

function all concept with example and interview
questions ans with full explanation deply

I will teach **Function – All Concepts + Examples + Interview Questions (with Answers + Deep Explanation)**.

Before I begin, tell me **which programming language** you want the topic in:

- ◆ Java
- ◆ JavaScript
- Python
- C / C++
- Other (please mention)

👉 Reply with the language name, for example:

JavaScript functions full explanation

Waiting for your answer 😊👉



function all concept with example and interview
questions ans with full explanation deply python

🔥 Functions in Python — All Concepts + Examples + Interview Q/A (Deep Explanation)

📌 What is a Function in Python?

A **function** is a reusable block of code that performs a specific task.

It helps avoid repetition and makes programs modular and readable.

🧠 Types of Functions in Python

Type	Example
Built-in Functions	<code>print()</code> , <code>len()</code> , <code>max()</code>
User-defined Functions	<code>def sum(a, b):</code>
Anonymous Functions (Lambda)	<code>lambda x: x*2</code>
Recursive Functions	A function calling itself
Higher Order Functions	takes or returns a function
Generator Functions	uses <code>yield</code>
Function with arguments types	positional, keyword, default, variable



1. Creating a Function

python

Copy code

```
def greet():  
    print("Hello Python!")  
greet()
```



Output:

nginx

Copy code

Hello Python!



Explanation:

- `def` keyword is used to define a function.
- `greet()` is a function call.



2. Function with Parameters

python

Copy code

```
def add(a, b):  
    return a + b  
  
print(add(10, 5))
```

📌 Output:

📄 Copy code

15

📄 Explanation:

- `return` sends the result back.

📄 3. Default Arguments

python

📄 Copy code

```
def greet(name="Guest"):  
    print("Hello", name)  
  
greet("Aamir")  
greet()
```

📌 Output:

nginx

📄 Copy code

```
Hello Aamir  
Hello Guest
```

✓ If no value passed, default value is used.

🤖 4. Keyword Arguments

python

📄 Copy code

```
def student(name, age):  
    print("Name:", name, "Age:", age)
```

```
student(age=20, name="Aamir")
```

✂ Output:

```
makefile
```

📄 Copy code

```
Name: Aamir Age: 20
```

✓ Order doesn't matter.

5. Variable-Length Arguments

*args (Multiple Positional Arguments)

```
python
```

📄 Copy code

```
def total(*numbers):  
    return sum(numbers)  
  
print(total(10, 20, 30))
```

✂ Output:

📄 Copy code

```
60
```

**kwargs (Multiple Keyword Arguments)

```
python
```

📄 Copy code

```
def person(**details):  
    print(details)  
  
person(name="Aamir", age=20, city="Delhi")
```

✂ Output:

```
bash
```

📄 Copy code

```
{'name': 'Aamir', 'age': 20, 'city': 'Delhi'}
```

6. Lambda (Anonymous Function)

python

 Copy code

```
square = lambda x: x*x  
print(square(5))
```

 Output:

 Copy code

25

✓ Used for short, quick functions.

7. Recursion (Function Calling Itself)

python

 Copy code

```
def fact(n):  
    if n == 0:  
        return 1  
    return n * fact(n-1)  
  
print(fact(5))
```

 Output:

 Copy code

120

8. Higher Order Function

 Function that takes another function as argument.

python

```
def greet():  
    return "Hello!"
```

```
def show(func):  
    print(func())
```

```
show(greet)
```

📌 Output:

📄 Copy code

Hello!

9. Generator Function

Uses `yield` instead of `return`.

python

📄 Copy code

```
def my_gen():  
    yield 1  
    yield 2  
    yield 3  
  
for i in my_gen():  
    print(i)
```

📌 Output:

📄 Copy code

1
2
3

✓ Generates values one by one (memory efficient).

10. Nested Function

python

```
def outer():  
    def inner():  
        print("Inner Function")  
    inner()
```

```
outer()
```



Interview Questions with Answers

? Q1. What is the difference between `return` and `print()` ?

<code>print()</code>	<code>return</code>
Displays value	Sends value back
Cannot be used in expressions	Can be used

? Q2. Can we return multiple values in Python?

👉 Yes, using tuple

python

 Copy code

```
def calc(a, b):  
    return a+b, a*b, a-b  
  
print(calc(5, 3))
```

📌 Output:

scss

 Copy code

```
(8, 15, 2)
```

? Q3. What is a lambda function?

