CS134:
Scope

Announcements & Logistics

- **HW 5** due Monday @ 10pm
- **Lab 4 Part 2** due Wednesday/Thursday 10pm
    - There will be a Gradescope - Part 2 assignment
- **Midterm reminders:**
    - **Midterm Exam** is Thursday, October 17 at 6pm or 8pm in TPL203
        - **Midterm Review** is in place of class on Wednesday 10/16 during class, 9am-11:50am **Bring Questions!!**
        - **To Prepare**: *Redo:* [homework, **practice exams**, POGIL questions (including Application Questions), pre-labs & labs] w paper & pencil...then check your answers with Python!
- **Final Exam** schedule is posted: Wednesday, December 11 at 9:30am

Do You Have Any Questions?

Last Time: Mutability & Aliasing

- Attempts to change **immutable** objects (e.g., strings) produce **clones**
  - Changes to clones do not affect originals
    - No aliasing!
- We can create **aliases** of **mutable** objects
  - Aliases refer to the same object, so changes to that object through any alias affect value that other aliases point to
- For the list data type, **+=** (append operator) mutates the list!

Goal was to demystify surprising behavior:
nothing in computer science is magic!

Today's Plan

- **Scope**: variables, functions, objects have limited accessibility/visibility.
  - Understanding how this works helps us make decisions about where to define variables/functions/objects

Goal is to again demystify surprising behavior: nothing in computer science is magic!

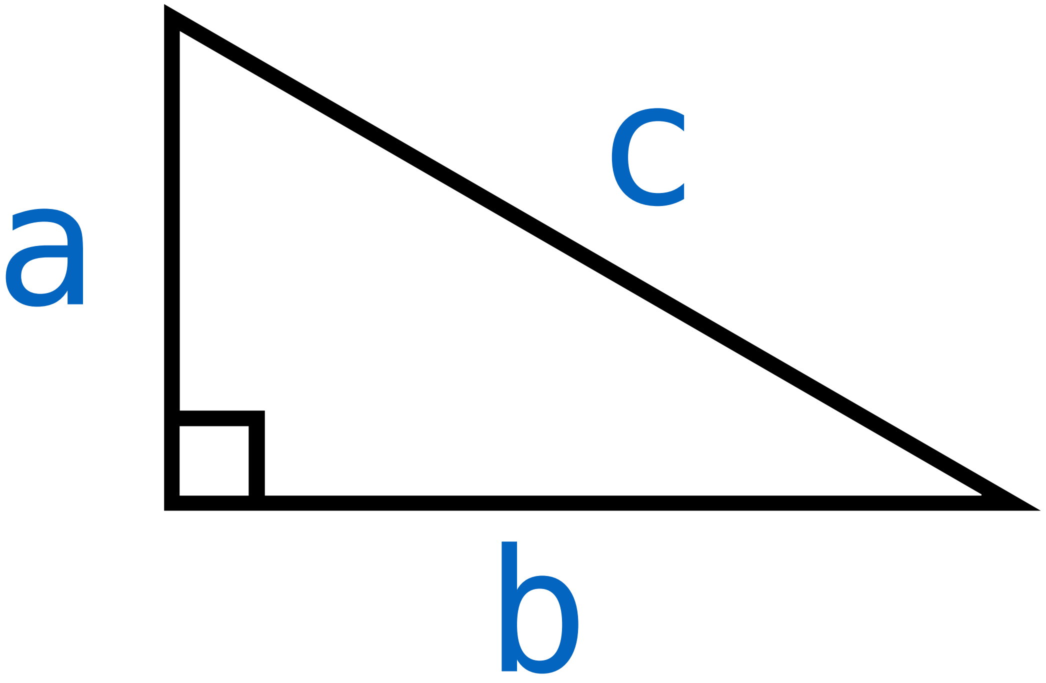
What gets printed to the screen?

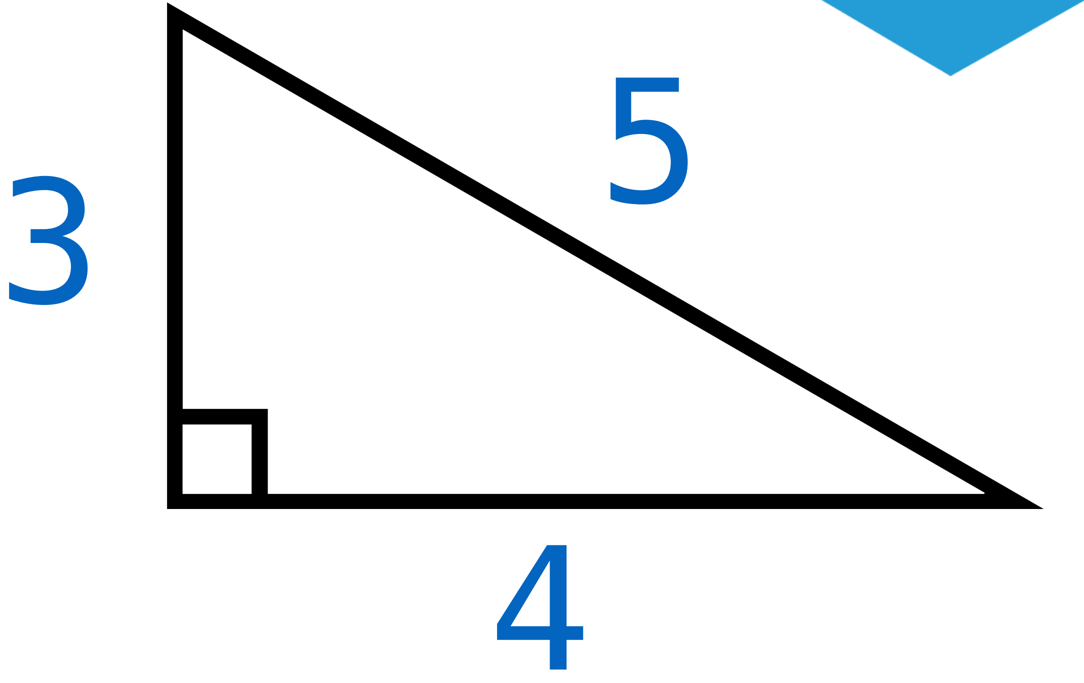
```python
a = 3
b = 4
def square(x):
    return x * x
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```

What gets printed to the screen?

```
a = 3
b = 4
def square(x):
    return x * x
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```

```
5.0
```





What gets printed to the screen?

What if we make this change?

```
a = 3
b = 4
def square(a):
    return a * a
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```

What gets printed to the screen?

Same output!

```
a = 3
b = 4
def square(a):
    return a * a
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```

```
5.0
```

What gets printed to the screen?

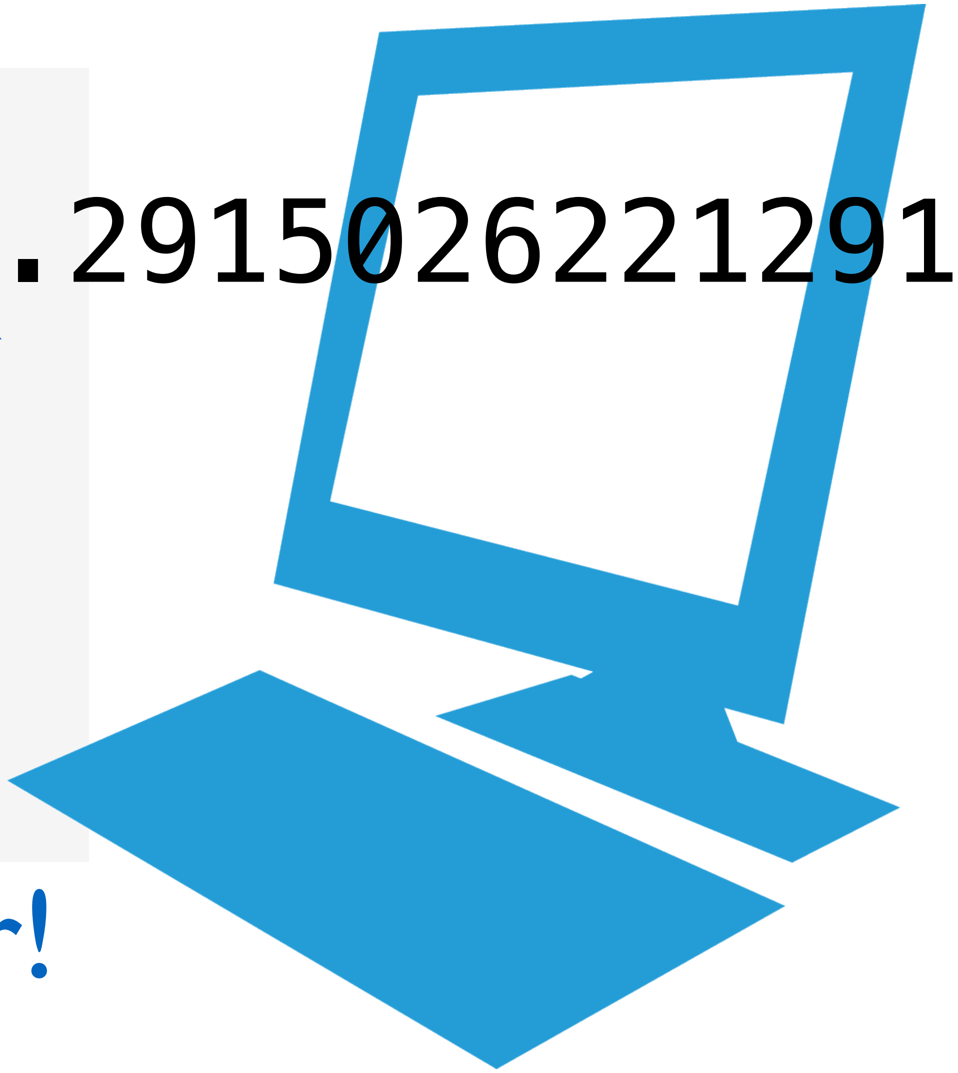
What if we make this change?

```
a = 3
b = 4
def square(a): b
    return a * a
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```

What gets printed to the screen?

Not the same output

```
a = 3
b = 4
def square(a): b
    return a * a
c = square(a) + square(b)
c = pow(c, 0.5)
print(c)
```



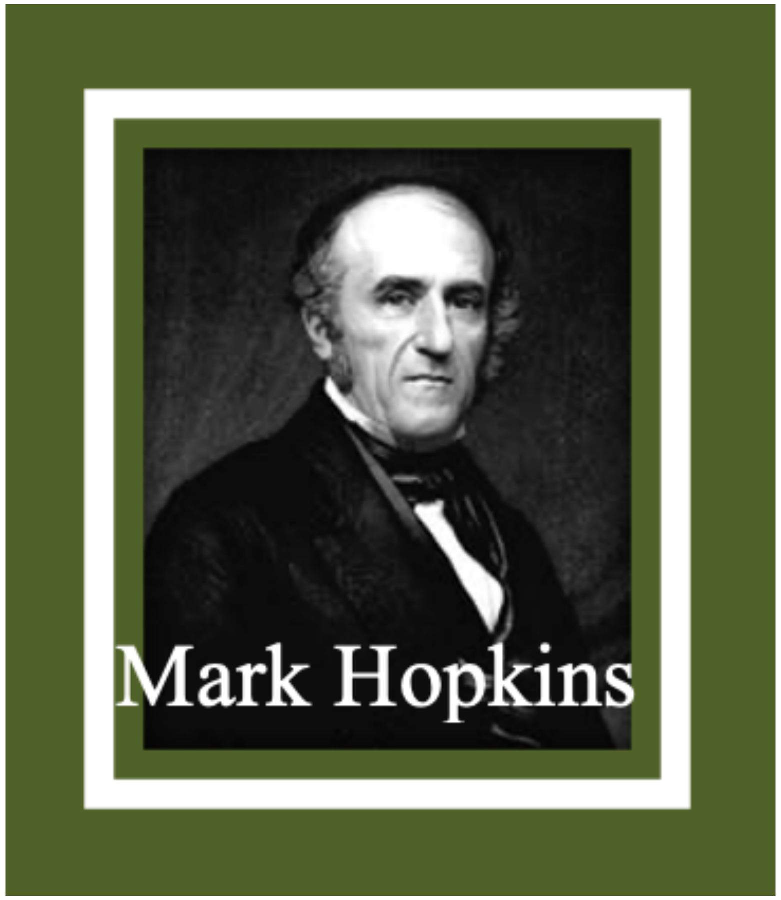
```
5.291502622129181
```

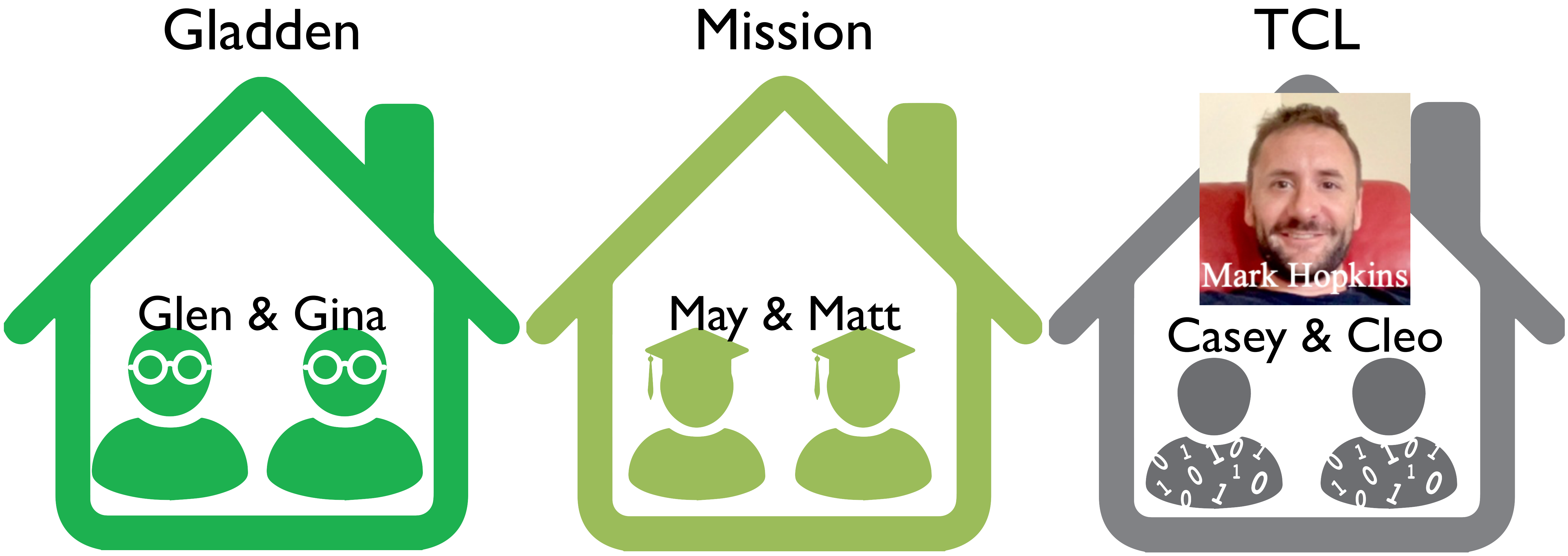
But also not an error!

Big Question: When we reuse variable names, how does Python know what a variable refers to?

Scope Diagram

- In Gladden & Mission dorms, "Mark Hopkins" refers to Mark Hopkins '1824, President of Williams College 1836-1872.
- In TCL, "Mark Hopkins" refers to Professor Mark Hopkins, who started working at Williams in 2022.





Let's see it in python!

scope.py

