

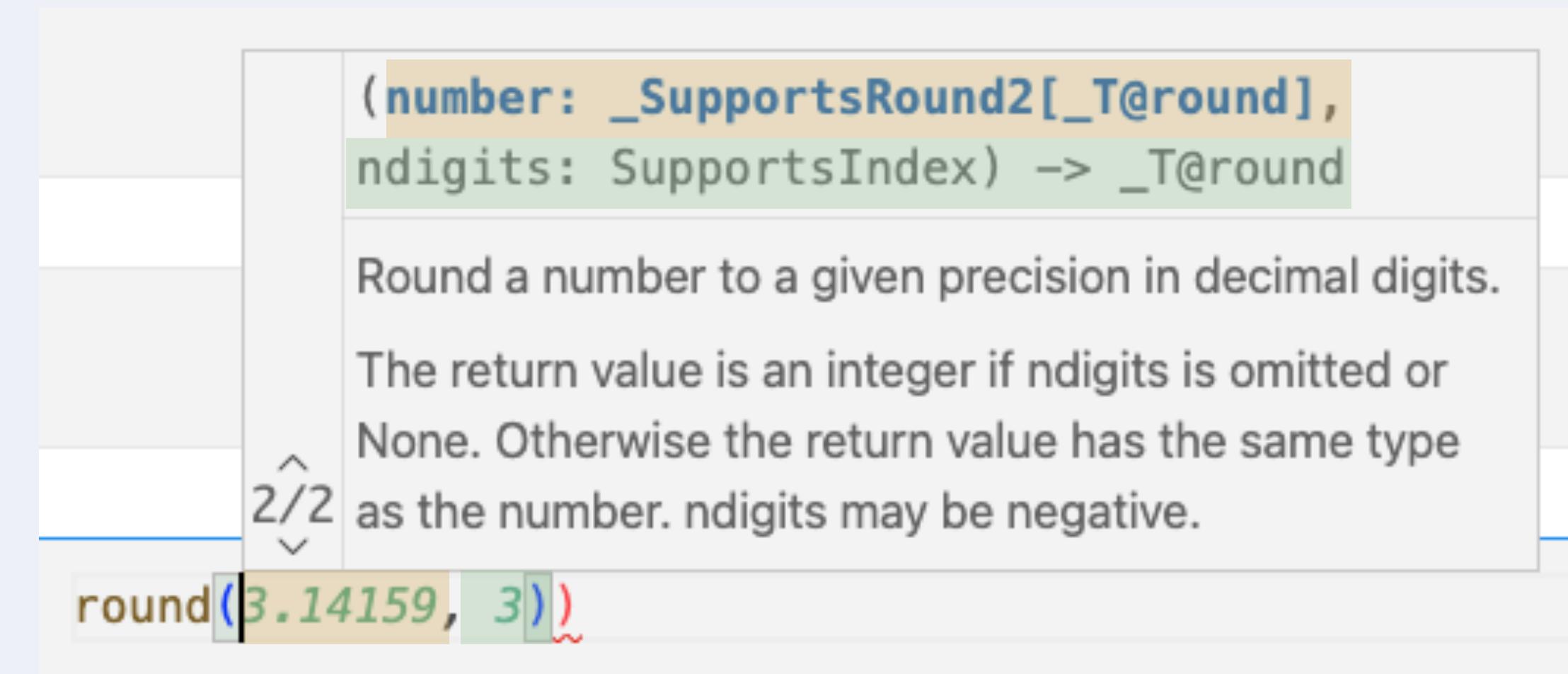
then assign result to the  
same variable

```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)

[17] ✓ 0.0s
...
Temperature in Fahrenheit is 33.8
```

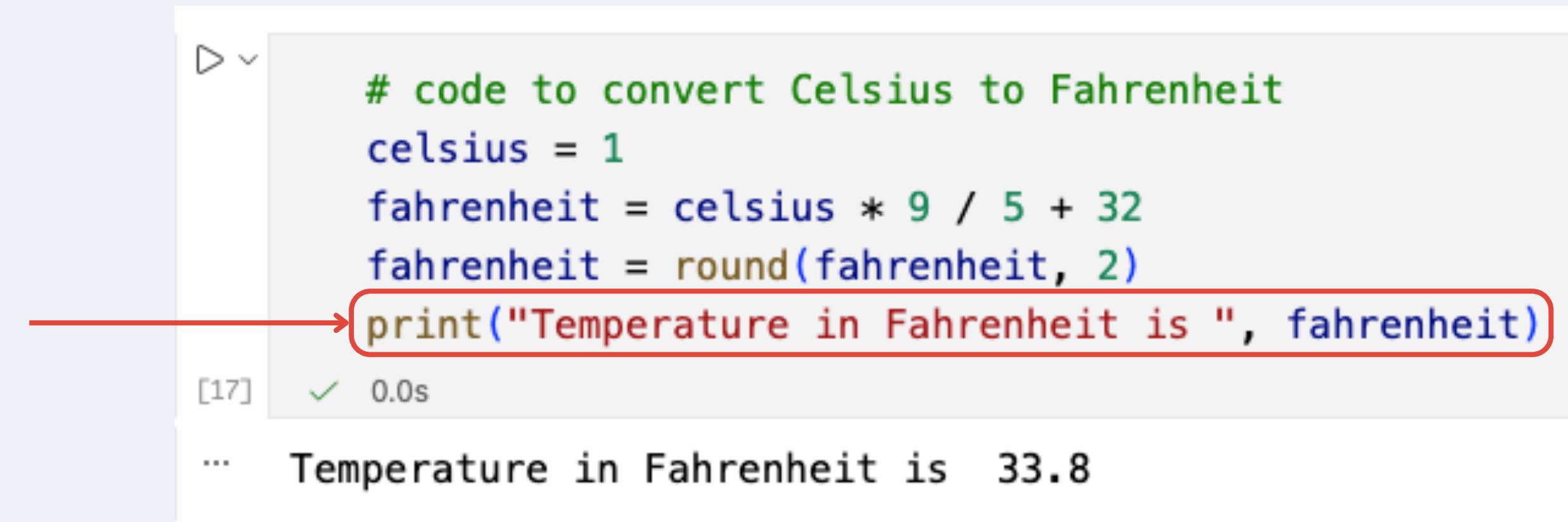
# WORKING ON FORMULAS

If you want to know more info about a certain command, you can use the auto-complete feature of VS Code



# WORKING ON FORMULAS

print some text followed by  
the value of  
fahrenheit



The screenshot shows a code editor window with a script for temperature conversion. A red arrow points from the explanatory text above to the `print` statement in the code.

```
# code to convert Celsius to Fahrenheit
celsius = 1
fahrenheit = celsius * 9 / 5 + 32
fahrenheit = round(fahrenheit, 2)
print("Temperature in Fahrenheit is ", fahrenheit)
```

