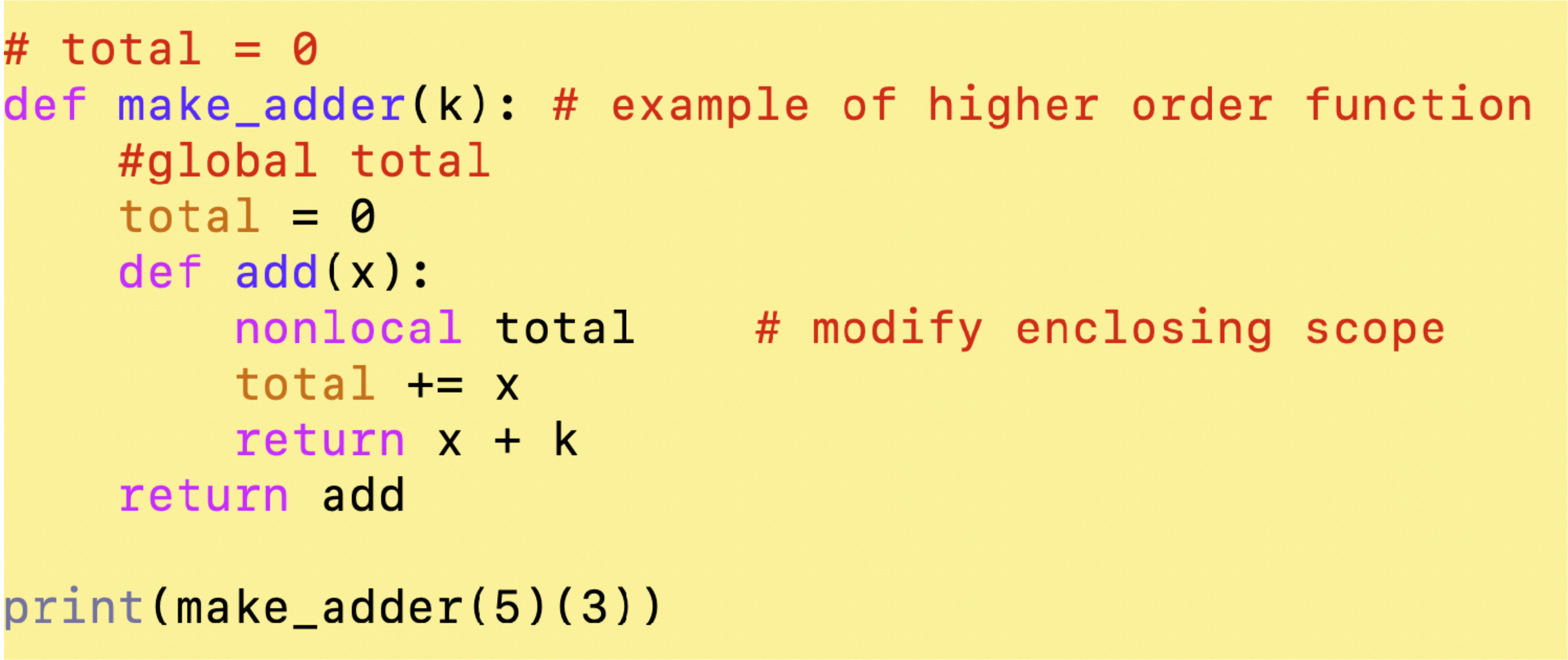
COL1000: Introduction to Programming

Functions

Subodh Sharma | Lec 19 | Sept 24

Functions — Semantics

Scope - Local, Global, Nonlocal



```python
# total = 0
def make_adder(k): # example of higher order function
    #global total
    total = 0
    def add(x):
        nonlocal total     # modify enclosing scope
        total += x
        return x + k
    return add

print(make_adder(5)(3))
```