```python
mar_hop = 111119 # Mark Hopkins ’1824 student ID number

def gladden():
    glen = 223456 # Glen’s student ID number
    gina = 287654 # Gina’s student ID number
    print(glen, gina, mar_hop)

def mission():
    may = 277777 # May's student ID number
    matt = 288888 # Matt's student ID number
    print(may, matt, mar_hop)

def tcl():
    mar_hop = 998877 # Mark Hopkins ’2022 student ID number
    casey = 212233 # Casey’s student ID number
    cleo = 233444 # Cleo’s student ID number
    print(casey, cleo, mar_hop)

```

```python
if __name__ == '__main__':
    gladden() # prints?
    mission() # prints?
    tcl() # prints?
```

Let's see it in python!

scope.py

```
mar_hop = 111119 # Mark Hopkins ’1824 student ID number

def gladden():
    glen = 223456 # Glen’s student ID number
    gina = 287654 # Gina’s student ID number
    print(glen, gina, mar_hop)

def mission():
    may = 277777 # May's student ID number
    matt = 288888 # Matt's student ID number
    print(may, matt, mar_hop)

def tcl():
    mar_hop = 998877 # Mark Hopkins ’2022 student ID number
    casey = 212233 # Casey’s student ID number
    cleo = 233444 # Cleo’s student ID number
    print(casey, cleo, mar_hop)
```

```
if __name__ == '__main__':
    gladden()   # 223456  287654  111119
    mission()   # 277777  288888  111119
    tcl()       # 212233  233444  998877
```

Let's see it in python!

scope.py

```python
mar_hop = 111119 # Mark Hopkins ’1824 student ID number

def gladden():
    glen = 223456 # Glen’s student ID number
    gina = 287654 # Gina’s student ID number
    print(glen, gina, mar_hop)

def mission():
    may = 277777 # May's student ID number
    matt = 288888 # Matt's student ID number
    print(may, matt, mar_hop)
    print(glen)
def tcl():
    mar_hop = 998877 # Mark Hopkins ’2022 student ID number
    casey = 212233 # Casey’s student ID number
    cleo = 233444 # Cleo’s student ID number
    print(casey, cleo, mar_hop)
    print(glen)
```

What if we print(glen) in mission() or tcl()?

```python
if __name__ == '__main__':
    gladden()
    mission()
    tcl()
```

NameError: name 'glen' is not defined

Local Before Global

- *When python encounters a new term, like a variable or function name, it* ***first*** *looks* ***local****ly, before looking higher up.*

- *If it can't find the value assigned to the term, you get a* `NameError`*.*

`triple(num)`

A Small Example

Example: triple(num)

A

```
def triple(num):
    multiplier = 3
    return multiplier * num
answer = triple(5)
print(answer)
```

in function

B

above/before function

```
multiplier = 3
def triple(num):
    return multiplier * num
answer = triple(5)
print(answer)
```

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

below/after function

D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

after function call

What will each of these print?

Example: triple(num)

A

```
def triple(num):
    multiplier = 3
    return multiplier * num
answer = triple(5)
print(answer)
```

in function

15

B

above/before function

```
multiplier = 3
def triple(num):
    return multiplier * num
answer = triple(5)
print(answer)
```

15

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

below/after function

15

D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

after function call

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

What will each of these print?

Function Frame Model

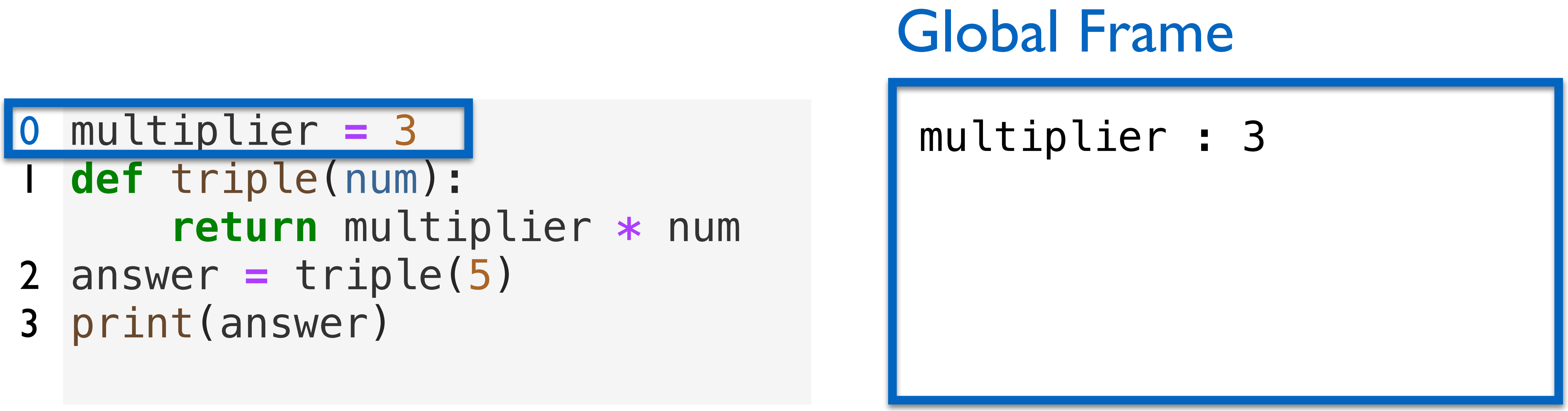
Scope: Function Frame Model

- By default, python reads code one line at a time, starting from line 0

```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```

Scope: Function Frame Model

- At first, when variables are assigned, their values are stored in the **global frame**



```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```

Global Frame

```
multiplier : 3
```

Scope: Function Frame Model

- Function definitions are treated like a single line of code
- A `def` statement does **not** call the function, it just defines it

```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```

Global Frame

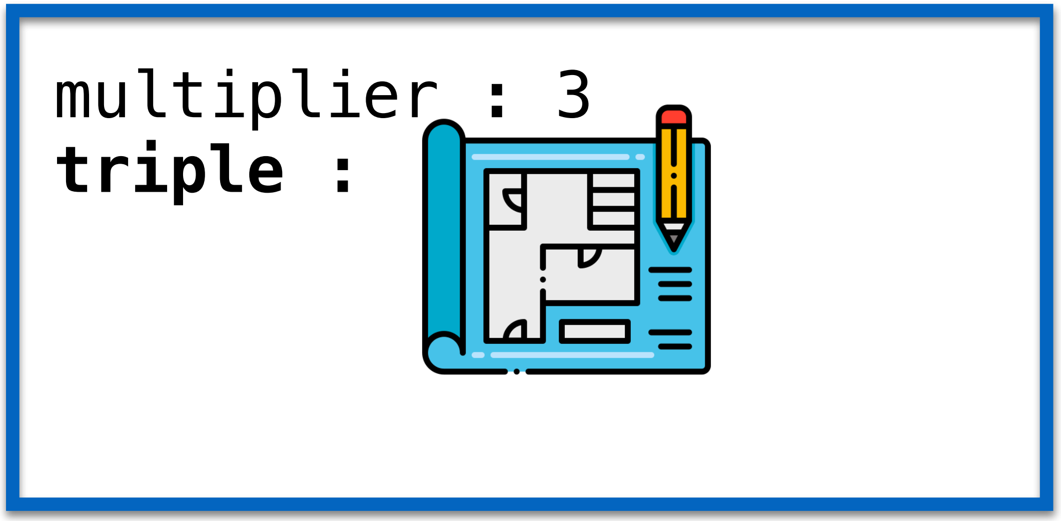
```
multiplier : 3
triple : multiplier * num
```

Scope: Function Frame Model

- Function definitions are treated like a single line of code
- A `def` statement does **not** call the function, it just defines it
- Effectively, it assigns the name of the function to a blueprint for computing the function

```python
multiplier = 3
def triple(num):
    return multiplier * num
answer = triple(5)
print(answer)
```

Global Frame



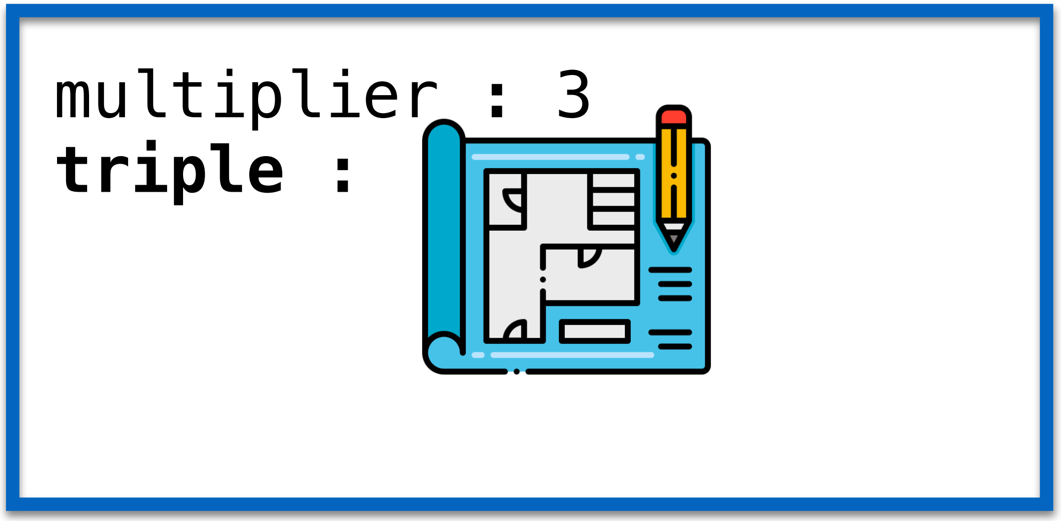
```
multiplier : 3
triple :
```

Scope: Function Frame Model

- To execute an assignment statement, python first computes the value of its **right-hand side**
- In this case, the **right-hand side** calls the `triple` function

```python
multiplier = 3
def triple(num):
    return multiplier * num
answer = triple(5)
print(answer)
```

Global Frame

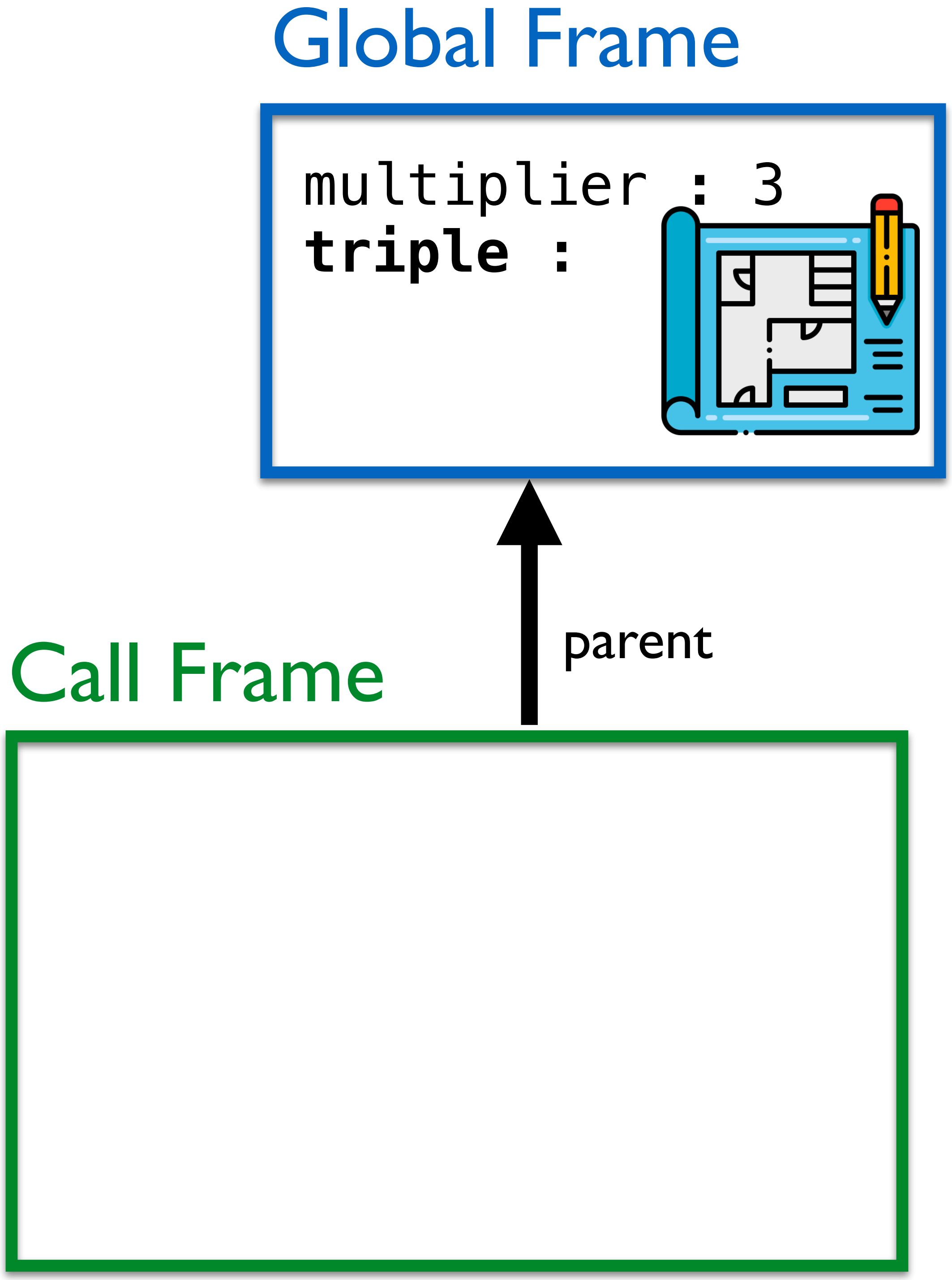


```
multiplier : 3
triple :
```

Scope: Function Frame Model

- When a function is called, a new frame is created to record the variables used by that function

