Python Stack Frames

Next Slide →

Introduction

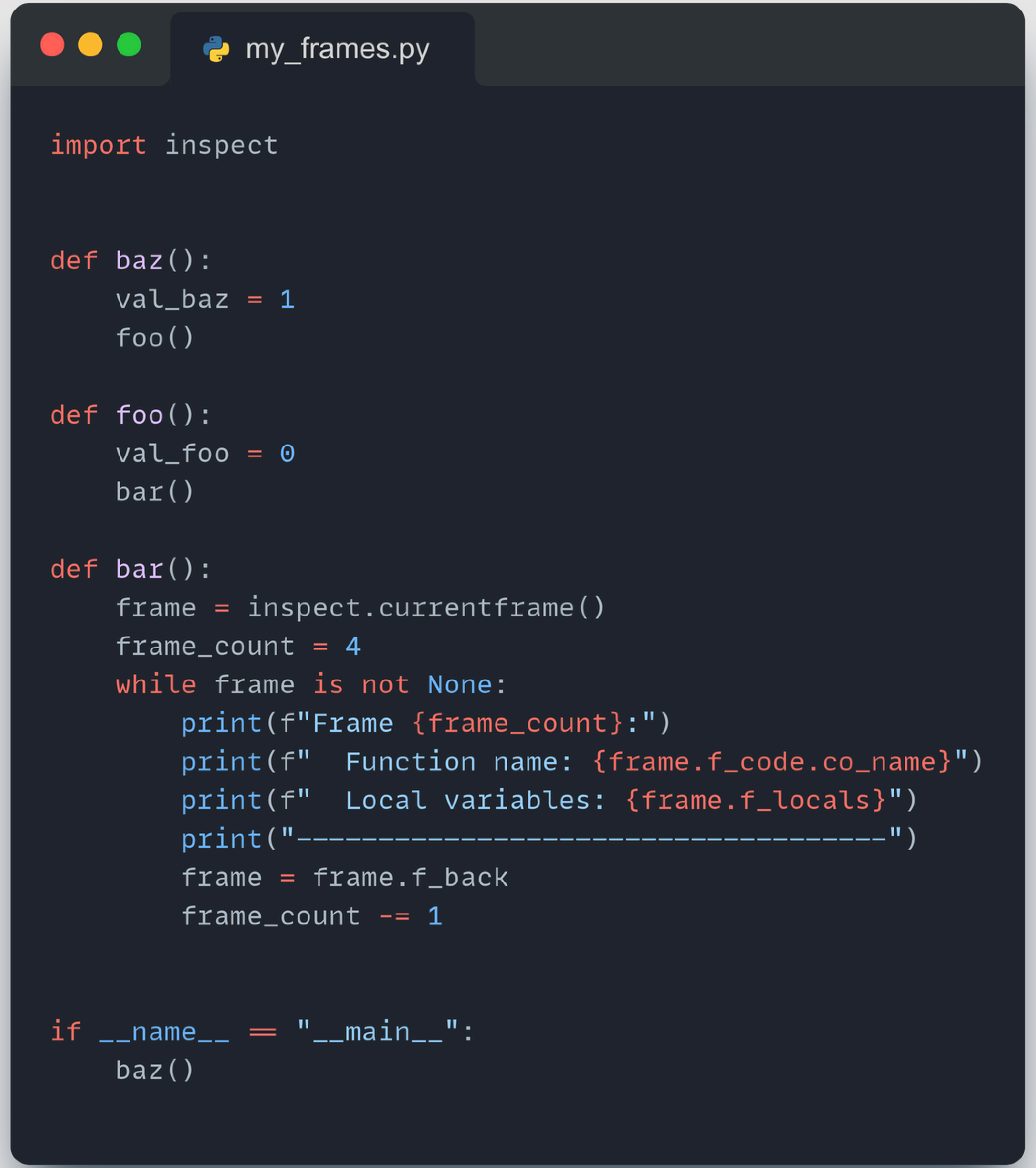
Python uses stack frames to manage function calls. Each frame holds details like the function name, local variables, and links to the previous frame. When a function is called, a new frame is created and added to the stack. Once the function finishes, the frame is removed. This stacking process reflects the program's flow.

What Next Slides Show:

1️⃣ **Code (Slide 3)**: A Python script demonstrating how to access and traverse stack frames using the inspect module.