- **Try declaring total as a global and modifying it in the nested functions**

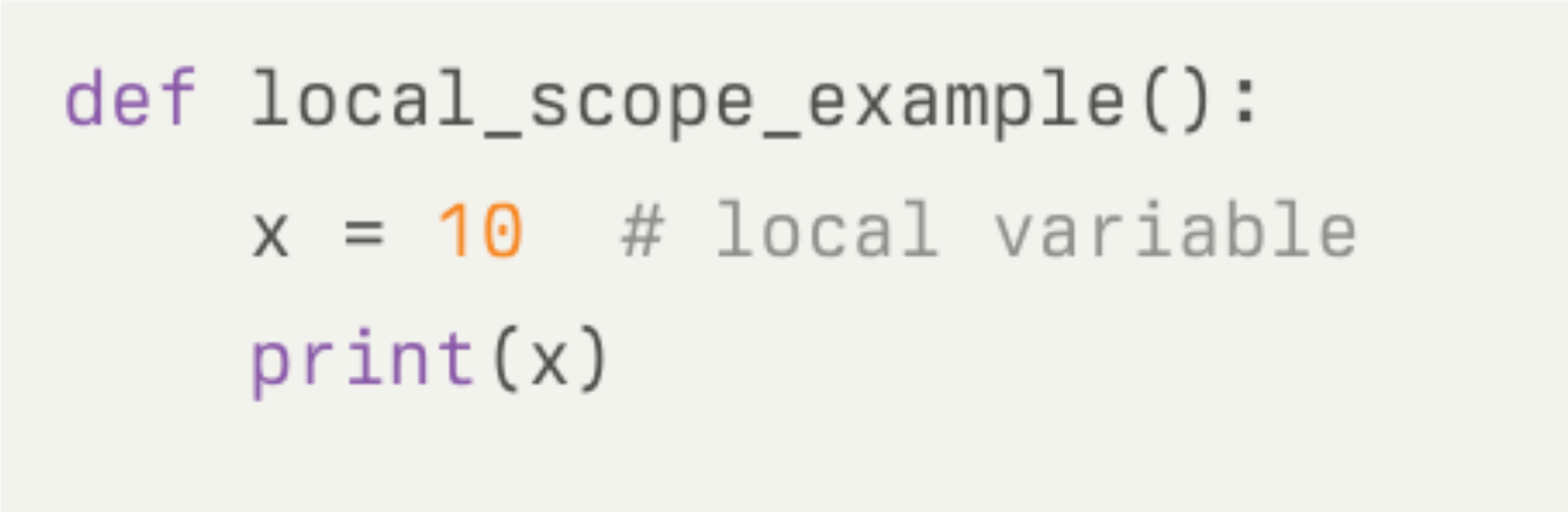
Functions — Semantics

Scope - Local, Global, Nonlocal

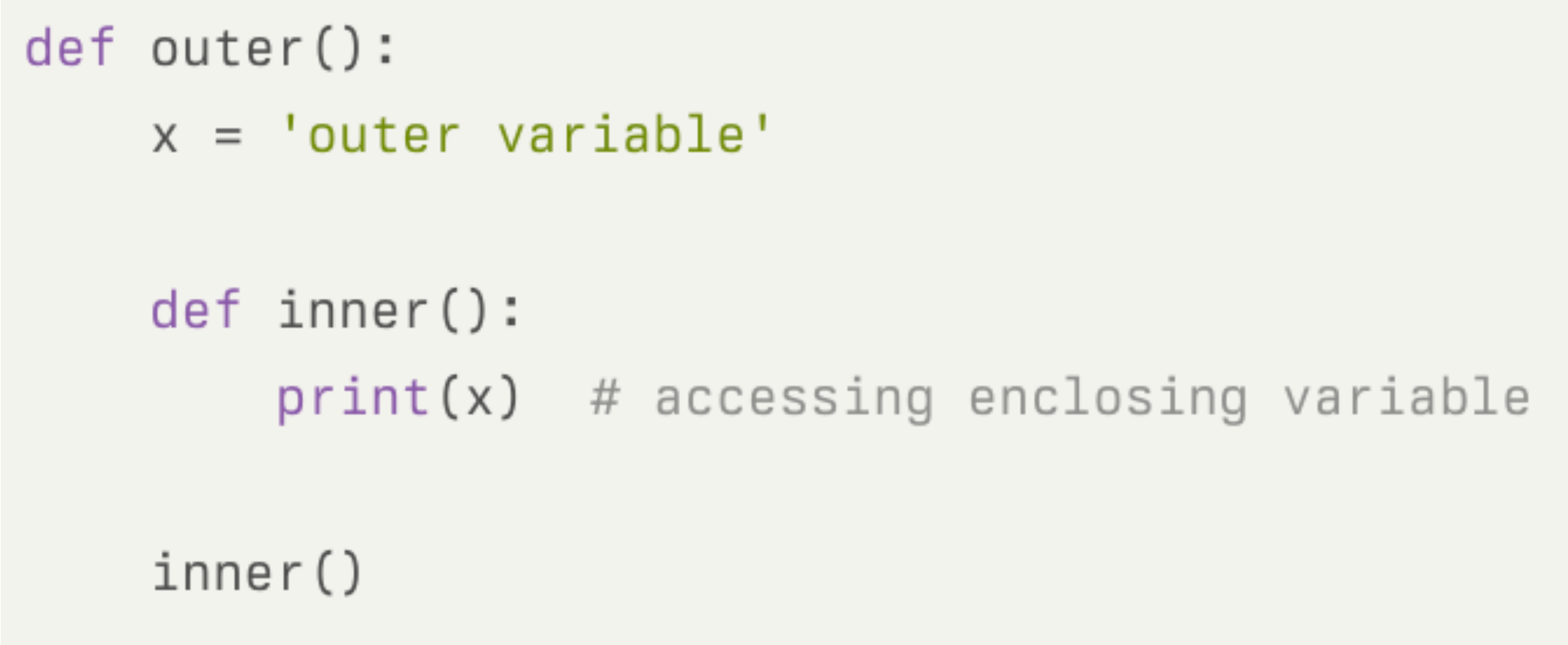
- **Local scope:** Vars defined inside a function are accessible within that function