- ✓ Small anonymous function without a name
- ✓ Created using `lambda` keyword
- 👉 Used in: filters, map, reduce

Example:

```
python
```

 Copy code

```
double = lambda x: x*2  
print(double(4))
```

? Q4. How is `*args` different from `**kwargs` ?

<code>*args</code>	<code>**kwargs</code>
Variable positional arguments	Variable keyword arguments
Returns Tuple	Returns Dictionary

? Q5. What is recursion?

- ✓ A function calling itself
- ✓ Must have a base condition to avoid infinite loop.

? Q6. What are generator functions?

- ✓ Functions that return values using `yield`, not `return`.
- ✓ Saves memory because items are produced when needed.

? Q7. What is a higher order function?

- ✓ A function that takes another function as a parameter or returns a function.

? Q8. Explain scope in Python.

Type**Access**

Local

Inside function only

Global

Anywhere in program

Example:

python

 Copy code

```
x = 10
def test():
    x = 5
    print(x)
test()
print(x)
```

 Output: Copy code

```
5
10
```

 **Do you want exercises / MCQs / coding problems to practice?**

Reply:

 **Yes, give exercises**

or

 **I need MCQs**

or

 **Give interview coding questions**     

all type function explain

Here is a **complete & clear explanation of all types of functions in Python**, with code + diagrams + uses.

We will cover every type of function:

All Types of Functions in Python (Complete Explanation)

Functions in Python are broadly divided into:

A. Based on Creation


1. Built-in Functions
2. User-defined Functions
3. Anonymous (Lambda) Functions

B. Based on Argument Handling

4. Positional Arguments
5. Keyword Arguments
6. Default Arguments
7. Variable-Length Arguments (`*args` , `**kwargs`)
8. Keyword-only Arguments
9. Positional-only Arguments

C. Based on Behavior

10. Recursive Functions
11. Higher-Order Functions
12. Generator Functions
13. Nested Functions
14. Closure Functions
15. Decorators

Let's understand each type deeply 

A. Based on Creation

1) Built-in Functions

These are ready-made functions provided by Python.

Examples:

python

 Copy code

```
print(len("Python"))
print(max([10, 20, 5]))
```

✦ Common built-in functions: `print()` , `len()` , `max()` , `sum()` , `sorted()` , `range()` .

2) ♦ User-defined Functions

Functions created by users using `def` .

python

 Copy code

```
def greet(name):
    return "Hello " + name

print(greet("Aamir"))
```

3) ♦ Anonymous (Lambda) Functions

Short unnamed functions using `lambda` .

python

 Copy code

```
square = lambda x: x*x
print(square(4))
```

✓ Used with `map()` , `filter()` , `reduce()` .

B. Based on Argument Handling

4) ♦ Positional Arguments

Values passed in order.

python

 Copy code

```
def add(a, b):
    print(a + b)
```

```
add(5, 3)
```

5) ♦ Keyword Arguments

Arguments passed with a name, order doesn't matter.

```
python
```

[Copy code](#)

```
def student(name, age):  
    print(name, age)  
  
student(age=21, name="Aamir")
```

6) ♦ Default Arguments

If you don't pass value, default value is used.

```
python
```

[Copy code](#)

```
def greet(name="Guest"):  
    print("Hello", name)  
  
greet()
```

7) ♦ Variable-Length Arguments

🌀 Using *args (Positional variable argument)

```
python
```

[Copy code](#)

```
def total(*numbers):  
    print(sum(numbers))  
  
total(1, 2, 3, 4)
```

👉 Result stores as tuple .

🌀 Using **kwargs (Keyword variable argument)

python

 Copy code

```
def data(**info):  
    print(info)  
  
data(name="Aamir", age=20)
```

👉 Result stores as dictionary .