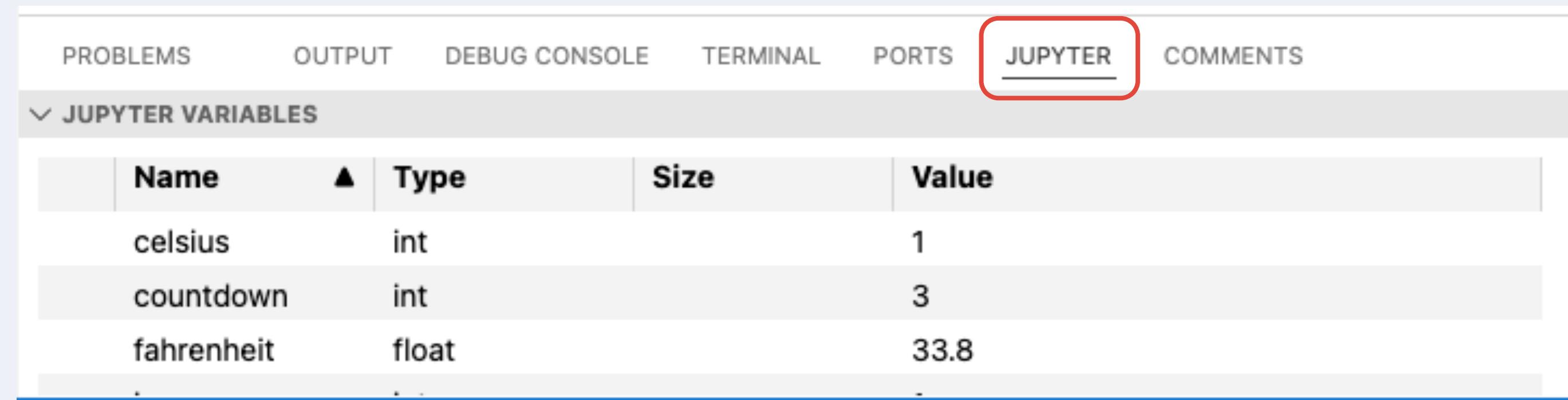
[17] ✓ 0.0s

... Temperature in Fahrenheit is 33.8

# WORKING ON FORMULAS

## More on Variables

Under Panel, there is a tab called JUPYTER that keeps track of the variables declared in your notebook



The screenshot shows the VS Code interface with the 'Panel' view open. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, JUPYTER (which is highlighted with a red box), and COMMENTS. Below the tabs, a section titled 'JUPYTER VARIABLES' is expanded. A table lists three variables: 'celsius' (int, size 1), 'countdown' (int, size 3), and 'fahrenheit' (float, size 33.8). The table has columns for Name, Type, Size, and Value.

	Name	Type	Size	Value
	celsius	int	1	
▼	countdown	int	3	
	fahrenheit	float	33.8	

# WORKING ON FORMULAS

## More on Variables

Under Panel, there is a tab called JUPYTER that keeps track of the variables declared in your notebook

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS	JUPYTER	COMMENTS
▼ JUPYTER VARIABLES						
	Name	Type	Size	Value		
	celsius	int	1			
	countdown	int	3			
	fahrenheit	float	33.8			

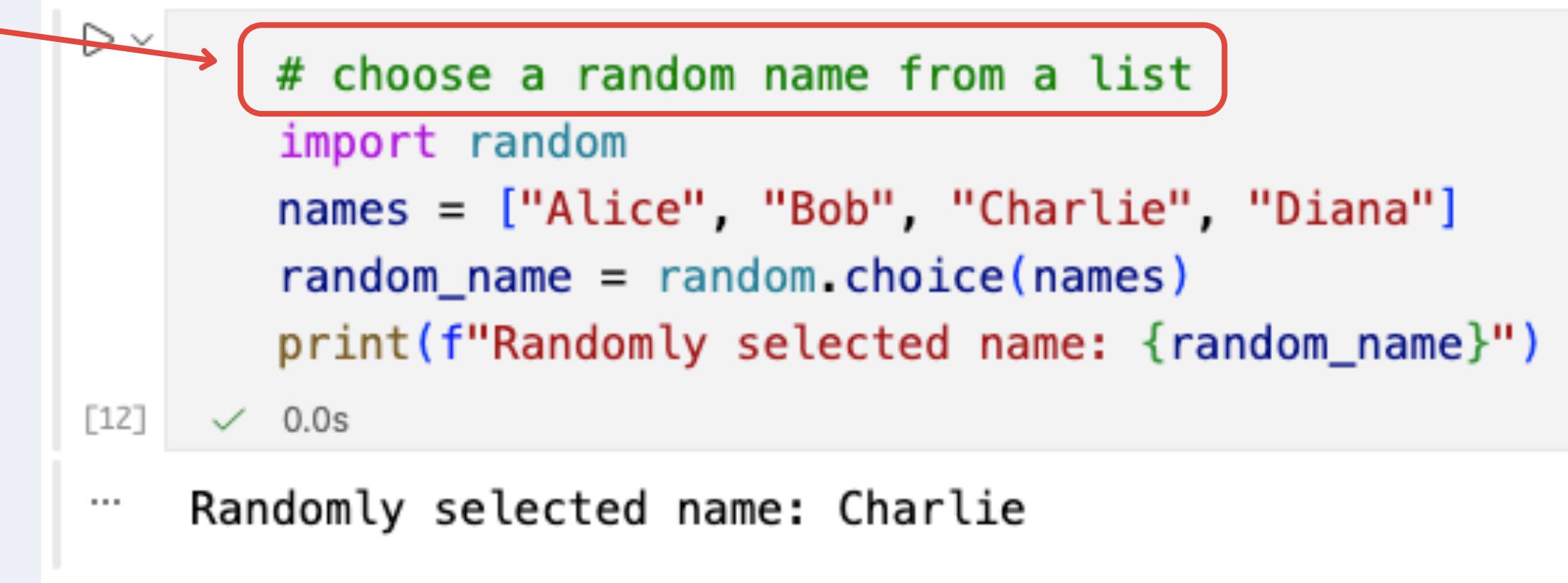
indicates the data type of the variable  
int means integer  
float means a decimal number

# CHOOSING A RANDOM NAME

```
▶ ▾ # choose a random name from a list
    import random
    names = ["Alice", "Bob", "Charlie", "Diana"]
    random_name = random.choice(names)
    print(f"Randomly selected name: {random_name}")
[12] ✓ 0.0s
... Randomly selected name: Charlie
```