- **Enclosing scope:** In nested functions, inner functions can access vars from the outer enclosing function

- **Global Scope:** Vars defined outside all functions have global scope



```
def local_scope_example():
    x = 10  # local variable
    print(x)
```

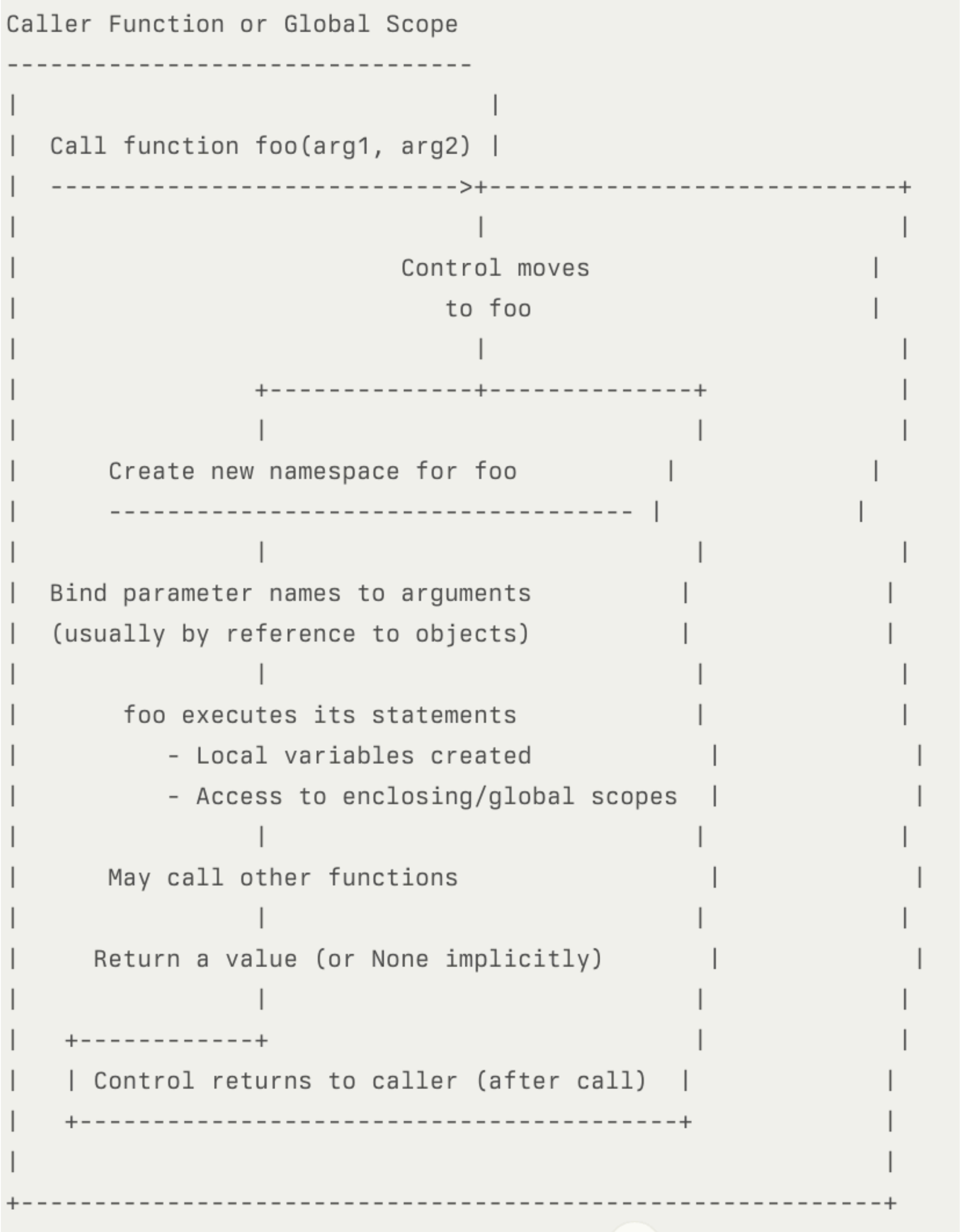


```
def outer():
    x = 'outer variable'

    def inner():
        print(x)  # accessing enclosing variable

    inner()
```