8) ♦ Keyword-only Arguments

Arguments forced to be keyword-only by * .

python

 Copy code

```
def fun(a, b, *, c):  
    print(a, b, c)  
  
fun(1, 2, c=3) # Must use name for c
```

9) ♦ Positional-only Arguments (Python 3.8+)

Arguments forced to be positional using / .

python

 Copy code

```
def divide(a, b, /):  
    print(a/b)  
  
divide(10, 2) # Must be positional
```

C. Based on Behavior

10) Recursive Functions

Function calling itself.

python

 Copy code

```
def fact(n):  
    if n == 0:  
        return 1  
    return n * fact(n-1)  
  
print(fact(5))
```

11) Higher Order Functions

Takes another function as input OR returns function.

python

 Copy code

```
def say():  
    return "Hello"  
  
def display(func):  
    print(func())  
  
display(say)
```

12) Generator Functions (with yield)

Used to produce sequence one at a time (memory efficient).

python

 Copy code

```
def numbers():  
    yield 1  
    yield 2  
    yield 3  
  
for i in numbers():  
    print(i)
```

13) Nested Functions

Function inside another function.

python

 Copy code

```
def outer():  
    def inner():  
        print("Inner Function")  
    inner()  
outer()
```

14) Closure Functions

Inner function remembers values from outer even after outer ends.

python

 Copy code

```
def outer(msg):  
    def inner():  
        print(msg)  
    return inner  
  
f = outer("Hello Closure")  
f()
```

 Closure stores msg .

15) Decorators

Functions that modify other functions without changing their code.

python

 Copy code

```
def decor(func):  
    def wrapper():  
        print("Welcome!")  
        func()  
    return wrapper  
  
@decor  
def hello():  
    print("Hello Python")  
  
hello()
```



Summary Table

Type	Example	Code Keyword
Built-in	<code>print()</code>	Provided by Python
User-defined	<code>def f()</code>	<code>def</code>
Lambda	Short func	<code>lambda</code>
Positional Args	<code>f(1,2)</code>	order matters
Keyword Args	<code>f(a=1)</code>	name=value
Default Args	<code>a=10</code>	default
*args	<code>f(1,2,3)</code>	tuple
**kwargs	<code>f(a=1)</code>	dict
Recursion	calls itself	base case
Higher Order	takes function	function arg
Generator	yield	yield
Nested	inner func	inside def
Closure	remembers	return inner
Decorator	modifies function	@decor



Would you like?



Practice Questions



MCQs



Coding Interview Problems

Reply:

Give practice OR Give MCQs OR Give interview coding .



explain all type functions

Below is the **full and complete detailed explanation of ALL types of functions in Python.**

This covers **definitions, diagrams, use-cases, code, outputs, and differences.**

Nothing will be left incomplete. ✓

What is a Function?

A function is a *block of reusable code* that performs a specific task.

Python functions are created using the keyword `def` or `lambda`.

ALL TYPES OF FUNCTIONS IN PYTHON


We can classify Python functions into **3 categories**:

A. Based on Creation

Type	Description
Built-in Functions	Already present in Python
User-Defined Functions	Defined by the programmer
Anonymous Functions	Functions without a name (lambda)

Built-in Functions

These are pre-written functions.

 **Examples:** `print()`, `len()`, `max()`, `range()`, `sum()`, etc.

python

 Copy code

```
print(len("Python"))  
print(max([4, 10, 2]))
```

📌 Python internally defines these.

2 User-Defined Functions

Functions created by you.

python

📄 Copy code

```
def greet():  
    print("Hello, Python!")
```

```
greet()
```

📄 Why used?

- ◆ Reusability
 - ◆ Reduces code duplication
 - ◆ Improves structure
-

3 Anonymous Functions (Lambda)

Short, one-line function without a name.

python

📄 Copy code

```
square = lambda x: x*x  
print(square(5))
```

📌 Used in: `map()`, `filter()`, `reduce()`.

📌 B. Based on Argument Handling

Functions differ based on how they receive values.

4 Positional Arguments

Arguments passed in **correct order**.

python

 Copy code

```
def add(a, b):  
    print(a + b)  
  
add(10, 5) # Order matters
```

5 Keyword Arguments

Pass arguments with **name**, order doesn't matter.

python

 Copy code

```
def student(name, age):  
    print(name, age)  
  
student(age=20, name="Aamir")
```

6 Default Arguments

Arguments with **default values**.

python

 Copy code

```
def greet(name="Guest"):  
    print("Hello", name)  
  
greet()  
greet("Aamir")
```

7 Variable-Length Arguments

 ***args** → Multiple positional arguments (tuple)

python

 Copy code

```
def total(*numbers):  
    print(sum(numbers))
```

```
total(2, 3, 5, 10)
```