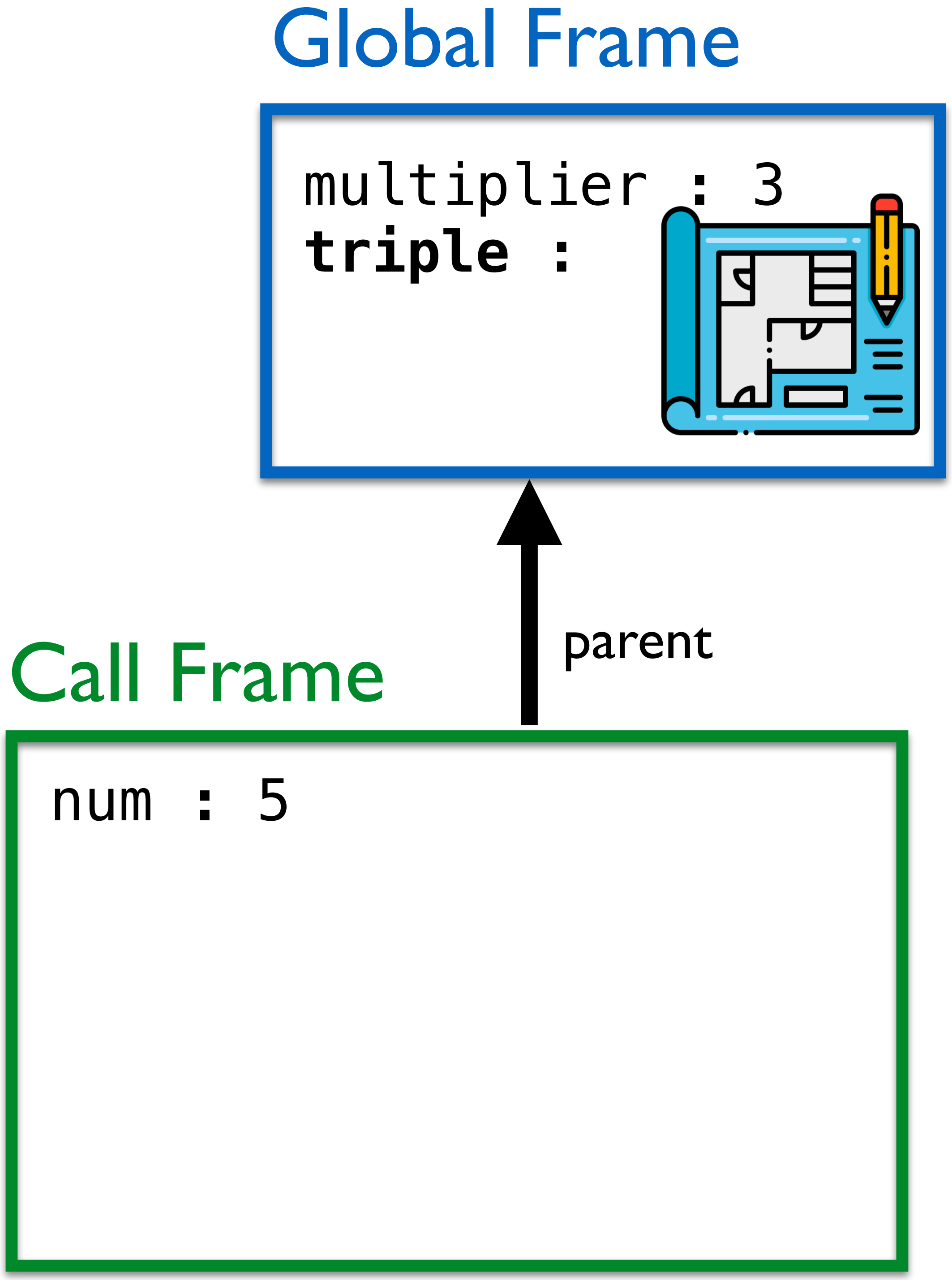
```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Scope: Function Frame Model

- First, the values of the argument variables are recorded in the **call (i.e., function) frame**

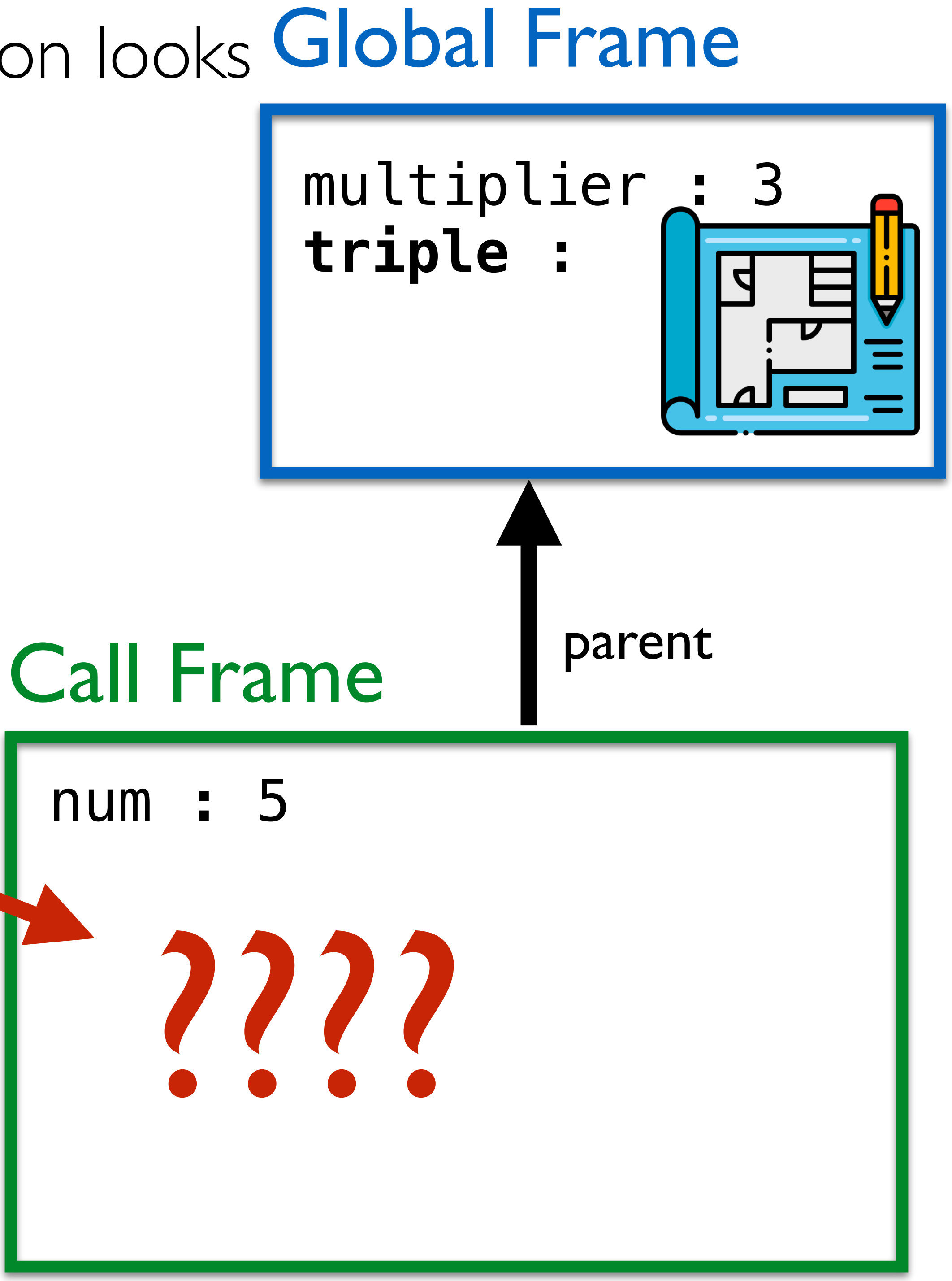
```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Scope: Function Frame Model

- Then, the lines of the function are executed in order
- To look up the value of a variable, first python looks in the **call frame**

```
multiplier = 3
def triple(num):
    return multiplier * num
answer = triple(5)
print(answer)
```



Global Frame

```
multiplier : 3
triple :
```

parent

Call Frame

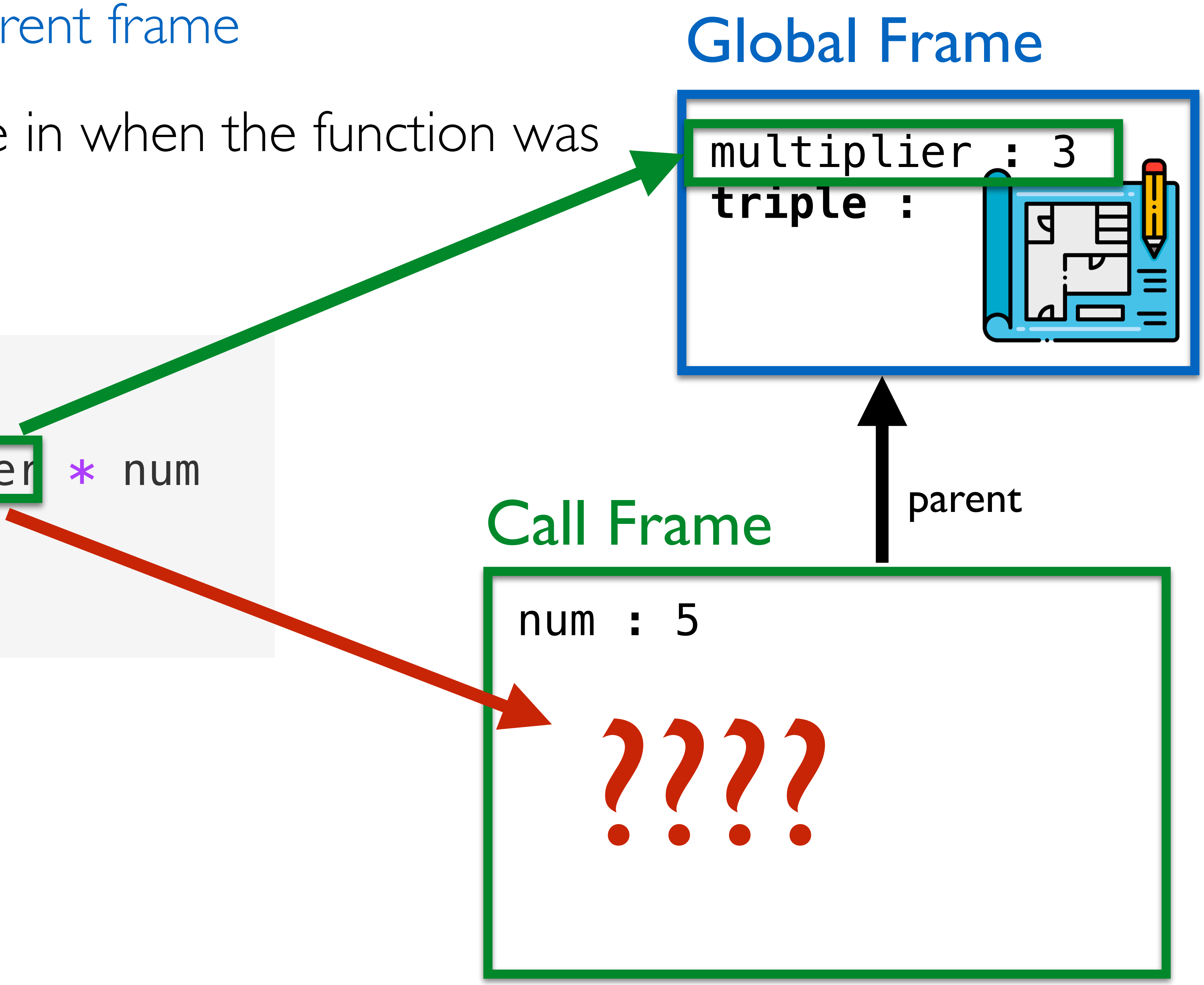
```
num : 5
```

????

Scope: Function Frame Model

- If the variable isn't found in the call frame, then python looks in the parent frame
  - (the frame we were in when the function was called)

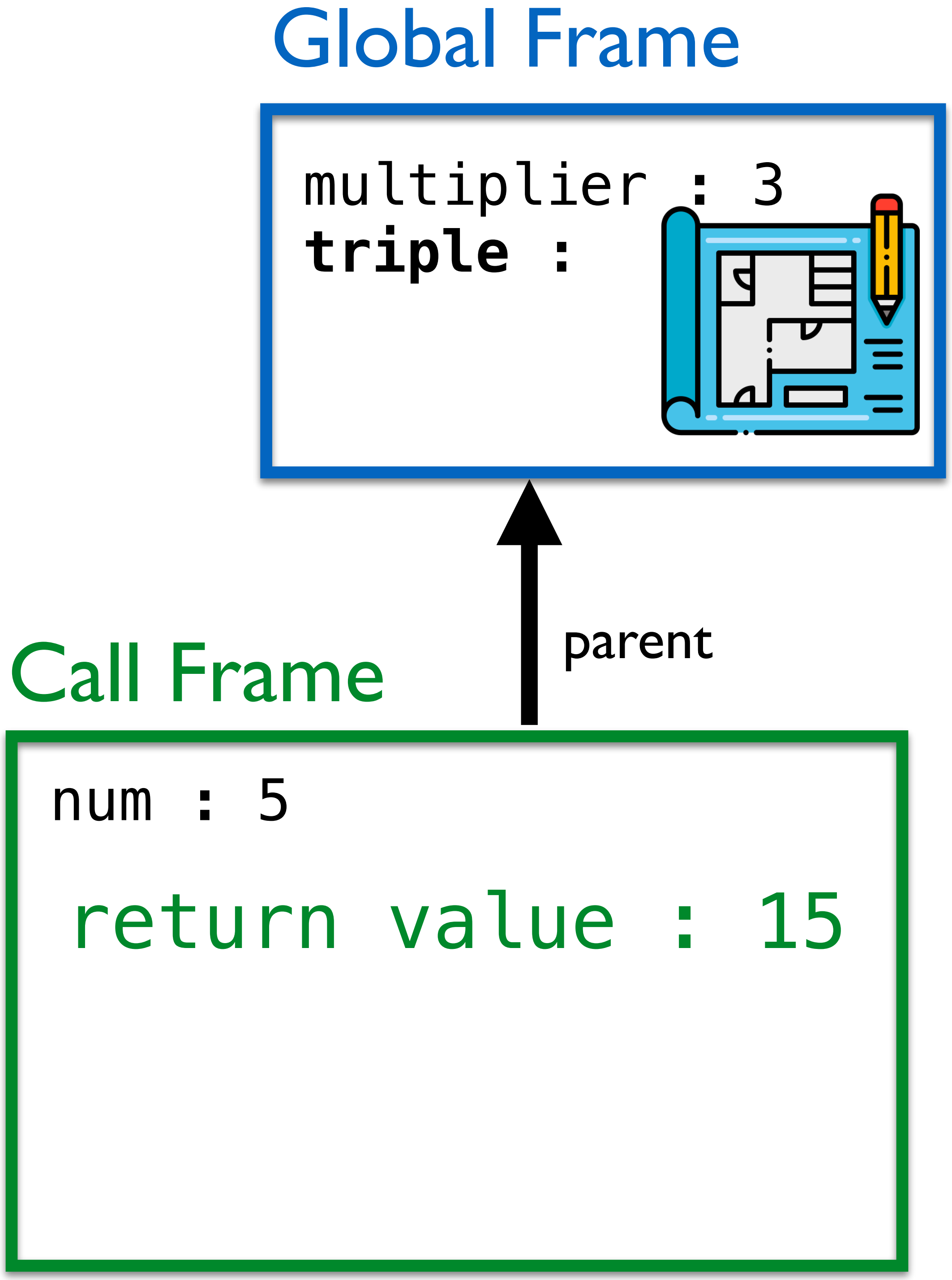
```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Scope: Function Frame Model

- Ultimately, a return value is computed for the function call