Functions — Semantics

Execution Semantics

- **Namespace for the callee** — creation of local scope
    - Each new function invocation creates a frame on the memory stack
- Parameter binding with arguments
    - **Either by reference or by value**



```
Caller Function or Global Scope
--------------------------------
|                                |
|  Call function foo(arg1, arg2) |
|  ---------------------------->+----------------------------+
|                               |                            |
|                          Control moves                   |
|                             to foo                       |
|                               |                            |
|                +--------------+--------------+             |
|                |                             |             |
|      Create new namespace for foo          |             |
|      ------------------------------------ |             |
|                |                             |             |
|  Bind parameter names to arguments          |             |
|  (usually by reference to objects)          |             |
|                |                             |             |
|       foo executes its statements            |             |
|          - Local variables created            |             |
|          - Access to enclosing/global scopes  |             |
|                |                             |             |
|      May call other functions                 |             |
|     Return a value (or None implicitly)       |             |
|                |                             |             |
|   +------------+                             |             |
|   | Control returns to caller (after call)  |             |
|   +-----------------------------------------+             |
|                                                           |
+-----------------------------------------------------------+
```

Functions — Semantics

Parameter Binding — Pass by Object Reference

- **In Python —** when an var is passed to a function as an argument, its reference is created (referring to the same object) and transferred to the callee's namespace!
  - In that sense it is neither transfer of value or direct reference.
- **Case 1: When the passed object is immutable**
  - Since the reference cannot be changed, any modification creates a copy of the object and the passed reference now points to the modified object
- **Case 2: When the passed object is mutable**
  - The passed reference points to the same reference with which the function was invoked