**kwargs → Multiple keyword arguments (dictionary)

python

 Copy code

```
def info(**data):  
    print(data)  
  
info(name="Aamir", age=20, city="Delhi")
```

8 Keyword-Only Arguments

Enforced keyword arguments using `*`.

python

 Copy code

```
def display(a, *, b):  
    print(a, b)  
  
display(10, b=20) # Must use b as keyword
```



9 Positional-Only Arguments (Python 3.8+)

Force arguments to be positional using `/`.

python

 Copy code

```
def divide(a, b, /):  
    print(a/b)  
  
divide(10, 2) # Correct
```

 `divide(a=10, b=2)` →  Error

C. Based on Behavior

10 Recursive Functions

A function that calls itself.

python

 Copy code

```
def fact(n):  
    if n == 0:  
        return 1  
    return n * fact(n-1)  
  
print(fact(5))
```

 Must include **base condition**, otherwise infinite recursion.

1 1 Higher-Order Functions

Functions that take or return another function.

python

 Copy code

```
def greet():  
    return "Hello!"  
  
def call(func):  
    print(func())  
  
call(greet)
```

1 2 Generator Functions (yield)

Used to get values one-by-one (memory efficient).

python

 Copy code

```
def numbers():  
    yield 1  
    yield 2  
    yield 3
```

```
for n in numbers():  
    print(n)
```

- Stores 1 value at a time → saves memory.

1 3 Nested Functions

Function inside another function.

python

 Copy code

```
def outer():  
    def inner():  
        print("Inner Function")  
    inner()
```

```
outer()
```

1 4 Closure Functions

Inner function remembers outer function's data even after outer ends.

python

 Copy code

```
def outer(msg):  
    def inner():  
        print(msg)  
    return inner
```

```
f = outer("Python Closure")  
f()  # msg still remembered
```

1 5 Decorators

Functions that modify other functions without changing their source code.

python

 Copy code

```
def decor(func):  
    def wrapper():
```

```
    print("Welcome!")
    func()
    return wrapper

@decor
def hello():
    print("Hello Python")

hello()
```



Summary Table

Type	Example Keyword	Returns
Built-in	print(), sum()	Varies
User-Defined	def f()	Defined by user
Lambda	lambda	Expression
Positional Args	f(1,2)	Ordered
Keyword Args	f(a=1)	Named
Default Args	a=10	Optional
*args	f(1,2,3)	tuple
**kwargs	f(a=1)	dict
Recursive	Calls itself	depends
Higher-Order	Takes/returns func	func
Generator	yield	iterator
Nested	def inside def	local func
Closure	return inner func	inner+data
Decorator	@decor	modified func

Want next?

Choose one:

- 1** Practice Questions
- 2** MCQs with Answers
- 3** Interview Coding Problems

Reply with your choice (e.g., 2) 📌 😊



****** keyword means

What does ****** mean in Python Function Arguments?

In Python, ****** is used to accept **keyword arguments** in a function. This syntax is written as ****kwargs** (short for *keyword arguments*).

Meaning of ******

- **** collects **multiple keyword arguments** (name-value pairs).
- It stores them in a **dictionary**.

Example

python

 Copy code

```
def info(**details):  
    print(details)  
  
info(name="Aamir", age=20, city="Delhi")
```

Output

bash

 Copy code

```
{'name': 'Aamir', 'age': 20, 'city': 'Delhi'}
```


📌 details becomes a dictionary where:

- keys = argument names (name , age , city)
- values = passed values (Aamir , 20 , Delhi)

😬 What's the difference between *args and **kwargs ?

Syntax	Use	Data Type
*args	Multiple positional arguments	tuple
**kwargs	Multiple keyword arguments	dictionary

Example:

python

📋 Copy code

```
def demo(*args, **kwargs):  
    print(args)  
    print(kwargs)  
  
demo(1, 2, 3, name="Aamir", age=20)
```

📄 Output:

bash

📋 Copy code

```
(1, 2, 3)  
{'name': 'Aamir', 'age': 20}
```

🏆 When to use **kwargs ?

- ✓ When we don't know how many keyword arguments will be passed
- ✓ Useful for flexible functions

If you want, I can also explain using **memory diagram** or **real-life examples**.


Just ask: 👉 "Show diagram" 😊

📋 👍 🗨️ ⬆️ ↺️ ...