```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Global Frame

multiplier : 3
triple :

parent

Call Frame

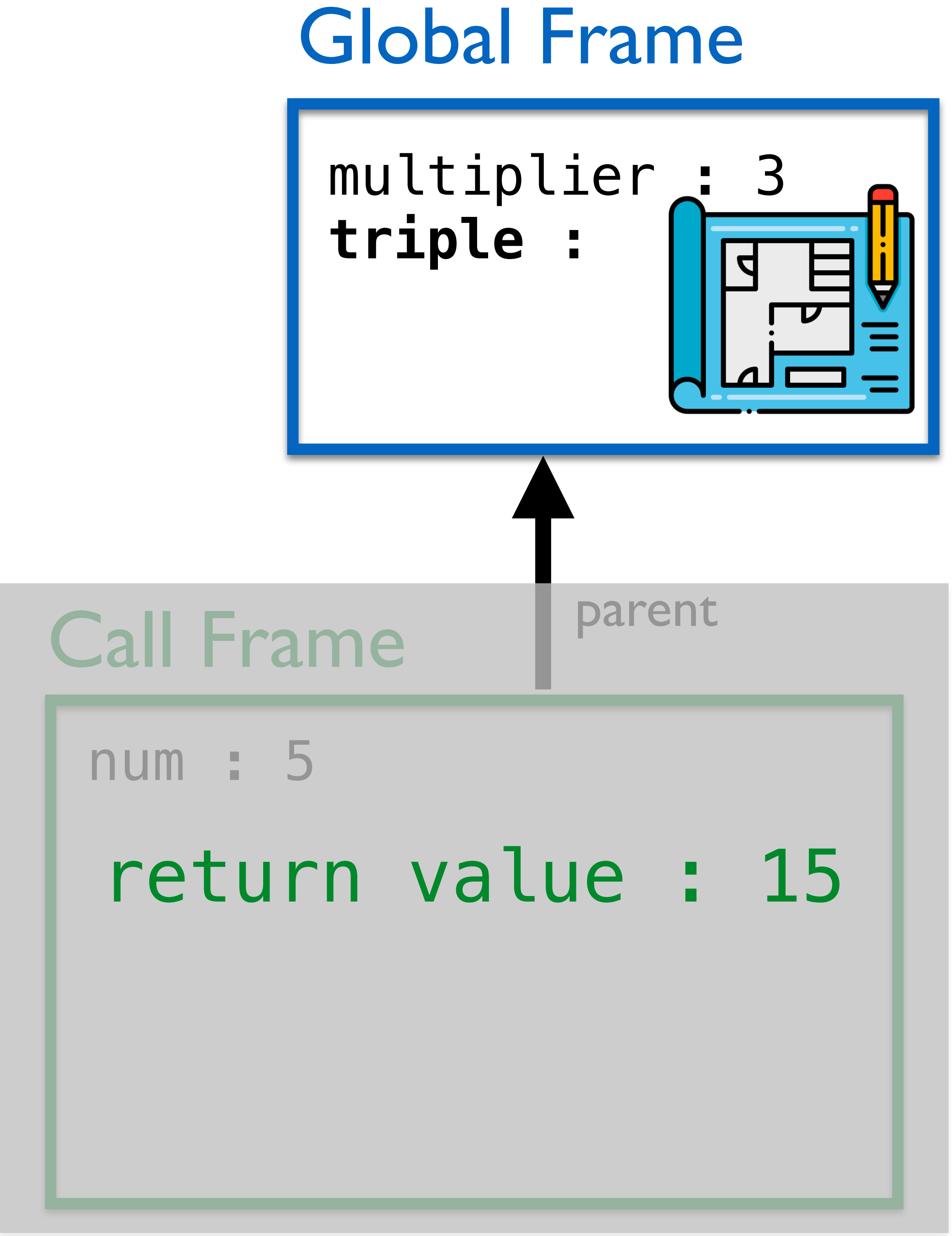
num : 5

return value : 15

Scope: Function Frame Model

- The call frame is destroyed

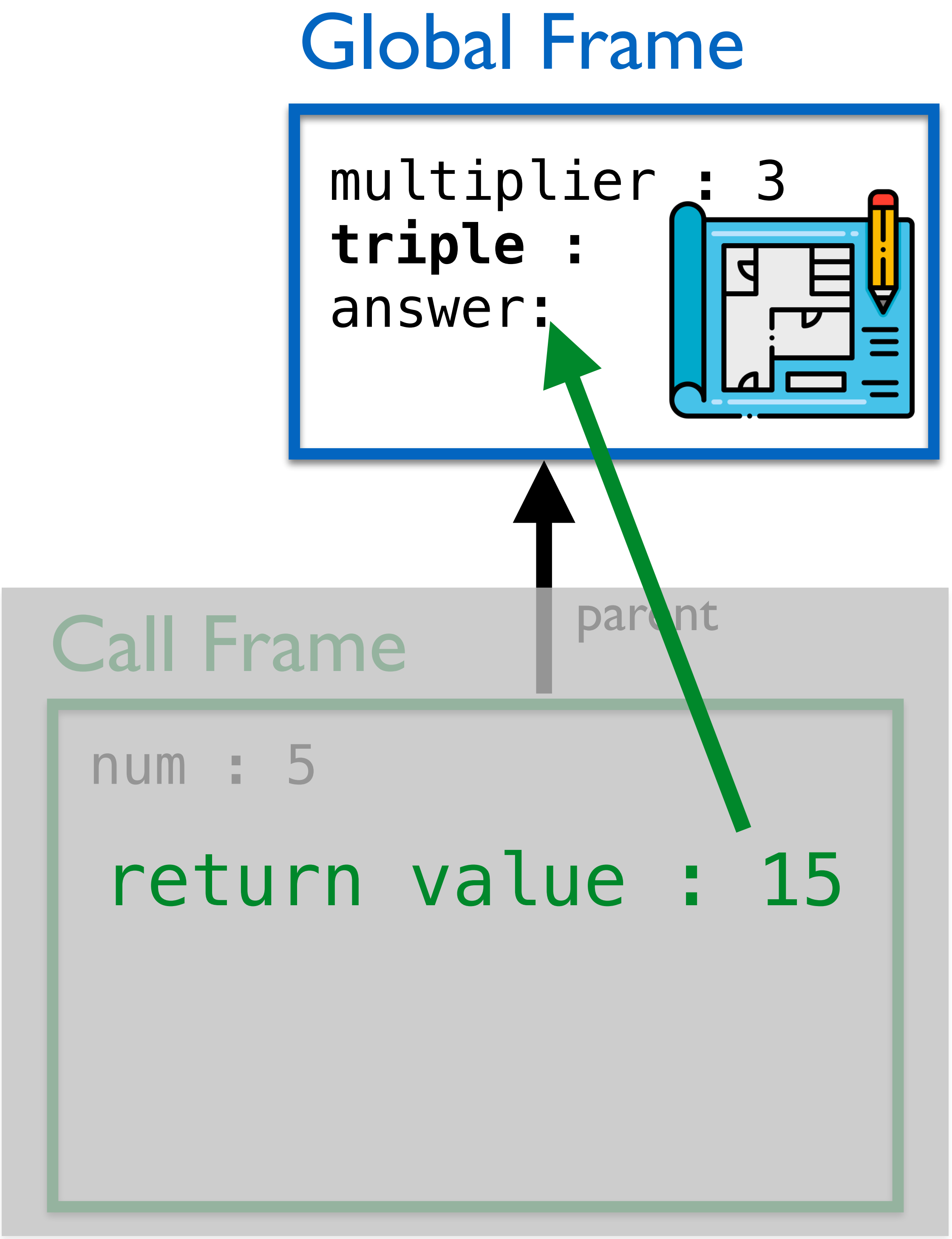
```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Scope: Function Frame Model

- ...and the return value of the function call is assigned to variable `answer` in the global frame

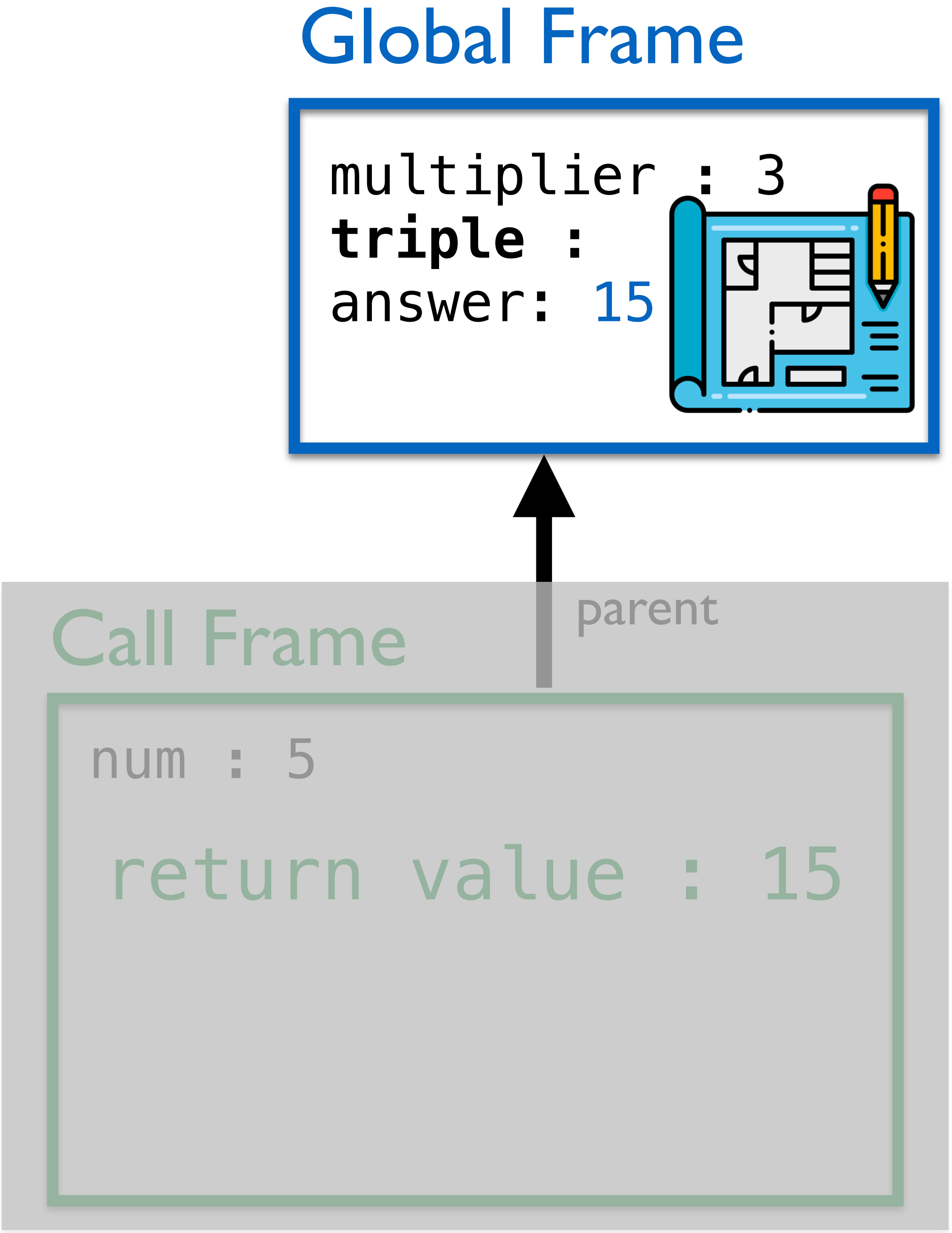
```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Scope: Function Frame Model

- ...and the return value of the function call is assigned to variable `answer` in the global frame

```python
multiplier = 3
def triple(num):
     return multiplier * num
answer = triple(5)
print(answer)
```