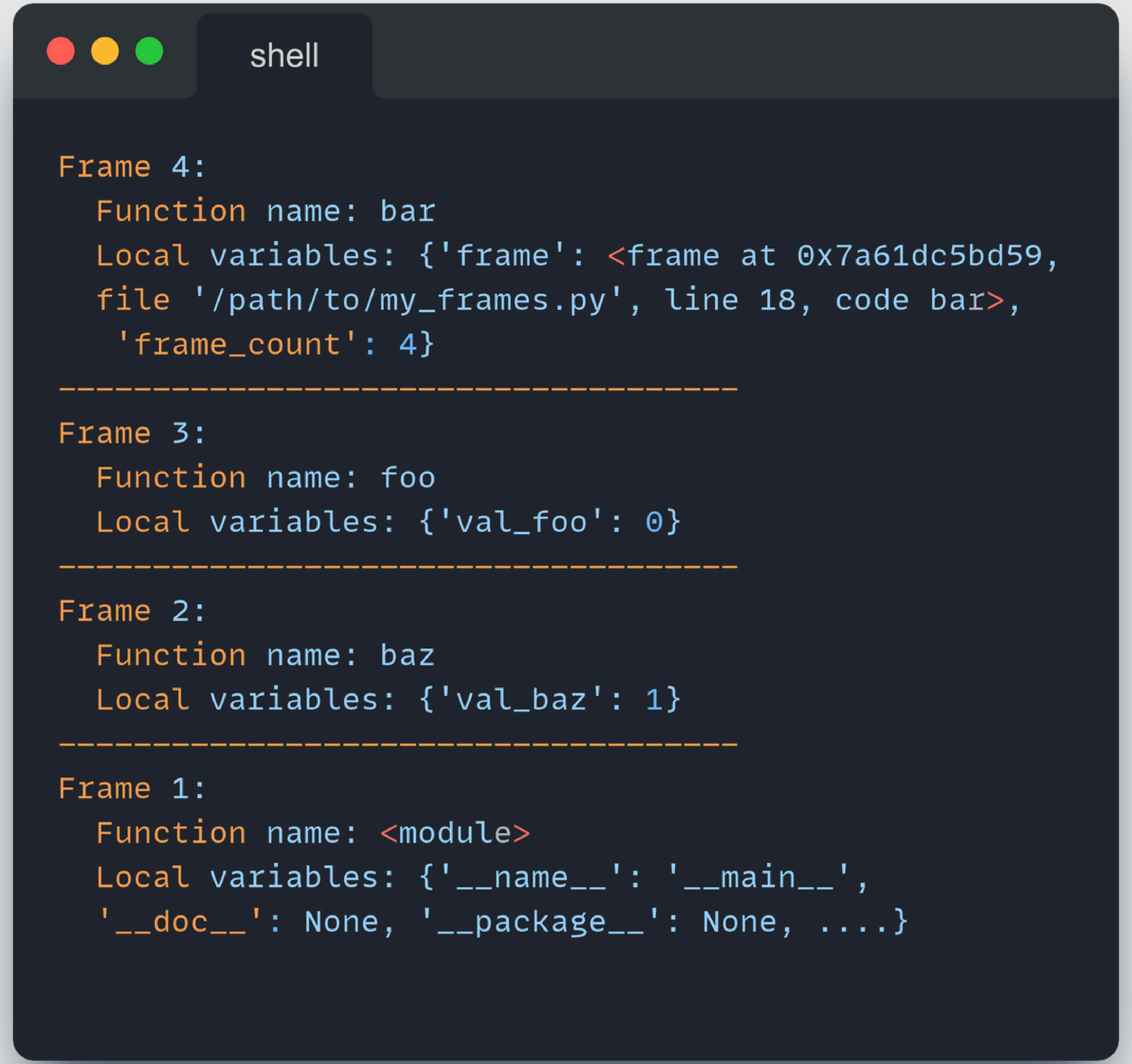
2️⃣ **Output (Slide 4)**: A snapshot of the stack frames at runtime, showing each frame’s function name and local variables as they are stacked and unwound.
This visualization highlights how functions like baz, foo, and bar are layered on the stack and how you can trace their relationships step by step.

Next Slide →


my_frames.py

```python
import inspect


def baz():
    val_baz = 1
    foo()

def foo():
    val_foo = 0
    bar()

def bar():
    frame = inspect.currentframe()
    frame_count = 4
    while frame is not None:
        print(f"Frame {frame_count}:")
        print(f"  Function name: {frame.f_code.co_name}")
        print(f"  Local variables: {frame.f_locals}")
        print("------------------------------------")
        frame = frame.f_back
        frame_count -= 1


if __name__ == "__main__":
    baz()
```

Next Slide



```
Frame 4:
  Function name: bar
  Local variables: {'frame': <frame at 0x7a61dc5bd59,
  file '/path/to/my_frames.py', line 18, code bar>,
   'frame_count': 4}
------------------------------------
Frame 3:
  Function name: foo
  Local variables: {'val_foo': 0}
------------------------------------
Frame 2:
  Function name: baz
  Local variables: {'val_baz': 1}
------------------------------------
Frame 1:
  Function name: <module>
  Local variables: {'__name__': '__main__',
  '__doc__': None, '__package__': None, ....}
```