# CHOOSING A RANDOM NAME

comment, no effect in code

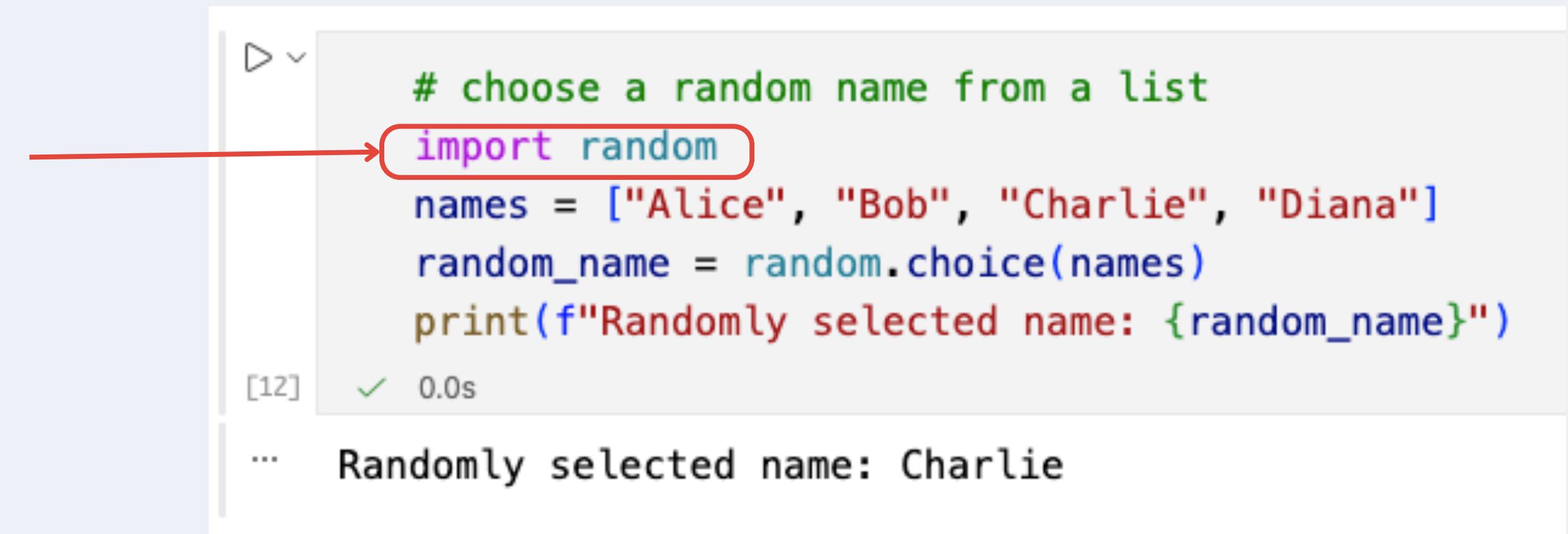


```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")

[12]    ✓  0.0s
... Randomly selected name: Charlie
```

# CHOOSING A RANDOM NAME

Calling the module  
random



The image shows a Jupyter Notebook cell with the following code:

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

The code imports the `random` module and defines a list of names. It then uses `random.choice` to select a name from the list and prints it. The output shows the program has run successfully and printed the name "Charlie".

[12] ✓ 0.0s

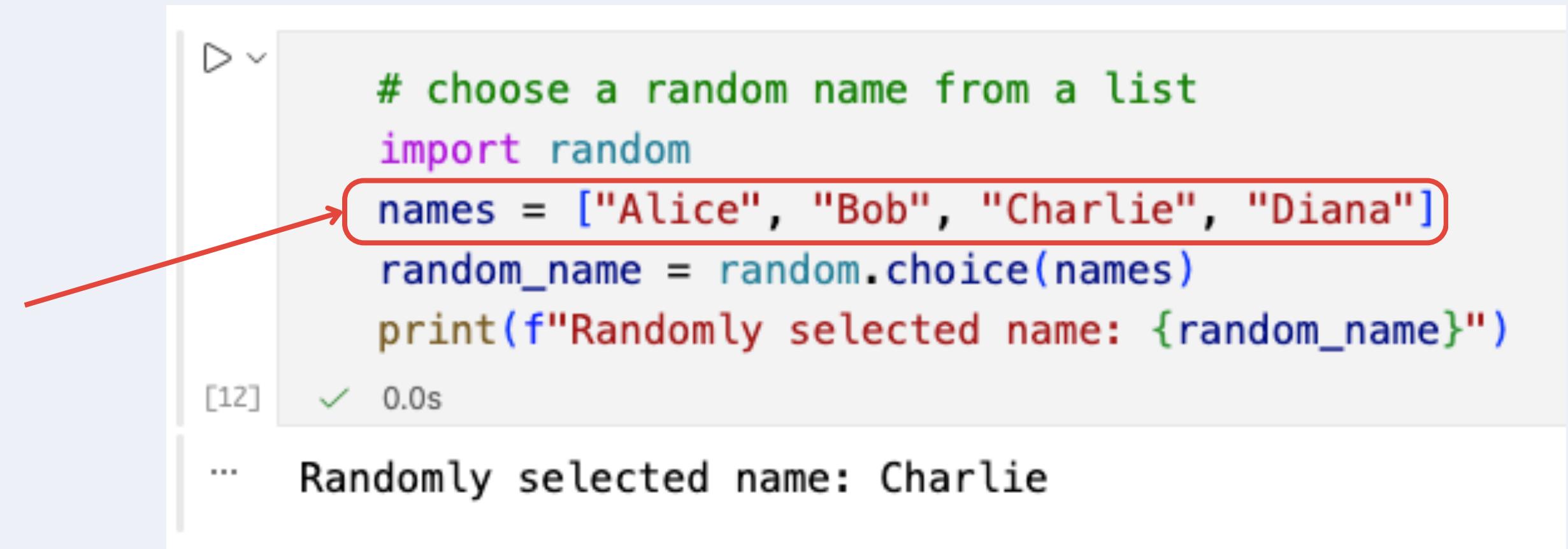
... Randomly selected name: Charlie

# CHOOSING A RANDOM NAME

- What is a module?
  - an external python file that we can import to our current python file
  - provides access to a collection of other commands (so-called functions)

# CHOOSING A RANDOM NAME

The variable `names` is used to remember a collection of values (in particular, strings) represented as a so-called **list**



A screenshot of a Jupyter Notebook cell. The cell contains the following Python code:

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

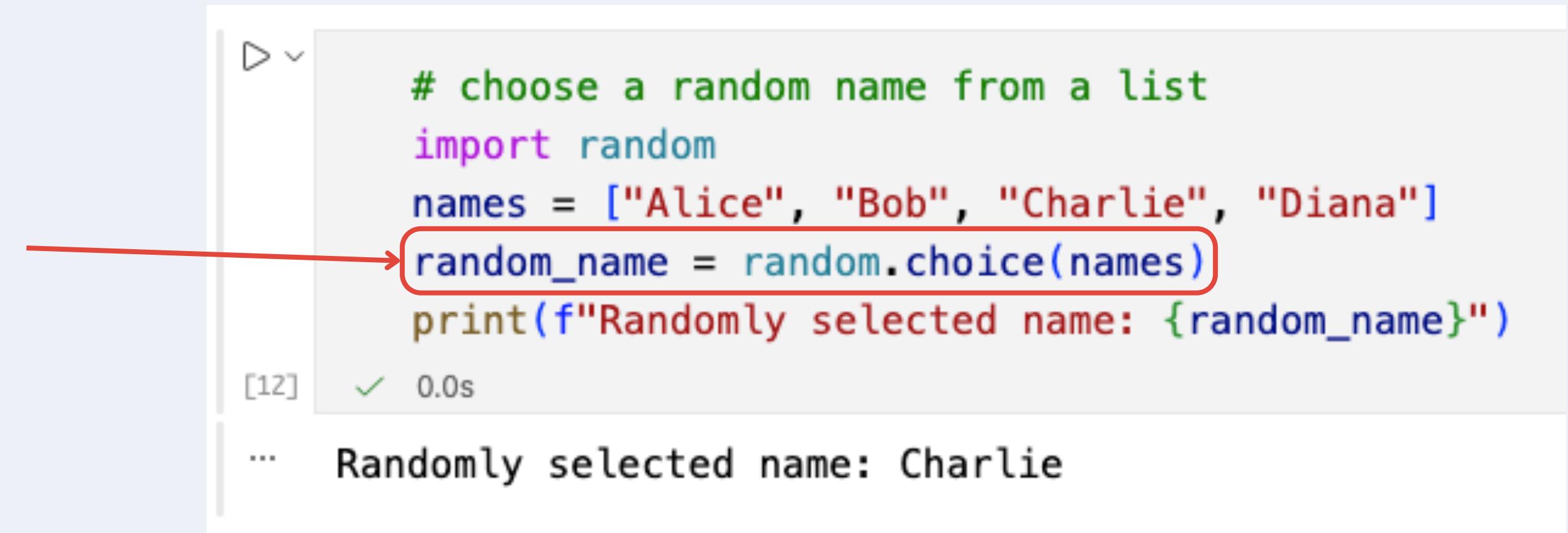
The line `names = ["Alice", "Bob", "Charlie", "Diana"]` is highlighted with a red rounded rectangle and has a red arrow pointing to it from the text above. The output of the cell is:

[12] ✓ 0.0s  
... Randomly selected name: Charlie

# CHOOSING A RANDOM NAME

left side of symbol =  
performs the function  
choice within module  
random

the chosen string is  
assigned to  
random\_name



The screenshot shows a Jupyter Notebook cell with the following code:

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

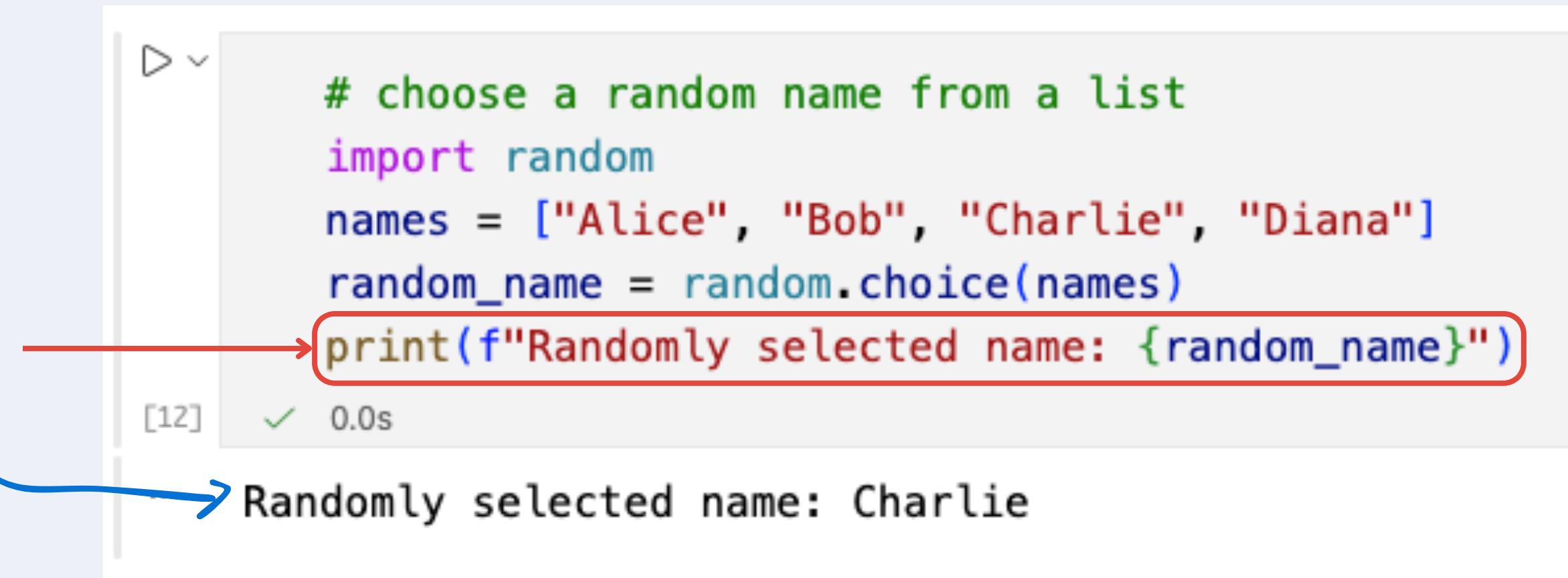
The line `random_name = random.choice(names)` is highlighted with a red box and an arrow points to it from the explanatory text above. The cell output shows the result:

[12] ✓ 0.0s

... Randomly selected name: Charlie

# CHOOSING A RANDOM NAME

prints a string and the chosen name



The screenshot shows a code editor window with a Python script. The script imports the `random` module and defines a list of names: `Alice`, `Bob`, `Charlie`, and `Diana`. It then uses `random.choice` to select a random name from the list and prints it using an f-string. A red box highlights the `print` statement. A blue arrow points from the explanatory text on the left to this highlighted line. Another blue arrow points from the highlighted line to the output console below, which displays the result: `Randomly selected name: Charlie`.

```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

[12] ✓ 0.0s

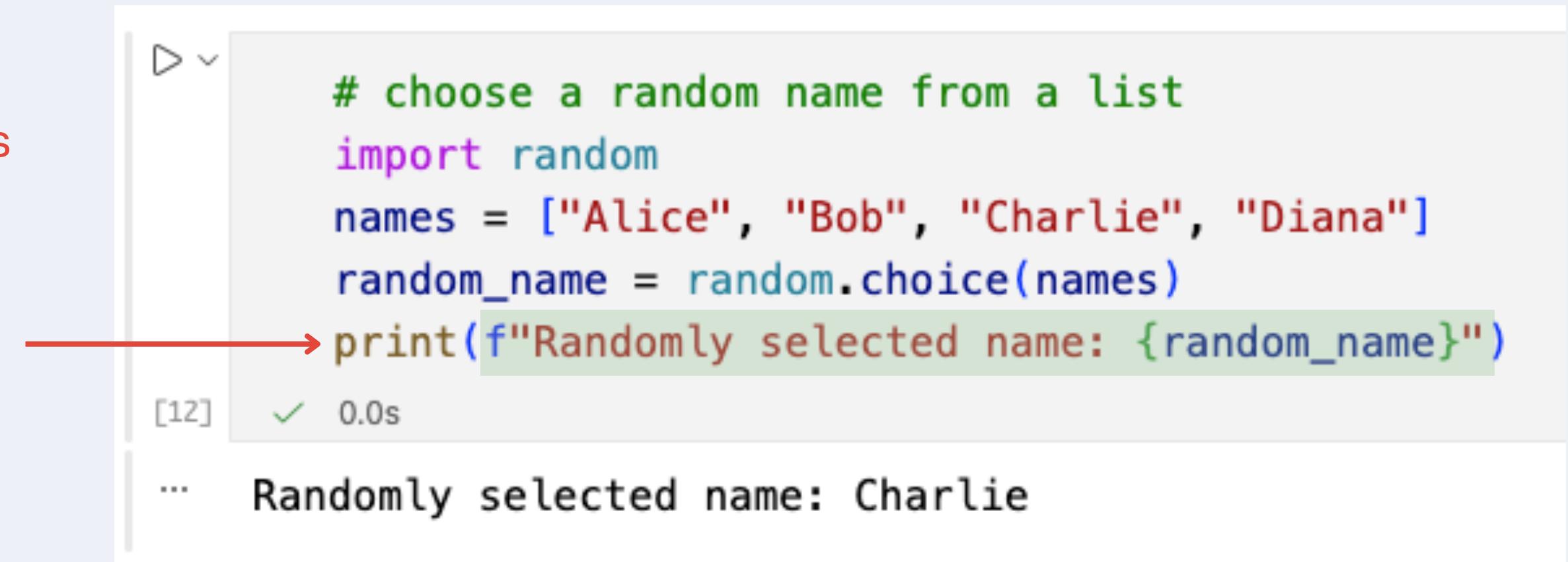
Randomly selected name: Charlie

# CHOOSING A RANDOM NAME

this is called a formatted-string  
(as marked by the f that precedes  
the string)

with formatted string, we can  
include variables inside the string

the variables are enclosed inside  
curly braces



```
# choose a random name from a list
import random
names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)
print(f"Randomly selected name: {random_name}")
```

[12] ✓ 0.0s

... Randomly selected name: Charlie

A screenshot of a Jupyter Notebook cell. The cell contains Python code: a comment, an import statement for 'random', a list of names ('names'), a variable assignment for 'random\_name' using 'random.choice', and a print statement using an f-string to output the randomly selected name. A red arrow points from the text 'with formatted string, we can include variables inside the string' to the f-string in the code. Another red arrow points from the text 'the variables are enclosed inside curly braces' to the curly braces in the f-string.

# CHOOSING A RANDOM NAME

```
# choose a random name from a list
import time
import random

names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)

countdown = 5
# countdown timer before printing the random name
for remaining in range(countdown, 0, -1):
    print(f"Printing random name in {remaining}...")
    time.sleep(1)

print(f"Randomly selected name: {random_name}")
[3] ✓ 5.0s
...
... Printing random name in 5...
Printing random name in 4...
Printing random name in 3...
Printing random name in 2...
Printing random name in 1...
Randomly selected name: Diana
```

# CHOOSING A RANDOM NAME

We already have some idea what the other lines do, except this part

```
# choose a random name from a list
import time
import random

names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)

countdown = 5
# countdown timer before printing the random name
for remaining in range(countdown, 0, -1):
    print(f"Printing random name in {remaining}...")
    time.sleep(1)

print(f"Randomly selected name: {random_name}")

[3] ✓ 5.0s
...
... Printing random name in 5...
Printing random name in 4...
Printing random name in 3...
Printing random name in 2...
Printing random name in 1...
Randomly selected name: Diana
```

# CHOOSING A RANDOM NAME

Let's use Copilot to help us understand this code chunk

```
# choose a random name from a list
import time
import random

names = ["Alice", "Bob", "Charlie", "Diana"]
random_name = random.choice(names)

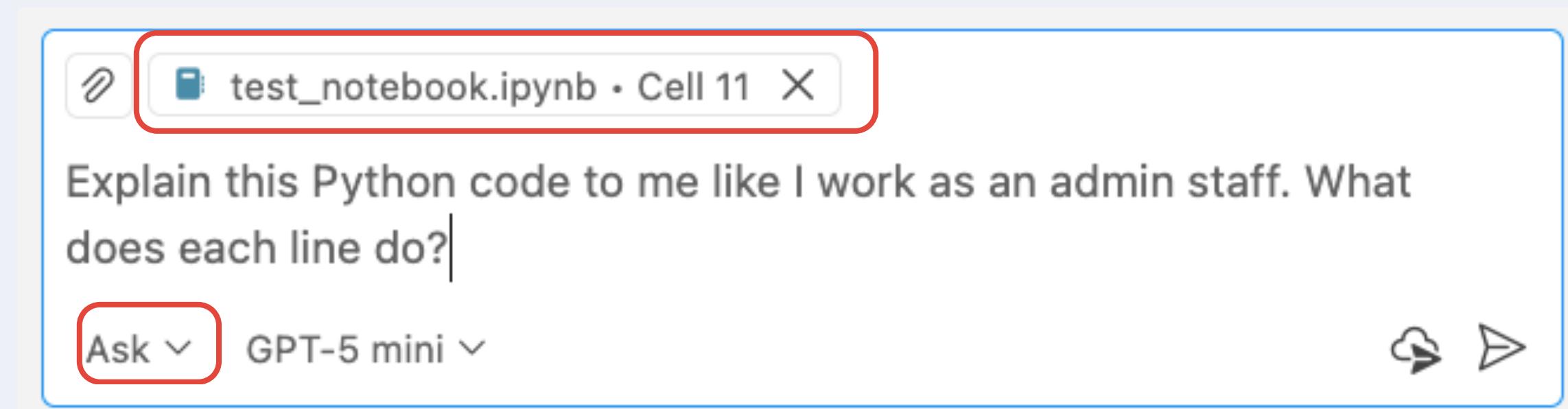
countdown = 5
# countdown timer before printing the random name
for remaining in range(countdown, 0, -1):
    print(f"Printing random name in {remaining}...")
    time.sleep(1)

print(f"Randomly selected name: {random_name}")

[3] ✓ 5.0s
...
... Printing random name in 5...
... Printing random name in 4...
... Printing random name in 3...
... Printing random name in 2...
... Printing random name in 1...
Randomly selected name: Diana
```

# CHOOSING A RANDOM NAME

1. Set Chat to be in Ask mode.
2. Click to the target code cell.
3. Ask away.



# CHOOSING A RANDOM NAME

1. Set Chat to be in Ask mode.
2. Click to the target code cell.
3. Ask away.

- `countdown = 5`
  - Sets a variable `countdown` to 5 — this is the number of seconds to wait.

- **countdown timer before printing the random name**

- Another comment explaining the next block.

- `for remaining in range(countdown, 0, -1):`
  - Starts a loop that sets `remaining` to 5, 4, 3, 2, 1 in turn.  
`range(start, stop, step)` goes down by 1 until it reaches 1 (stop is exclusive).