Global Frame

multiplier : 3
triple :
answer: 15

parent

Call Frame

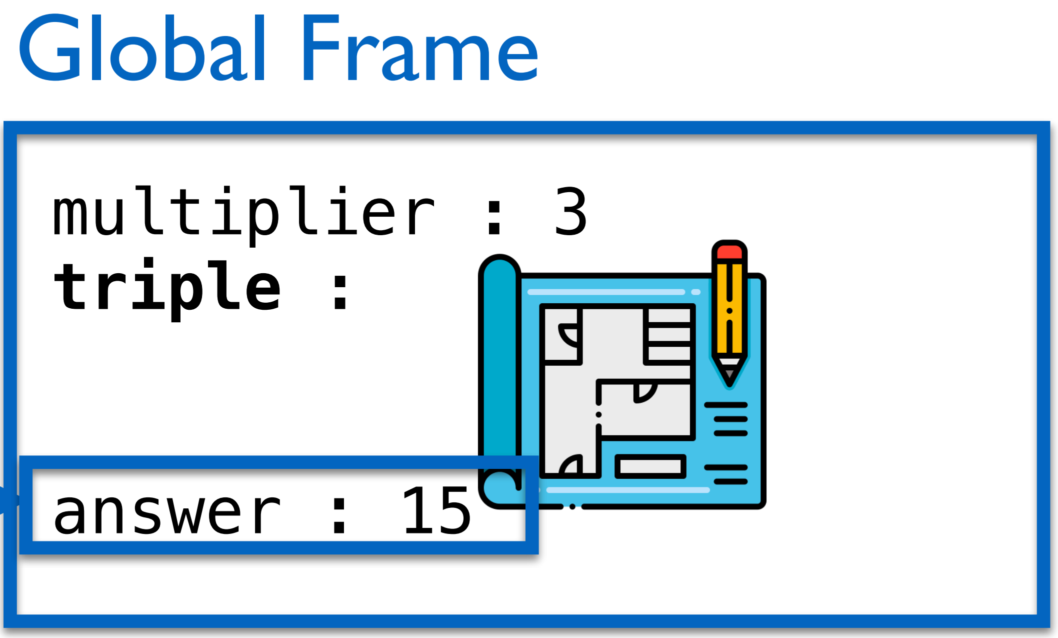
num : 5

return value : 15

Scope: Function Frame Model

- Finally, the value of **`answer`** is looked up in the global frame
- And printed to the screen

```
0 multiplier = 3
1 def triple(num):
      return multiplier * num
2 answer = triple(5)
3 print(answer)
```



Global Frame

```
multiplier : 3
triple :
answer : 15
```

Function Frame Model:
Side-by-Side

Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

below/after function

D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

after function call

Let's use these principles to trace the execution of these two programs

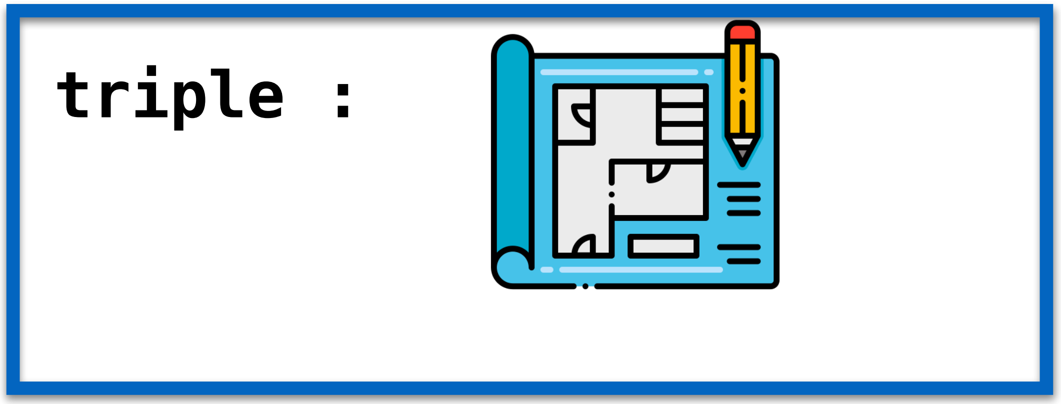
Side-By-Side

Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

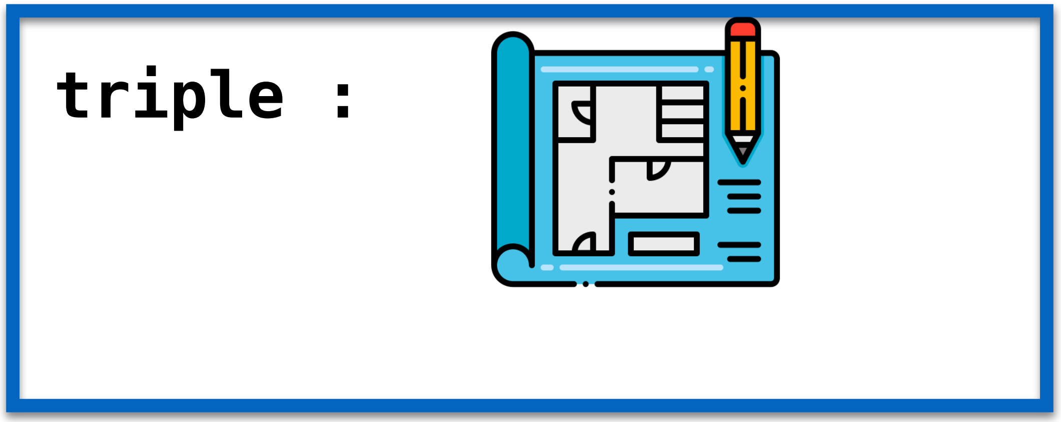
Global Frame



D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

Global Frame

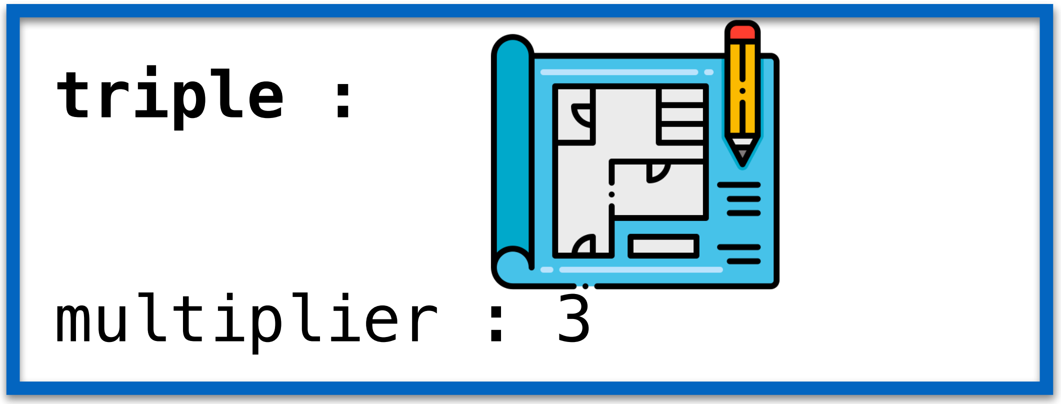


Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

Global Frame



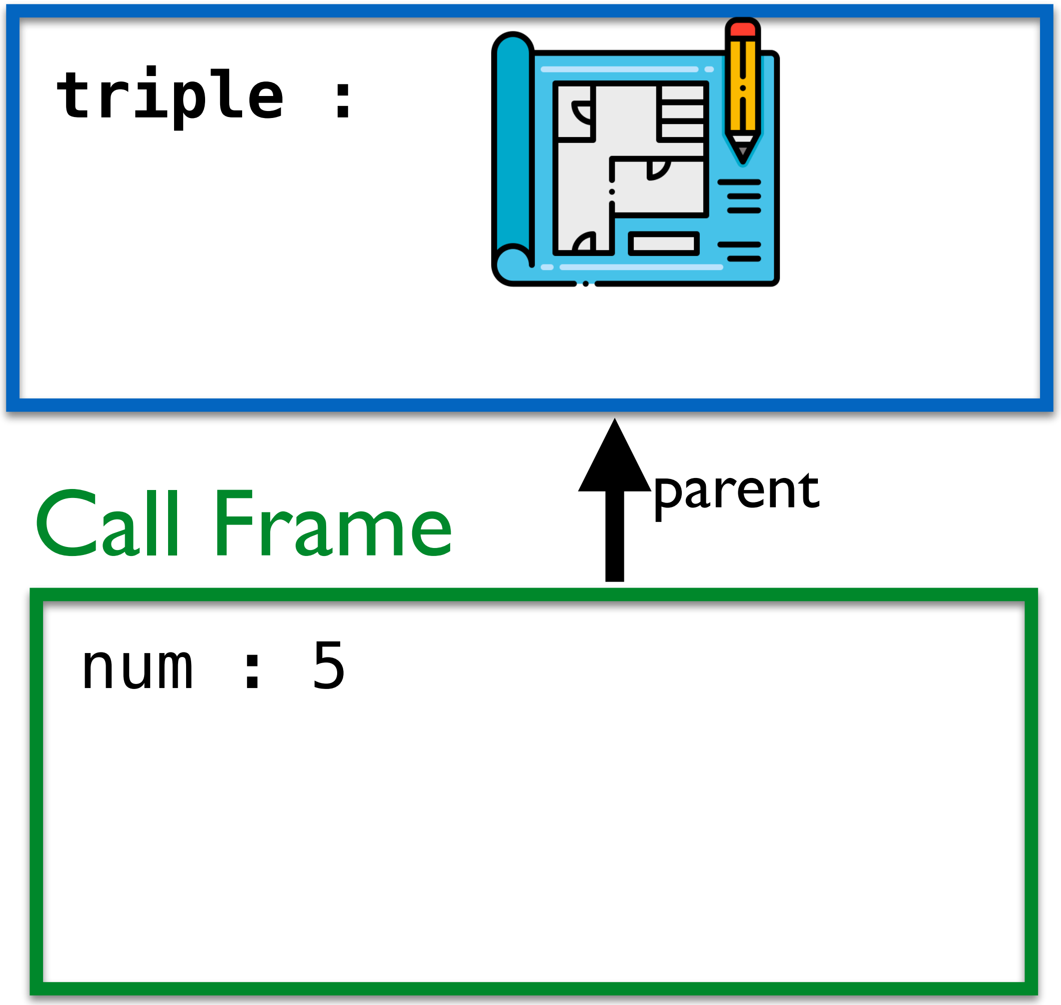
triple :

multiplier : 3

D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

Global Frame



triple :