*keyword means

What does "keyword" mean in Python functions?

In Python, a **keyword argument** is an argument passed to a function **using the parameter name**.

 Instead of passing values by order, we pass values by **name (keyword)**.

Example of Keyword Argument

python

 Copy code


```
def student(name, age):  
    print("Name:", name)  
    print("Age:", age)  
  
student(age=20, name="Aamir")
```

Output

makefile

 Copy code

```
Name: Aamir  
Age: 20
```

 Here `age=20` and `name="Aamir"` are keyword arguments because we used the keywords (parameter names).

Why is it called "keyword"?

Because the **argument value** is connected to a **keyword (parameter name)** like:

Keyword	Value
name	"Aamir"
age	20

Difference: Positional vs Keyword

Positional Argument

Depends on order

Just values

Example: `student("Aamir", 20)`

Keyword Argument

Order doesn't matter

`name=value`

Example: `student(age=20, name="Aamir")`

Summary

- **Keyword** = name of parameter
- Keyword argument = `name=value`

If you want, I can now explain:

✓ **Default Arguments**

✓ **Keyword-only Arguments**

Which one do you want? 😊



* keyword means

What does the * (single star) mean in Python function parameters?

The symbol * is used to collect **multiple positional arguments** into a **tuple**.

It is commonly used as `*args`, where:

- * = collect many values
- `args` = name (can be anything)

Example

python

 Copy code

```
def show(*numbers):  
    print(numbers)
```

```
show(10, 20, 30)
```

**Output:**

SCSS

Copy code

```
(10, 20, 30)
```



All values (10, 20, 30) are collected into a **tuple**.

**So what does `*keyword` mean?**

When you see `*variable_name` (like `*data`, `*values`, etc.), it means:

Collect multiple positional arguments into a single variable as a tuple.

`*keyword` is **not** a Python keyword, it is a sign that **this variable will store many values**.

Example:

python

Copy code

```
def test(*fruits):  
    print(fruits)
```

```
test("apple", "banana", "mango")
```

**Output:**

bash

Copy code

```
('apple', 'banana', 'mango')
```



`fruits` becomes a **tuple of all passed values**.

**Bonus: Difference between `*` and `**`**

Symbol	Meaning	Stores As
*	Collects multiple positional arguments	tuple
**	Collects multiple keyword arguments	dict

If you want, I can explain using **diagram** or **real-life examples**.

Just reply: 👉 "Show diagram" 😊



```
ValueError                                Traceback (most recent call
last)
c:\Users\aaamir\anaconda3\Lib\site-
packages\streamlit\type_util.py in ?(data,
max_unevaluated_rows, ensure_copy, allow_styler)
    611         # Try to use index orient as back-up to
support key-value dicts
    612         return pd.DataFrame.from_dict(data,
orient="index")
--> 613         raise errors.StreamlitAPIException(
    614             f""""

~\AppData\Roaming\Python\Python312\site-
packages\pandas\core\frame.py in ?(self, data, index,
columns, dtype, copy)
    839     else:
    840         if index is None or columns is None:
--> 841             raise ValueError("DataFrame
constructor not properly called!")
    842
```

ValueError: DataFrame constructor not properly called!

The above exception was the direct cause of the following exception:

```
StreamlitAPIException                    Traceback (most
recent call last)
```

Cell In[4], line 7

```
4 set2 = (1,2,3,4,5,6,7,8,19,10)
6 for i in set2 :
----> 7     table(i)
```

File c:\Users\amir\anaconda3\Lib\site-packages\streamlit\runtime\metrics_util.py:397, in gather_metrics.<locals>.wrapped_func(*args, **kwargs)

```
395     _LOGGER.debug("Failed to collect command
telemetry", exc_info=ex)
396 try:
--> 397     result = non_optional_func(*args, **kwargs)
398 except RerunException as ex:
399     # Duplicated from below, because static
analysis tools get confused
400     # by deferring the rethrow.
401     if tracking_activated and command_telemetry:
```

File c:\Users\amir\anaconda3\Lib\site-packages\streamlit\elements\arrow.py:283, in ArrowMixin.table(self, data)

```
280 default_uuid = str(hash(delta_path))
282 proto = ArrowProto()
--> 283 marshall(proto, data, default_uuid)
284 return self.dg._enqueue("arrow_table", proto)
```

