Two Meanings of 'Global' in Python

1. Global Scope — Inside a Single Module

When talking about scope, 'global' means: 'Visible throughout the current module (the current .py file).' In Python's LEGB rule (Local → Enclosing → Global → Built-in):

- The Global scope refers to names defined at the top level of a module.
- These are accessible to all functions within that same module (unless shadowed by locals).

```
x = 10   # global within this module

def show():
    print(x)  # refers to the global x above

show()   # prints 10
```

Here, x lives in the module's global namespace, so every function inside the same file can access it (read-only by default).

2. The 'global' Keyword — Modifying Global Variables Within a Function

Normally, if you assign to a variable inside a function, Python treats it as local. Let's see what happens if you try to modify a global variable without declaring it global:

```
counter = 0   # global variable

def increment():
    counter += 1   # tries to modify global counter

increment()
# UnboundLocalError: local variable 'counter' referenced before assignment
```

Python throws an UnboundLocalError because assigning to 'counter' inside the function makes it a local variable, and there is no local variable initialized before use.

Now let’s fix it using the 'global' keyword:

```
counter = 0   # global variable
def increment():
    global counter   # refers to the global counter in this module
    counter += 1

increment()
print(counter)  # prints 1
```

By declaring 'global counter', the function tells Python to use the variable from the module’s global namespace. The assignment now updates the global value instead of creating a new local variable.

3. Global Variables Across Modules

When you import one module into another, the global variables of the imported module live in that module’s own namespace — not in the importing module’s namespace. Each module in Python has its own separate global namespace.

```
# globals.py
x = 42


# main.py
import globals

print(globals.x)   # access globals.py's variable
globals.x = 99     # modify it
print(globals.x)   # prints 99
```

Even though both files are part of the same program, their globals are not shared directly — you access them through the module name (globals.x).

If you just do:

```
from globals import x
```

you get a copy of the reference to x in your local namespace. Changing x in one module does not automatically update the other’s variable, unless both refer to a mutable object.

4. Summary — 'Global' Has Two Contexts

Context	Meaning	Scope	Example	Shared?
Scope (LEGB)	Variables defined at the top level of a	Within the same .py file	x = 10	Yes, within one module

	module			
'global' keyword	Allows function to modify variable in the module's global namespace	Local to one module	global counter	Affects only that module
Cross-module 'globals'	Top-level variables in other modules	Each module has its own namespace	globals.x	Must access via module name

5. Key Takeaways

• Each module in Python has its own global namespace. That's why 'global' doesn't cross module boundaries.

• The 'global' keyword affects only variable lookup within a single module — not across imports.

• To share data between modules, use imports

Example — import

```
# globals.py
shared_value = 0


# a.py
import globals
globals.shared_value += 1


# b.py
import globals
print(globals.shared_value)   # prints 1
```

This is the safe, explicit way to share global state across modules.