parent

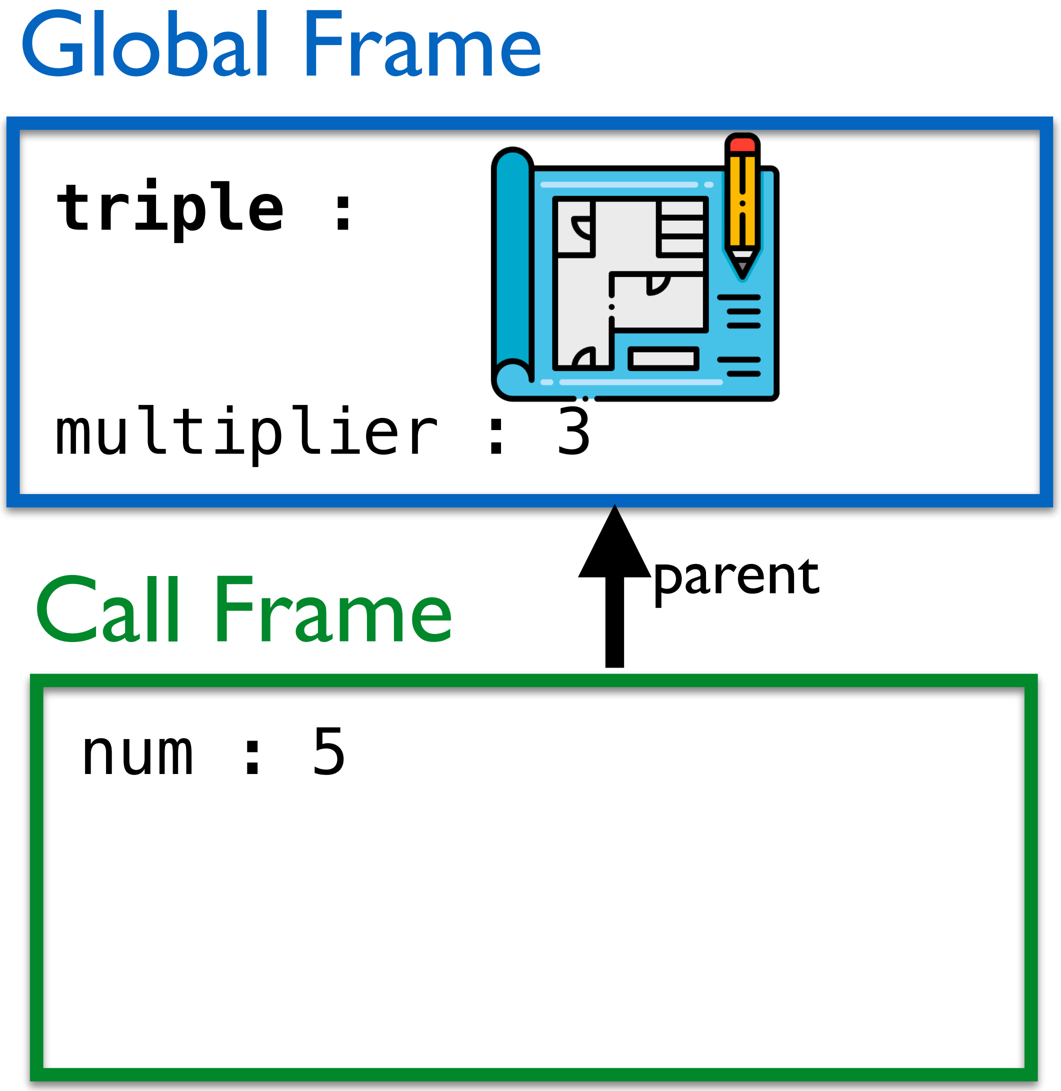
Call Frame

num : 5

Side-by-Side

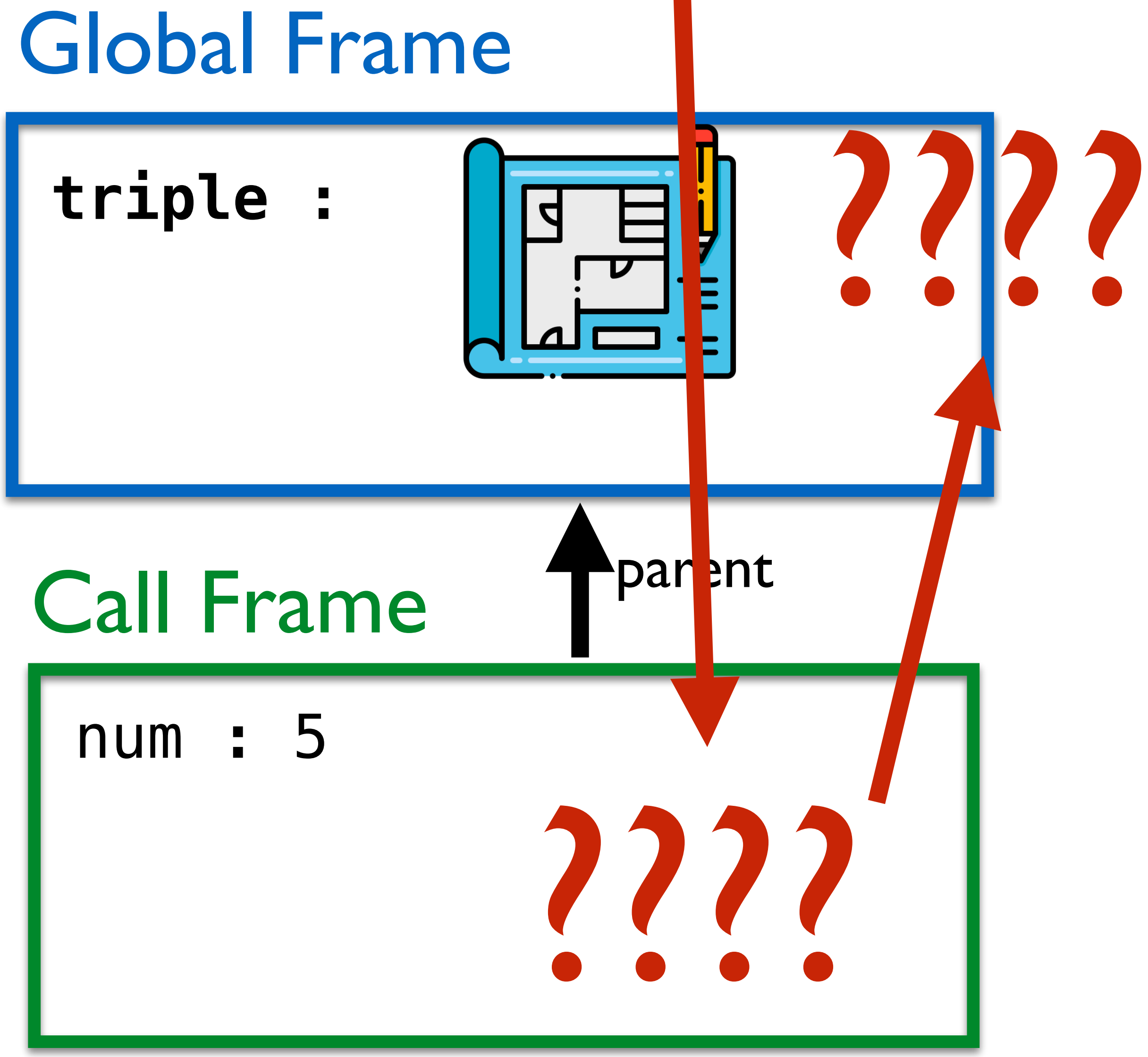
C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```



D

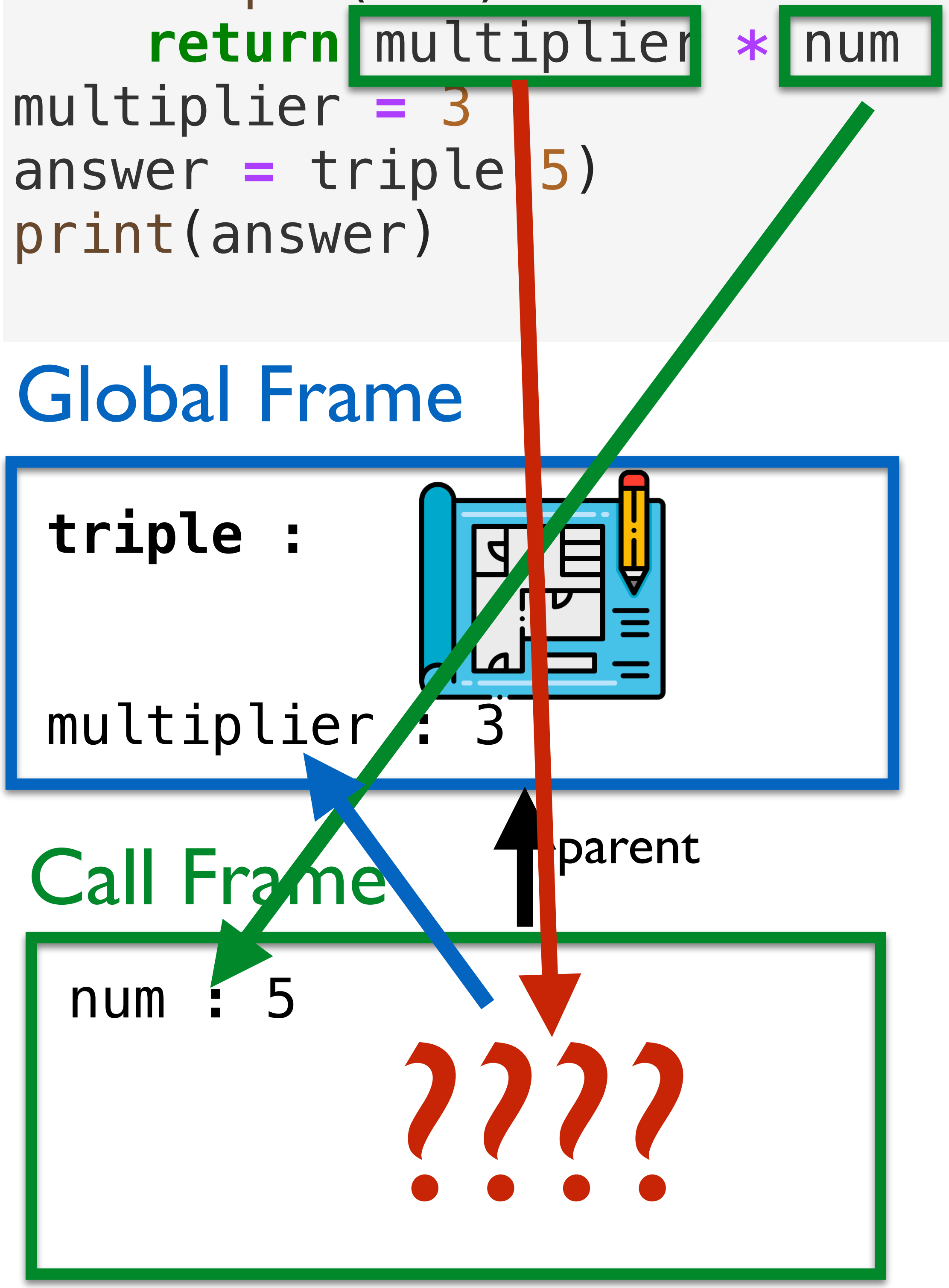
```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```



Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```



Global Frame

```
triple :
multiplier : 3
```

parent

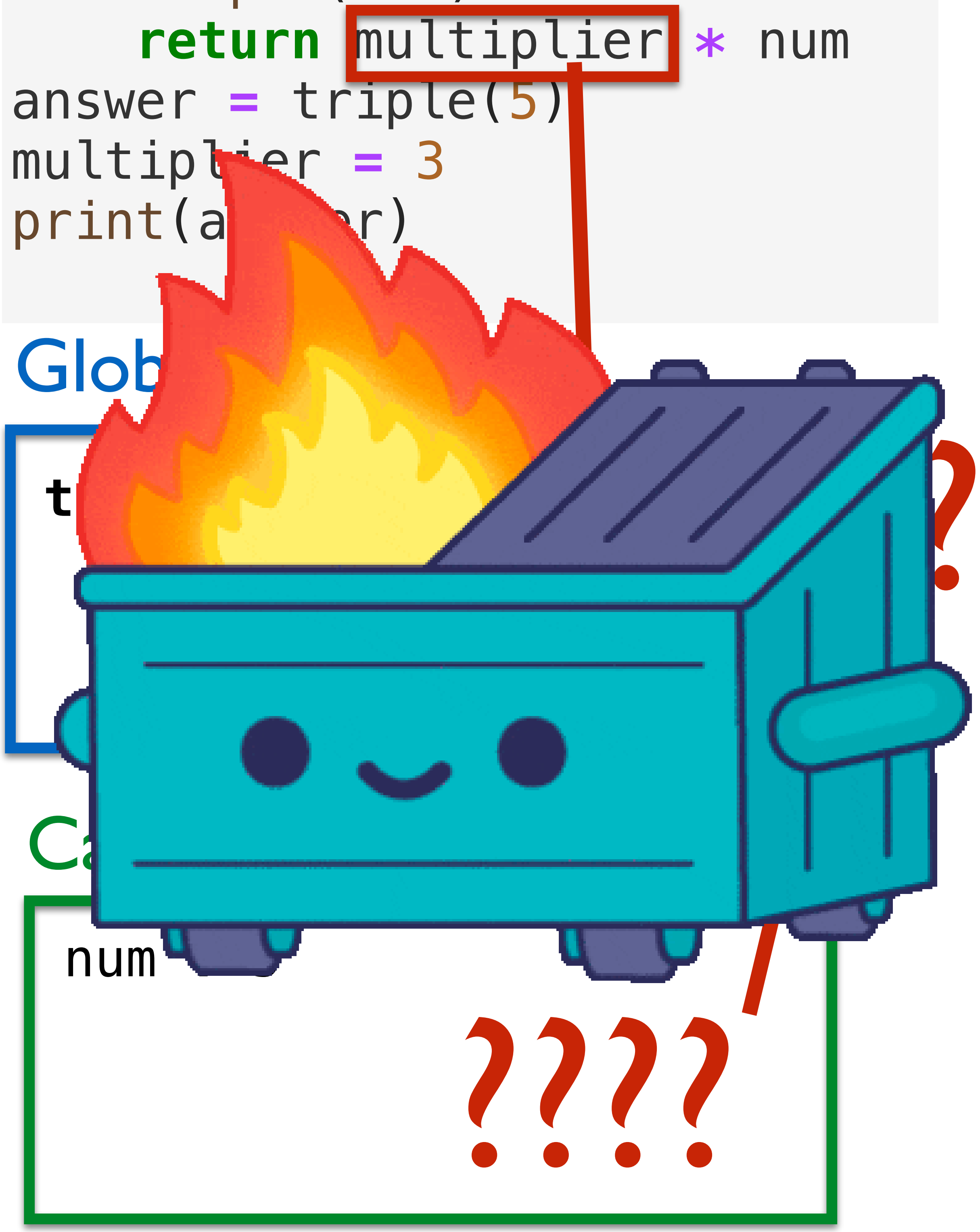
Call Frame

```
num : 5
```

????

D

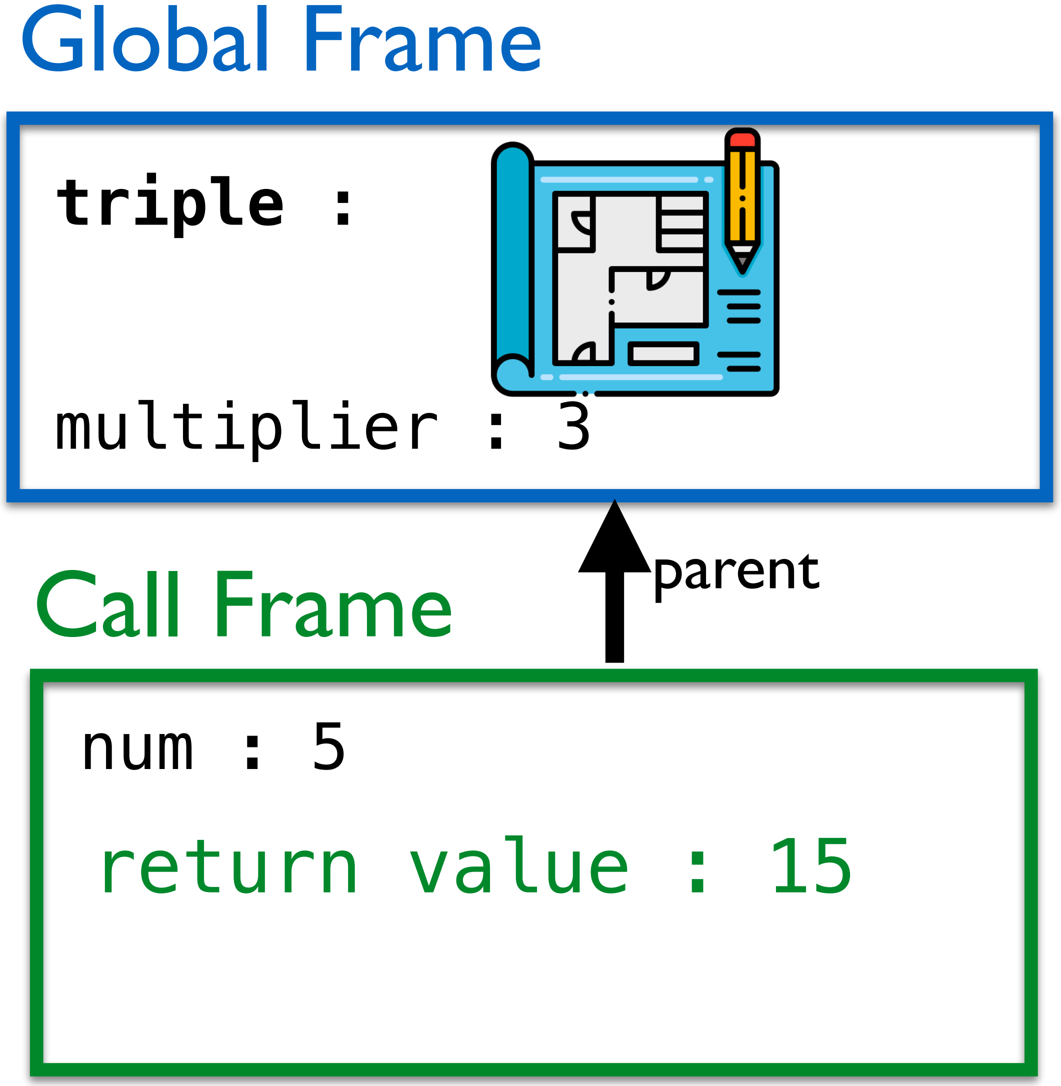
```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```



Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```



Global Frame

triple :

multiplier : 3

parent

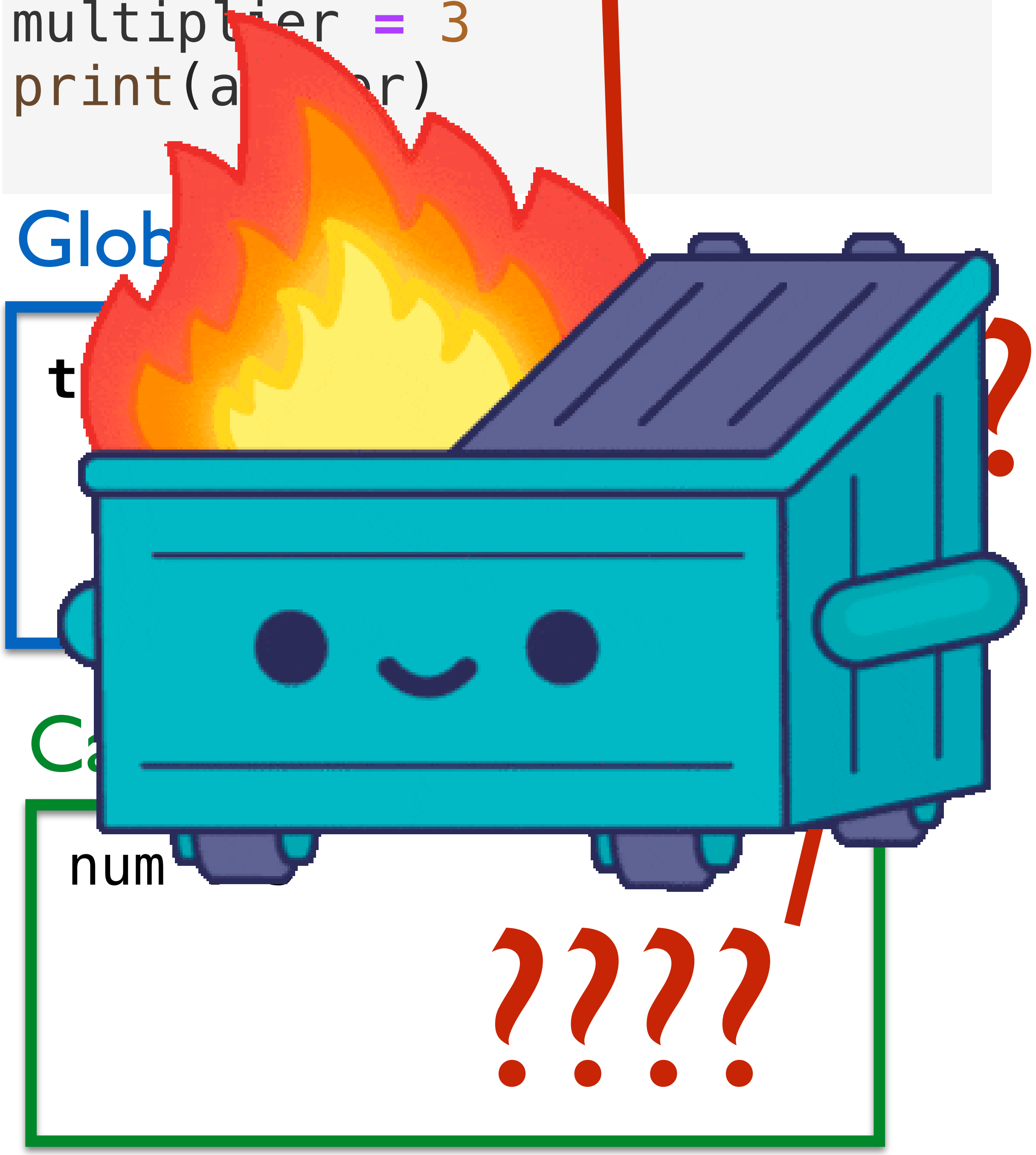
Call Frame

num : 5

return value : 15

D

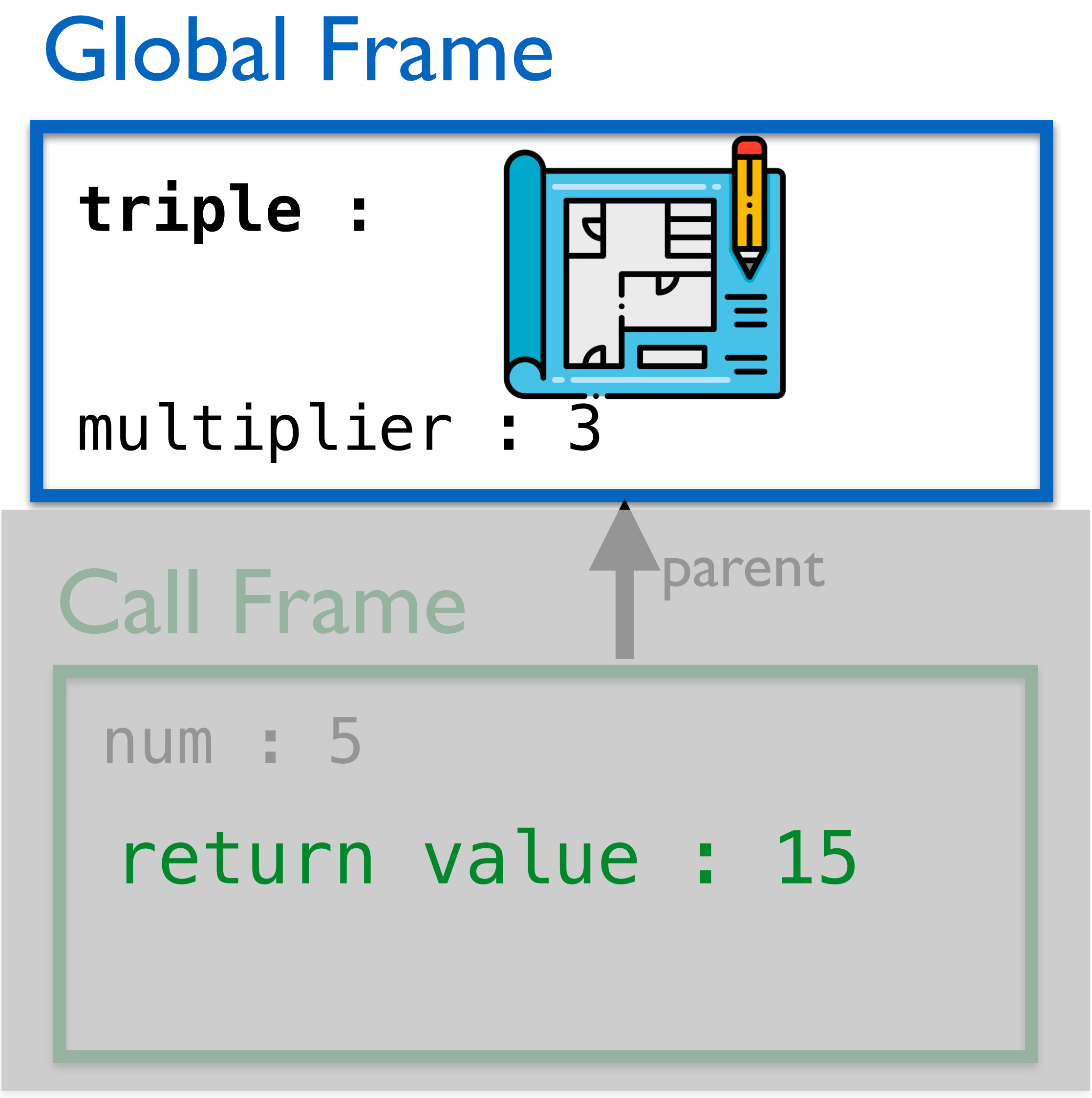
```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```



Side-by-Side

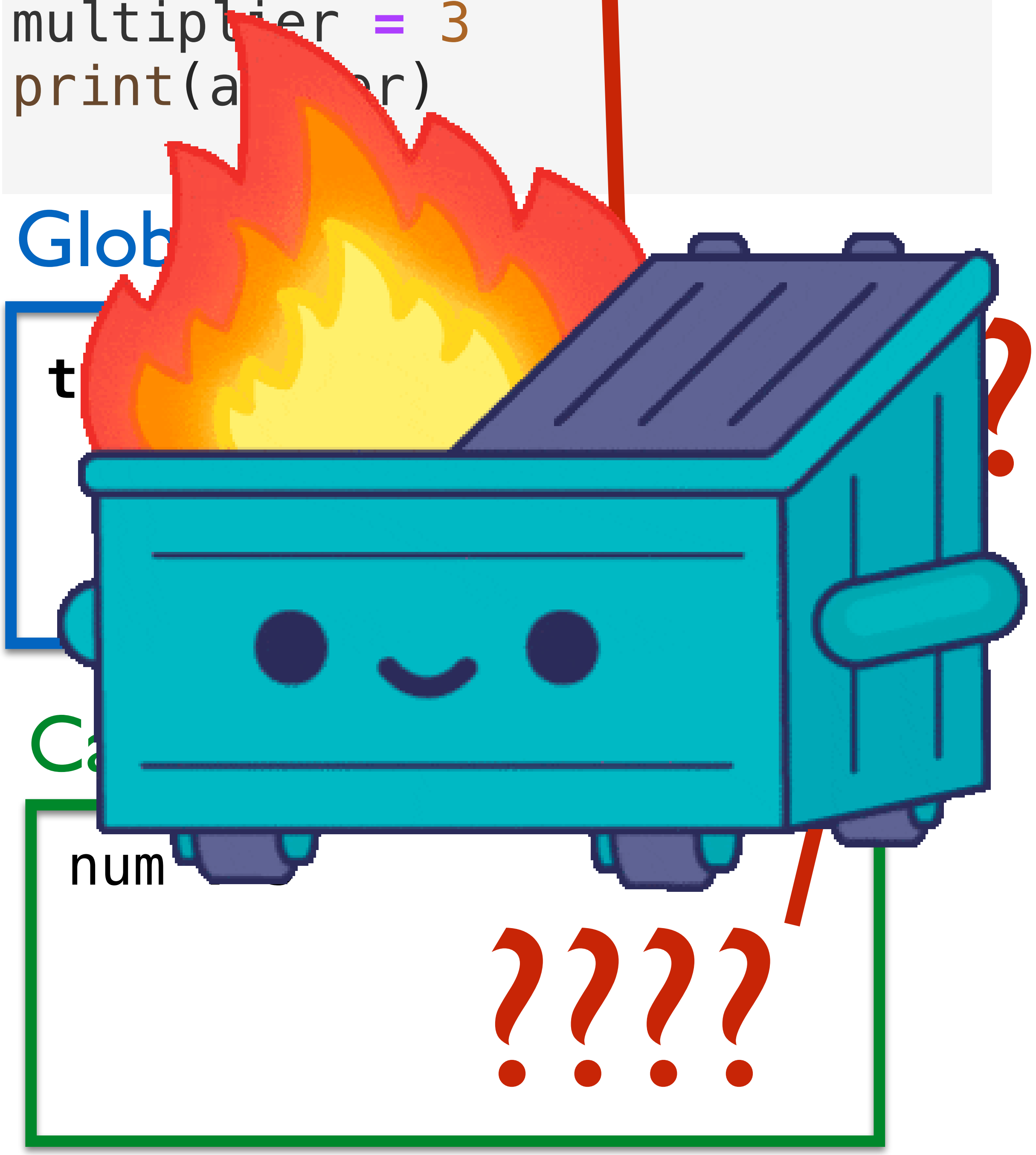
C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```



D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(answer)
```