File c:\Users\amir\anaconda3\Lib\site-packages\streamlit\elements\arrow.py:376, in marshall(proto, data, default_uuid)

```
374     proto.data =
type_util.pyarrow_table_to_bytes(data)
375 else:
--> 376     df = type_util.convert_anything_to_df(data)
377     proto.data = type_util.data_frame_to_bytes(df)
```

File c:\Users\amir\anaconda3\Lib\site-packages\streamlit\type_util.py:613, in convert_anything_to_df(data, max_unevaluated_rows, ensure_copy, allow_styler)

```
610         with contextlib.suppress(ValueError):
611             # Try to use index orient as back-up to
support key-value dicts
612             return pd.DataFrame.from_dict(data,
```

```

orient="index")
--> 613     raise errors.StreamlitAPIException(
        614         f"""
        615 Unable to convert object of type {type(data)} to
pandas.DataFrame.
        616 Offending object:
        617
py
        618 {data}
        619
"""
        620     ) from ex

StreamlitAPIException:
Unable to convert object of type <class 'int'> to
pandas.DataFrame.
Offending object:
py
1

```

You are getting this error because:

`st.table()` cannot display a single integer (`int`) directly. It expects a **DataFrame-like object**.

You are doing this:

python

 Copy code

```

for i in set2 :
    table(i)  # ✗ wrong: i is just an int

```

⚠ Streamlit tries to convert that integer into a DataFrame → Error:

kotlin

 Copy code

`ValueError: DataFrame constructor not properly called!`

✅ Correct Fix

Convert the value to a list, dict, or DataFrame before using `st.table()` .

✔ Option 1: Show values as a DataFrame row

python

📄 Copy code

```
import streamlit as st
import pandas as pd

set2 = (1,2,3,4,5,6,75,8,19,10)

for i in set2:
    st.table(pd.DataFrame({"Value": [i]}))
```

✔ Option 2: Show whole tuple as a table at once

python

📄 Copy code

```
import streamlit as st
import pandas as pd

set2 = (1,2,3,4,5,6,75,8,19,10)

st.table(pd.DataFrame({"Numbers": set2}))
```

✔ Option 3: Use `st.write()` instead of `table`

python

📄 Copy code

```
import streamlit as st

set2 = (1,2,3,4,5,6,75,8,19,10)

for i in set2:
    st.write(i) # Works without DataFrame
```

💡 Why does `st.table()` need a DataFrame?

Because `table()` is designed to show **tabular format (rows & columns)**.

A single integer has **no column/row structure**, so Streamlit cannot convert it to a table

automatically.

If you want, I can show:

- ✦ How to make a table with **index + value**
- ✦ How to show a table with **keys and values**
- ✦ How to style tables (colors, width)

Which one do you want? 😊



2025-11-21 23:11:41.406

Warning: to view this Streamlit app on a browser, run it with the following command:

```
streamlit run  
C:\Users\amir\AppData\Roaming\Python\Python312\site-packages\ipykernel_launcher.py [ARGUMENTS]
```

This message is **NOT an error** — it is just a **warning from Streamlit**.

You are trying to run Streamlit **inside Jupyter Notebook / IPython**, so Streamlit tells you:

arduino

Copy code

Warning: to view **this** Streamlit app on a browser, run it with **this** command:
streamlit run C:\Users\amir\...\ipykernel_launcher.py

✦ Why this warning appears?

Because Streamlit apps must be run from **terminal / command prompt**, not inside a notebook.

✅ How to Run Your Streamlit App Correctly

1. Save your code in a Python file, for example:

```
app.py
```

2. Open **Command Prompt / Terminal** and run:

```
arduino
```

 Copy code

```
streamlit run app.py
```

Example

app.py

```
python
```

 Copy code

```
import streamlit as st
```

```
st.title("Welcome to Streamlit")
```

```
st.write("Hello, Aamir!")
```

Now run on terminal:

```
arduino
```

 Copy code

```
streamlit run app.py
```

Your app will open in the browser.

If you want to run Streamlit inside Jupyter?

Install this:

```
nginx
```

 Copy code

```
pip install jupyter-streamlit
```

and then use magic command:

```
python
```

 Copy code

```
%streamlit run app.py
```

Summary

Situation	What to do
Running in Notebook	You get a warning
To open in browser	Run using <code>streamlit run filename.py</code>