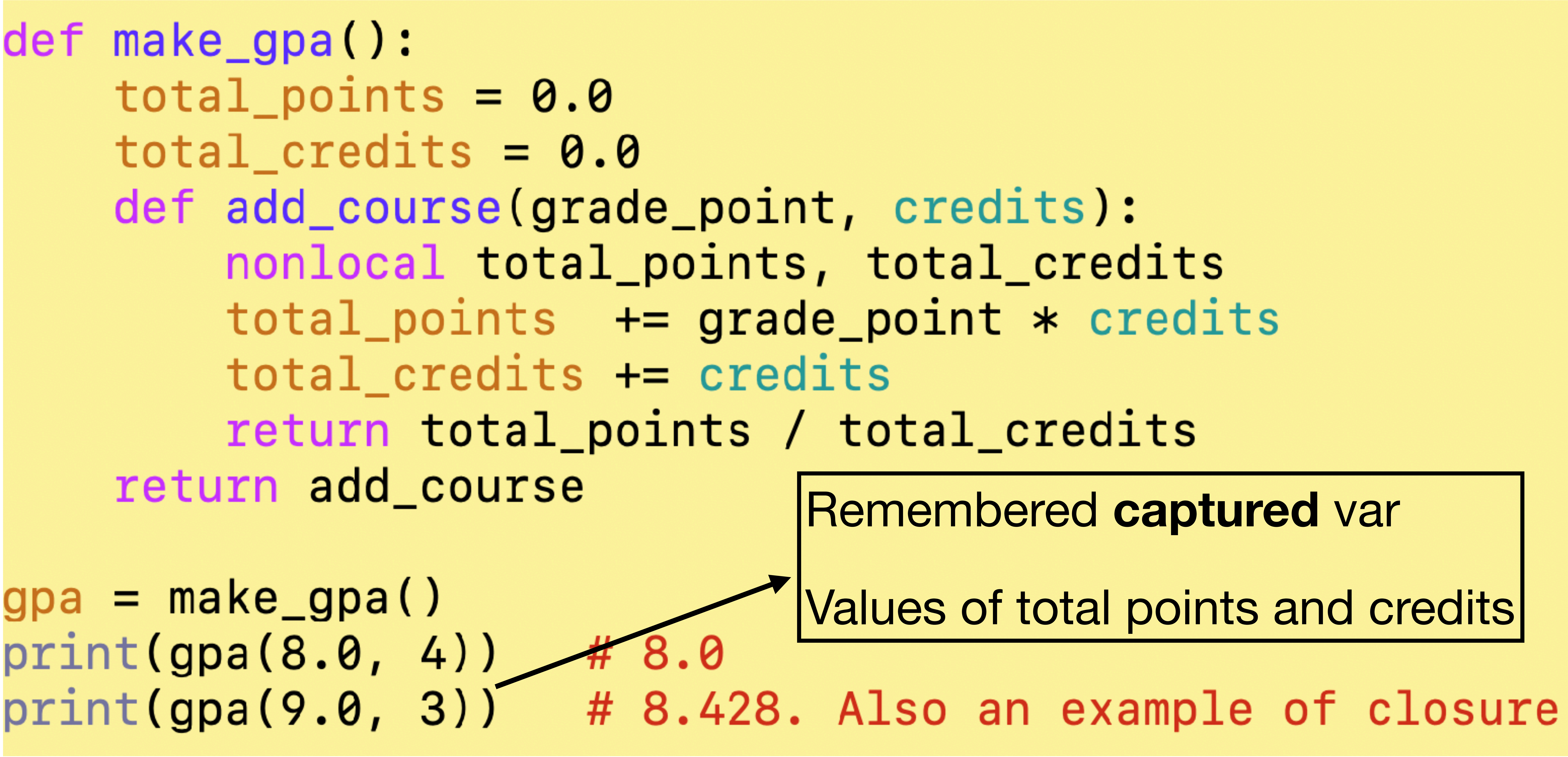
Functions as First-class objects

What does it mean?

- First-class => Functions can be
  - **Stored in to variables** or data structures like lists, etc.
  - **Passed as arguments** to other functions (Eg: filters, accumulators etc.)
  - Can be **returned** from another functions

Functions: Closures

- A **closure** is a function object that **remembers values from its enclosing scope even after that scope has finished its execution**



```python
def make_gpa():
    total_points = 0.0
    total_credits = 0.0
    def add_course(grade_point, credits):
        nonlocal total_points, total_credits
        total_points  += grade_point * credits
        total_credits += credits
        return total_points / total_credits
    return add_course

gpa = make_gpa()
print(gpa(8.0, 4))    # 8.0
print(gpa(9.0, 3))    # 8.428. Also an example of closure
```

Remembered **captured** var

Values of total points and credits

Functions: Lambda

- **Lambda functions**: They are anonymous functions (i.e. don’t have a user-specified name)
  - Syntax: `lambda arguments: expression`
    - Arguments can be 0 or more, but **have a single expression** whose **results are returned**!
    - E.g., `add_five = lambda x: x+5; print(add_five(7)) #12`
    - E.g., `lst = list(map(lambda x: ?, [1,2,3]))`

Functions: Exceptions

- Exceptions in functions **denote errors or unexpected conditions**
- There are two ways of handling exceptions in the code:
  - Explicitly **raising the errors with an appropriated exception message**
  - **Catching** them to handle them gracefully
  - Let us see examples of each!