```
print(f"Printing random name in {remaining}...")
```

- - Inside the loop: prints a short message that tells you how many seconds are left.

```
time.sleep(1)
```

- - Inside the loop: pauses the program for 1 second, so the printed countdown waits in real time.
- (blank line)
  - Readability only.

# TYPES OF ERRORS IN PYTHON

- Syntax Errors
  - indicate that Python didn't understand the instruction or code provided
- Runtime Errors
  - indicate that Python understand the instruction or code, however, upon execution, a problem was encountered
- Semantic Errors
  - these are errors that Python cannot detect; the code may run smoothly, but the intended result is not reflected

```
import random

# generate a random integer between 1 and 100
random.randint(1, 100) = random_integer
random_integer = random.randint(1, 100)
print(f"Randomly generated integer: {random_integer}")

[5] 0.0s
```

Cell In[5], line 4  
random.randint(1, 100) = random\_integer  
^  
SyntaxError: cannot assign to function call here. Maybe you meant '==' instead of '='

[ ]

Keep Undo ⌘ | 1 of 1 ↑ ↓

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER Filter (e.g. text, !exclude, \escape...) Q ═ | ☰ X

how do I fix my code  
test\_notebook.ipynb • Cell 3:1-5

Step-by-step fix:

1. Correct the invalid assignment (swap sides) to set random\_integer = random.randint(1, 100).
2. Ensure print is spelled correctly and prints the integer (no division).
3. Move each cell's top-level id into metadata.id so each existing cell has metadata.id.

test\_notebook.ipynb  
Fix invalid assignment and add metadata.id to existing cells.  
test\_notebook.ipynb +1 -1  
C ↻ ⌛ ⌚ ... GPT-5 mini • 0x

Keep Undo ⌘ | +0 -0

1 file changed +0 -0  
test\_notebook.ipynb  
Edit or refactor selected code  
Edit ▾ GPT-5 mini ▾



```
# get all even numbers from 1 to 50 and print each in word form
even_numbers = [num for num in range(1, 51) if num % 2 == 0]
print(f"Even numbers from 1 to 50: {even_numbers}")
```

[7]

✓ 0.0s

Python

```
... Even numbers from 1 to 50: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32]
```

```
# get all even numbers from 1 to 50 and print each in word form
even_numbers = [num for num in range(1, 51) if num % 2 == 0]
print(f"Even numbers from 1 to 50: {even_numbers}")

[7] ✓ 0.0s
```

Even numbers from 1 to 50: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32]

```
# get all even numbers from 1 to 50
# get all even numbers from 1 to 50 and print each in word form
def num_to_words(n):
    units = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine',
             'eleven', 'twelve', 'thirteen', 'fourteen', 'fifteen', 'sixteen', 'seventeen']
    tens = ['', '', 'twenty', 'thirty', 'forty', 'fifty']
    if n < 20:
        return units[n]
    else:
        t = n // 10
        u = n % 10
        return tens[t] + ('' if u == 0 else ' ' + units[u])

even_numbers = [num for num in range(1, 51) if num % 2 == 0]
print(f"Even numbers from 1 to 50: {even_numbers}")

for n in even_numbers:
    print(f"{n}: {num_to_words(n)}")
```

[ ] ✓ 0.0s



modify the code to print each number in word form

test\_notebook.ipynb • Cell 4:1-4

Step-by-step solution

1. Update cell 4 (the code that builds even\_numbers) to convert each number to words and print one per line. Add a small num\_to\_words helper that covers 1-50.
2. Ensure every existing cell has metadata.id (move the current top-level id into metadata.id).
3. Save the notebook.

test\_notebook.ipynb

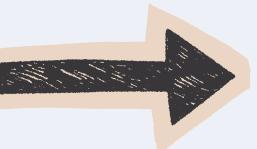
Change cell 4 to print each even number in word form and add metadata.id to existing cells.

test\_notebook.ipynb +1 -1

C ↪ ⌂ ⌂ ⌂ ...

# SOME NOTES IN PROMPTING

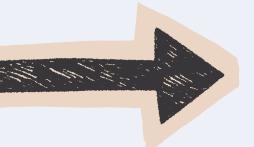
- Be specific
- Provide examples
- Ask for changes iteratively
- Ask for verification



Write a simple Python script that reads an Excel file called employee\_hours.xlsx, adds up all hours in the 'Hours Worked' column, and prints the total.

# SOME NOTES IN PROMPTING

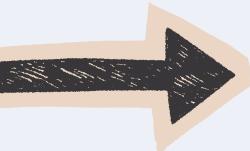
- Be specific
- Provide examples
- Ask for changes iteratively
- Ask for verification



"Here are two examples of input/output...  
Now write a function that..."

# SOME NOTES IN PROMPTING

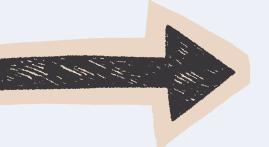
- Be specific
- Provide examples
- Ask for changes iteratively
- Ask for verification



- “add error handling”
- “turn this into a reusable function”
- add a check that warns me if the file is empty?

# SOME NOTES IN PROMPTING

- Be specific
- Provide examples
- Ask for changes iteratively
- Ask for verification



- “List possible bugs or edge cases in this code.”
- “Suggest improvements.”

# SUMMARY (SO FAR)

We learned about:

- some Python concepts commonly found in codes
- how to use Copilot to understand and fix codes

# EXERCISES:

1. Create a “Hello, World!” program
2. Ask Copilot to greet all office members
3. Ask Copilot to generate code for “printing today’s date”
4. Ask Copilot to generate code for savings greetings to a file.

Download Guided Worksheet 01 at <https://github.com/rbjuayong-ship-it/dxlabs/tree/main>

**10-min BREAK**

# **WORKING WITH DATA USING COPILOT**

**OBJECTIVE:** Learn some way to load and analyze data through codes generated by Copilot

# A SAMPLE CSV FILE

	A	B	C	D
1	name	department	training_completed	
2	Alice Johnson	Human Resources	Yes	
3	Brian Smith	Information Technology	No	
4	Carla Martinez	Finance	Yes	
5	David Lee	Marketing	Yes	
6	Emily Chen	Operations	No	
7	Frank Turner	Information Technology	Yes	
8	Grace Patel	Finance	Yes	
9	Henry Adams	Human Resources	No	
10	Isabella Cruz	Marketing	Yes	
11	Jason Wright	Operations	Yes	
12	Karen Brooks	Finance	No	
13	Liam O'Connor	Information Technology	Yes	
14	Mia Thompson	Human Resources	Yes	
15	Noah Davis	Marketing	No	
16	Olivia Lewis	Operations	Yes	
17	Paul Robinson	Information Technology	No	
18	Quinn Foster	Finance	Yes	
19	Riley Morgan	Human Resources	Yes	
20	Sophia Bennett	Marketing	Yes	
21	Thomas Walker	Operations	No	

```
name,department,training_completed
Alice Johnson,Human Resources,Yes
Brian Smith,Information Technology,No
Carla Martinez,Finance,Yes
David Lee,Marketing,Yes
Emily Chen,Operations,No
Frank Turner,Information Technology,Yes
Grace Patel,Finance,Yes
Henry Adams,Human Resources,No
Isabella Cruz,Marketing,Yes
Jason Wright,Operations,Yes
Karen Brooks,Finance,No
Liam O'Connor,Information Technology,Yes
Mia Thompson,Human Resources,Yes
Noah Davis,Marketing,No
Olivia Lewis,Operations,Yes
Paul Robinson,Information Technology,No
Quinn Foster,Finance,Yes
Riley Morgan,Human Resources,Yes
Sophia Bennett,Marketing,Yes
Thomas Walker,Operations,No
```

# SAMPLE CODE TO LOAD CSV

Copilot suggestion

```
▶ ▾ # load "training.csv" data in and print each row
    import csv
    with open('training.csv', mode='r') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            print(row)

[5] ✓ 0.0s Python
...
['name', 'department', 'training_completed']
['Alice Johnson', 'Human Resources', 'Yes']
['Brian Smith', 'Information Technology', 'No']
['Carla Martinez', 'Finance', 'Yes']
['David Lee', 'Marketing', 'Yes']
['Emily Chen', 'Operations', 'No']
```

# SAMPLE CODE TO LOAD CSV

```
# load "training.csv" data in and print each row
import csv

path = 'training.csv'
with open(path, mode='r', encoding='utf-8-sig', newline='') as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row)

[13] ✓ 0.0s
{"name": "Alice Johnson", "department": "Human Resources", "training_completed": "Yes"}, {"name": "Brian Smith", "department": "Information Technology", "training_completed": "No"}, {"name": "Carla Martinez", "department": "Finance", "training_completed": "Yes"}, {"name": "David Lee", "department": "Marketing", "training_completed": "Yes"}, {"name": "Emily Chen", "department": "Operations", "training_completed": "No"}, {"name": "Frank Turner", "department": "Information Technology", "training_completed": "Yes"}, {"name": "Grace Patel", "department": "Finance", "training_completed": "Yes"}, {"name": "Henry Adams", "department": "Human Resources", "training_completed": "No"}, {"name": "Isabella Cruz", "department": "Marketing", "training_completed": "Yes"}, {"name": "Jason Wright", "department": "Operations", "training_completed": "Yes"}, {"name": "Karen Brooks", "department": "Finance", "training_completed": "No"}, {"name": "Liam O'Connor", "department": "Information Technology", "training_completed": "Yes"}
```

another Copilot  
suggestion

# SAMPLE CODE TO PROCESS CSV

Copilot suggestion

but is slightly wrong

```
# load "training.csv" data in and count names who completed the training
import csv
completed_count = 0
with open('training.csv', mode='r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        if row[1].lower() == 'yes':
            completed_count += 1
print(f"Number of people who completed the training: {completed_count}")
```

✓ 0.0s

Python

Number of people who completed the training: 0

# SAMPLE CODE TO PROCESS CSV

corrected

Copilot suggestion

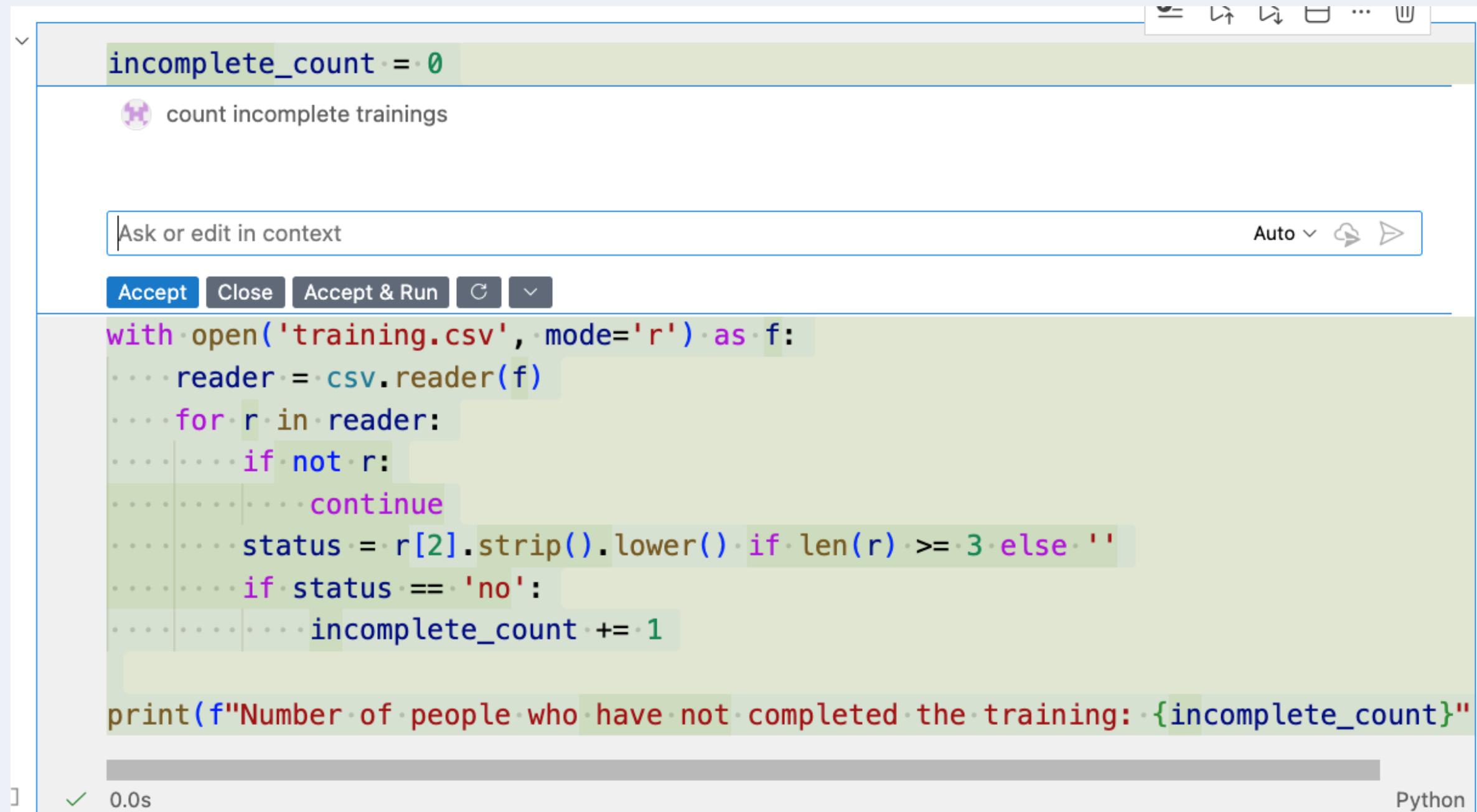
```
# load "training.csv" data in and count names who completed the training
import csv
completed_count = 0
with open('training.csv', mode='r') as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        if row[2].lower() == 'yes':
            completed_count += 1
print(f"Number of people who completed the training: {completed_count}")
```

✓ 0.0s

Python

Number of people who completed the training: 13

# SAMPLE CODE TO PROCESS CSV



The screenshot shows a code editor window with the following Python code:

```
incomplete_count = 0
# count incomplete trainings
Ask or edit in context
Auto ▾ ⌂ ⌂
Accept Close Accept & Run ⌂ ▾
with open('training.csv', mode='r') as f:
    reader = csv.reader(f)
    for r in reader:
        if not r:
            continue
        status = r[2].strip().lower() if len(r) >= 3 else ''
        if status == 'no':
            incomplete_count += 1
print(f"Number of people who have not completed the training: {incomplete_count}")

```

The code counts the number of incomplete trainings from a CSV file named "training.csv". It uses the `csv.reader` function to read the file and an `if not r` condition to skip empty rows. It then checks the third column (`r[2]`) for the string "no" to determine if the training was incomplete. The code is annotated with inline comments: "# count incomplete trainings" and "Ask or edit in context". The code editor interface includes buttons for "Accept", "Close", "Accept & Run", and "Auto" dropdown.

Using inline chat  
to modify the code

# SAMPLE CODE TO PROCESS CSV

```
incomplete_count = 0
with open('training.csv', mode='r') as f:
    reader = csv.reader(f)
    for r in reader:
        if not r:
            continue
        status = r[2].strip().lower() if len(r) >= 3 else ''
        if status == 'no':
            incomplete_count += 1

print(f"Number of people who have not completed the training: {incomplete_count}")
```

[7] ✓ 0.0s

Python

... Number of people who have not completed the training: 7

Using inline chat  
to modify the code

# TRY IT YOURSELF!

Ask AI to modify the code in order to:

1. Show names of people needing training
2. Group by department
3. Change the names to uppercase and print the result in a new file

# TRY IT YOURSELF!

```
# group by department and count completed vs incomplete (no/other)
dept_counts = {}

with open('training.csv', 'r', encoding='utf-8') as f:
    reader = csv.reader(f)
    for i, row in enumerate(reader):
        if not row:
            continue
        # skip header if present
        if i == 0 and any(h.strip().lower() in ('name', 'department', 'status') for h in row):
            continue

        dept = row[1].strip() if len(row) > 1 and row[1].strip() else 'Unknown'
        status = row[2].strip().lower() if len(row) > 2 else ''

        completed = status in ('yes', 'y', 'completed', 'complete', 'true', '1')
        counts = dept_counts.setdefault(dept, {'completed': 0, 'incomplete': 0, 'total': 0})

        if completed:
            counts['completed'] += 1
        else:
            counts['incomplete'] += 1
        counts['total'] += 1

# print results sorted by department name
for dept in sorted(dept_counts):
    c = dept_counts[dept]
    print(f'{dept}: {c["completed"]} completed, {c["incomplete"]} incomplete (total {c["total"]})')

[8] ✓ 0.0s
...
Finance: 3 completed, 1 incomplete (total 4)
Human Resources: 3 completed, 1 incomplete (total 4)
Information Technology: 2 completed, 2 incomplete (total 4)
Marketing: 3 completed, 1 incomplete (total 4)
Operations: 2 completed, 2 incomplete (total 4)
```

# TRY IT YOURSELF!

```
# load "training.csv" data in and modify the names to uppercase, then print each row to a new file "training_uppercase.csv"
import csv
with open('training.csv', mode='r', newline='', encoding='utf-8') as infile, \
    open('training_uppercase.csv', mode='w', newline='', encoding='utf-8') as outfile:
    csv_reader = csv.reader(infile)
    csv_writer = csv.writer(outfile)

    for row in csv_reader:
        if row: # ensure the row is not empty
            row[0] = row[0].upper() # convert the name (first column) to uppercase
            csv_writer.writerow(row)
```

# TRY IT YOURSELF!

training\_uppercase.csv

	NAME	department	training_completed
1	ALICE JOHNSON	Human Resources	Yes
2	BRIAN SMITH	Information Technology	No
3	CARLA MARTINEZ	Finance	Yes
4	DAVID LEE	Marketing	Yes
5	EMILY CHEN	Operations	No
6	FRANK TURNER	Information Technology	Yes
7	GRACE PATEL	Finance	Yes
8	HENRY ADAMS	Human Resources	No
9	ISABELLA CRUZ	Marketing	Yes
10	JASON WRIGHT	Operations	Yes
11	KAREN BROOKS	Finance	No
12	LIAM O'CONNOR	Information Technology	Yes
13	MIA THOMPSON	Human Resources	Yes
14	NOAH DAVIS	Marketing	No
15	OLIVIA LEWIS	Operations	Yes
16	PAUL ROBINSON	Information Technology	No
17	QUINN FOSTER	Finance	Yes
18	RILEY MORGAN	Human Resources	Yes
19	SOPHIA BENNETT	Marketing	Yes
20	THOMAS WALKER	Operations	No

# CULMINATION ACTIVITY

## CO-PILOT YOUR OWN CSV FILE

Use co-pilot in VSCode to generate code that will read a .csv file of your own dataset and automate a **simple task** on that file.

# **CULMINATION ACTIVITY**

## **CO-PILOT YOUR OWN CSV FILE**

1. Share to the group a short description of the dataset.
2. Talk about the goal or task you want to perform on your dataset
3. Share the result of co-pilot and describe your insights regarding the generated code. How helpful was it? Was it able to do correctly what you want it to be.

# **WRAP-UP, REFLECTION, AND PREVIEW**

**OBJECTIVE:** Reinforce concepts and connect to future sessions

# RECAP

We learned and experienced:

1. Basic navigation on Visual Studio Code
2. How to work on Jupyter notebooks
3. How to use Github Copilot
4. Study some simple Python codes
5. How to use Copilot to understand codes
6. How to use Copilot to process some data

# FURTHER EXPLORATION

1. Try editing Copilot's code suggestions
2. Explore VS Code keyboard shortcuts
3. Optional: “Push your Week 1 Notebook” to Github

# FROM WEEK 01 TO WEEK 02

No problem generating code,  
but how good are we at understanding,  
modifying and applying the codes?

# FROM WEEK 01 TO WEEK 02

No problem generating code,  
but how good are we at understanding,  
modifying and applying the codes?

A good overview of fundamental concepts will  
help us improve our code understanding.

# FROM WEEK 01 TO WEEK 02

Data Types and Operations

Variables and Assignments

Branching

Functions

Print Statements

Collection of Values

Iterations

OOP

Numpy

pandas

matplotlib

towards EDA

**END OF SLIDES**