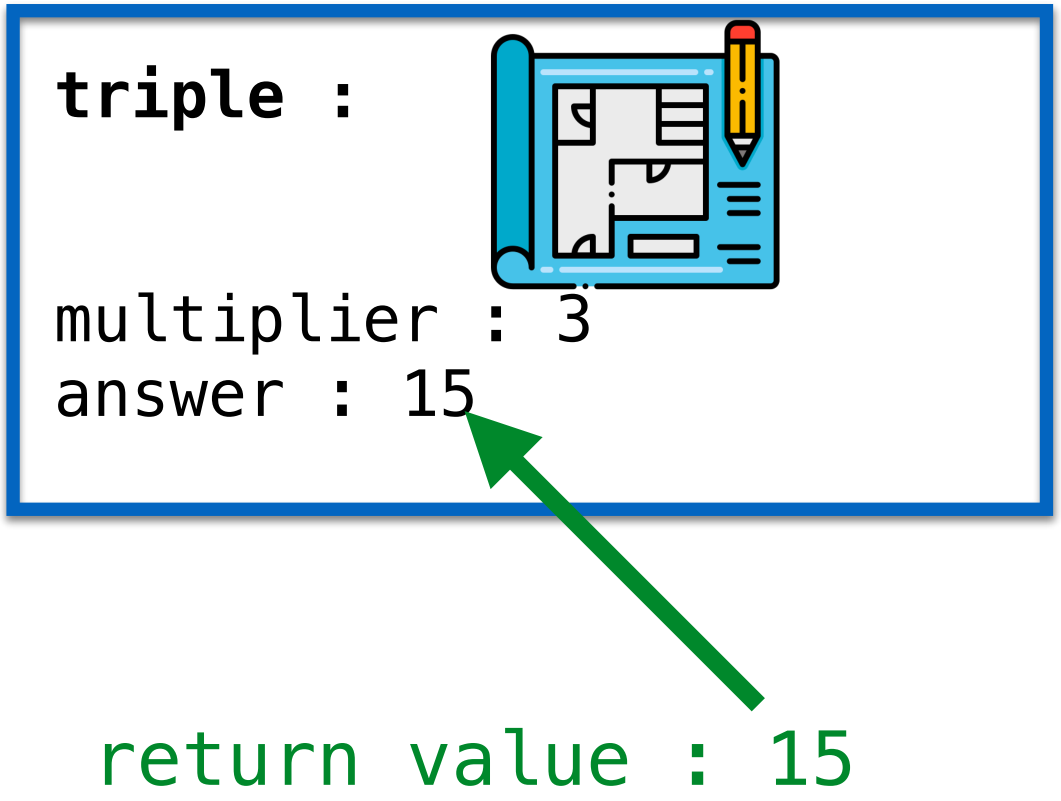


Side-by-Side

C

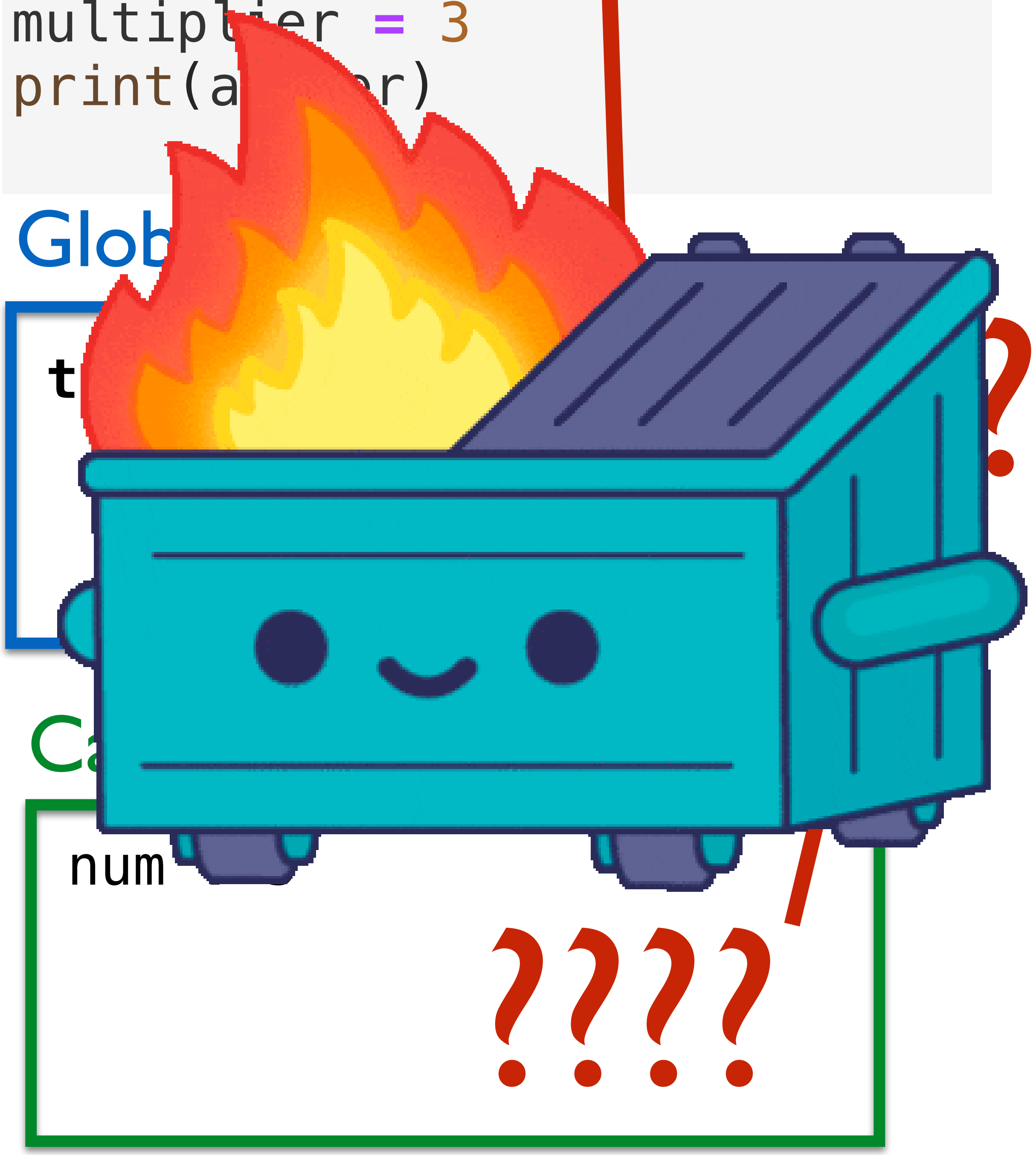
```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

Global Frame



D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(a    r)
```



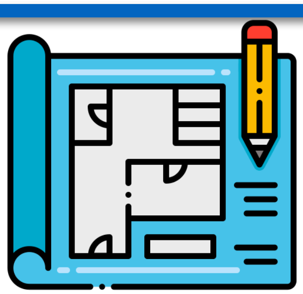
Side-by-Side

C

```
def triple(num):
    return multiplier * num
multiplier = 3
answer = triple(5)
print(answer)
```

Global Frame

```
triple :
```



```
multiplier : 3
answer : 15
```

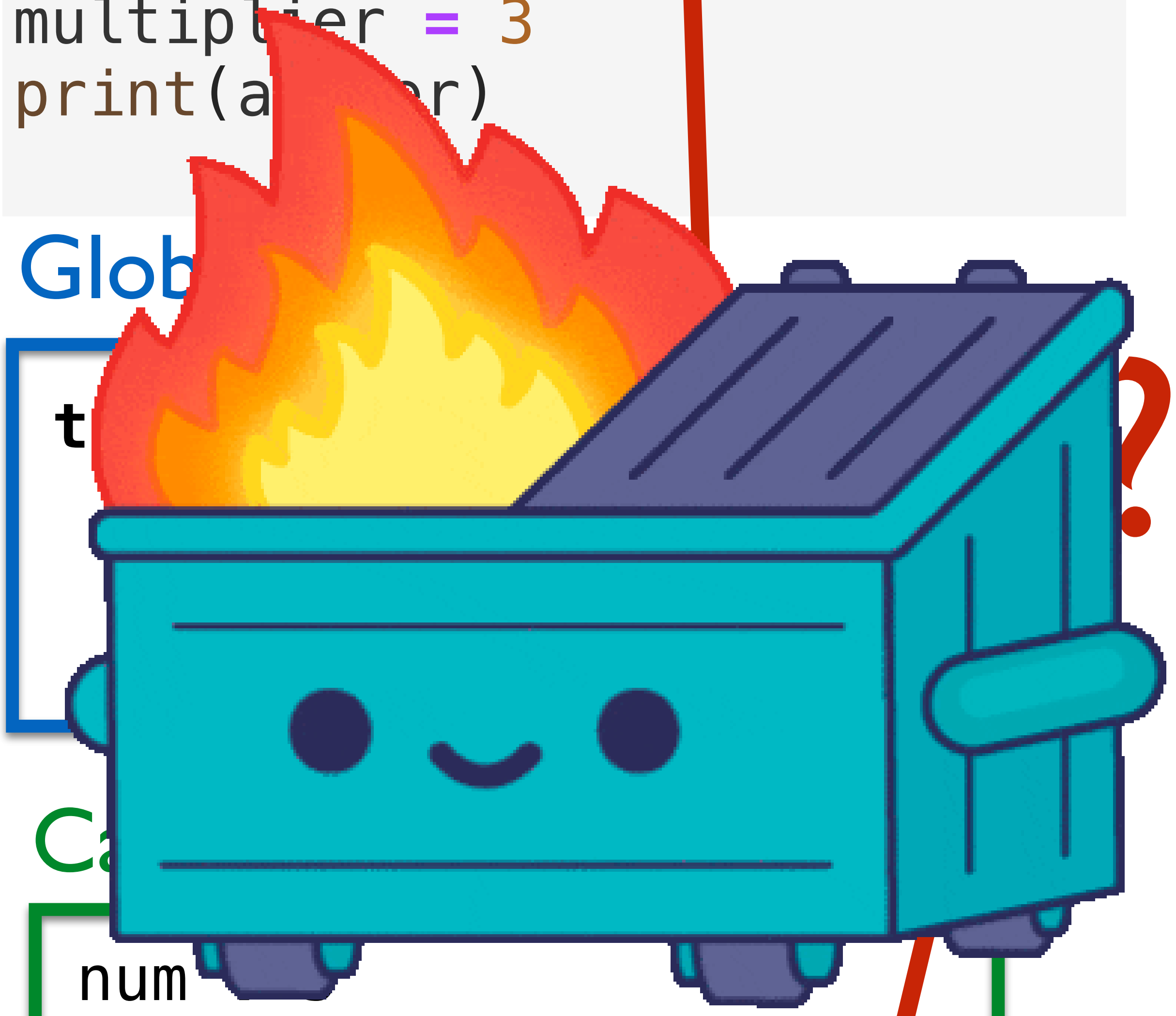
D

```
def triple(num):
    return multiplier * num
answer = triple(5)
multiplier = 3
print(a    r)
```

Glob

t

Ca



num

????

?

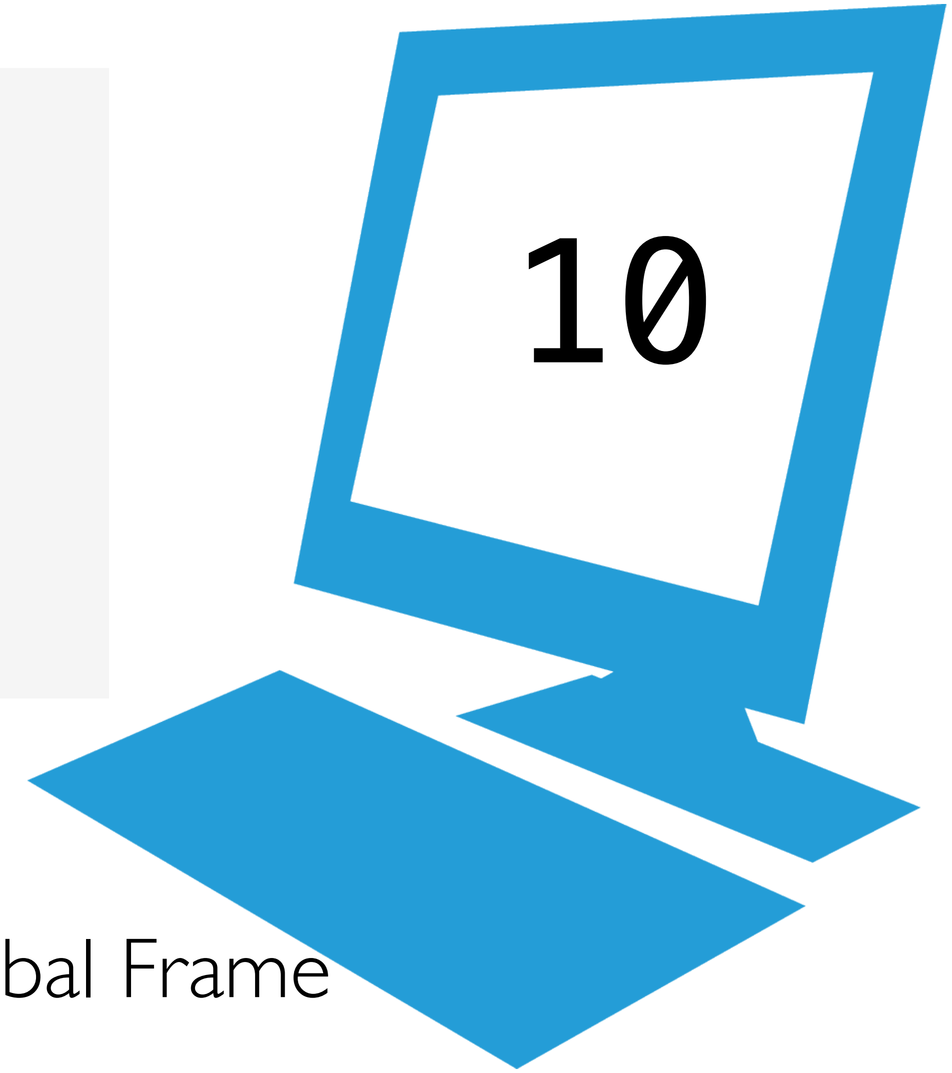
More Examples

What gets printed to the screen?

```
multiplier = 3
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```

What gets printed to the screen?

```python
multiplier = 3
def mystery(num):
    return multiplier * num
multiplier = 2
answer = mystery(5)
print(answer)
```

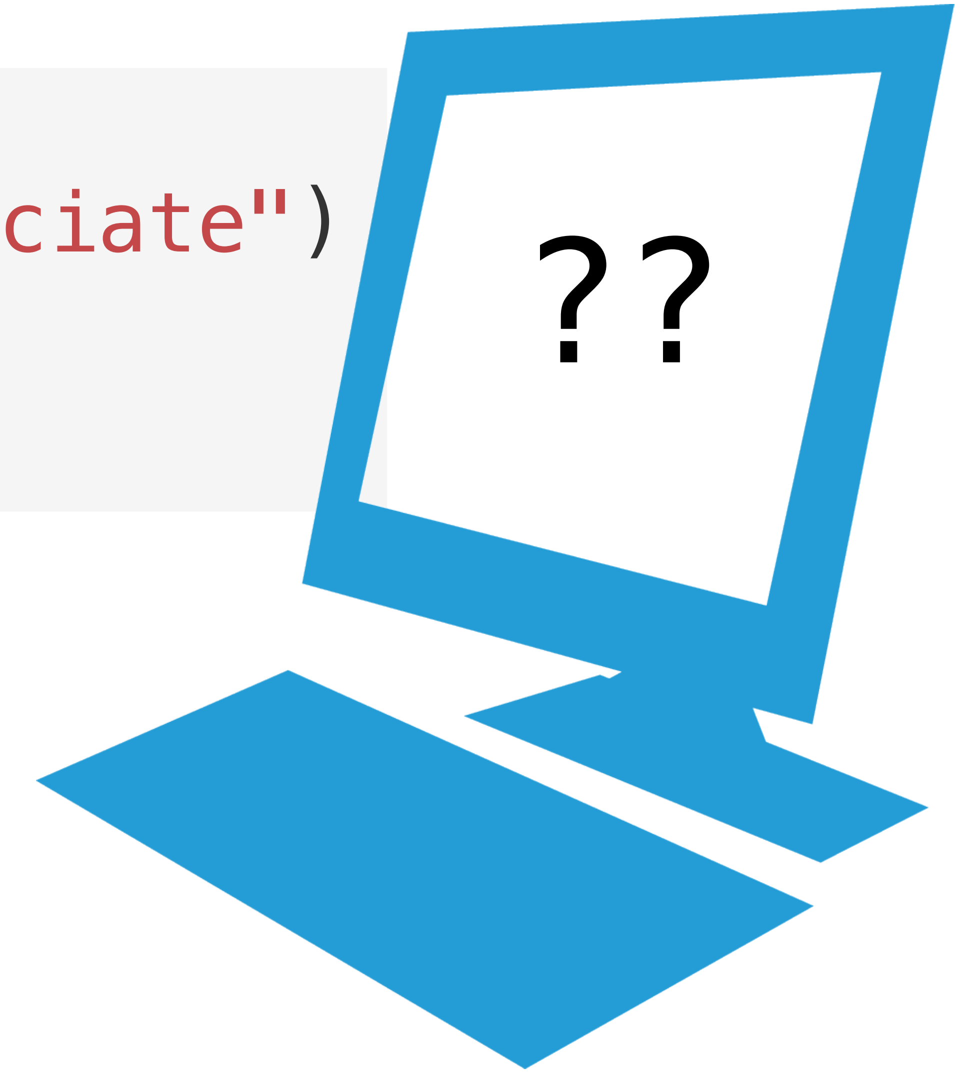


```
10
```

- `multiplier` is recorded as 3 on the Global Frame
- Then the `mystery()` blueprint is recorded on the Global Frame
- Then `multiplier` is re-assigned the value 2 on the Global Frame
- …

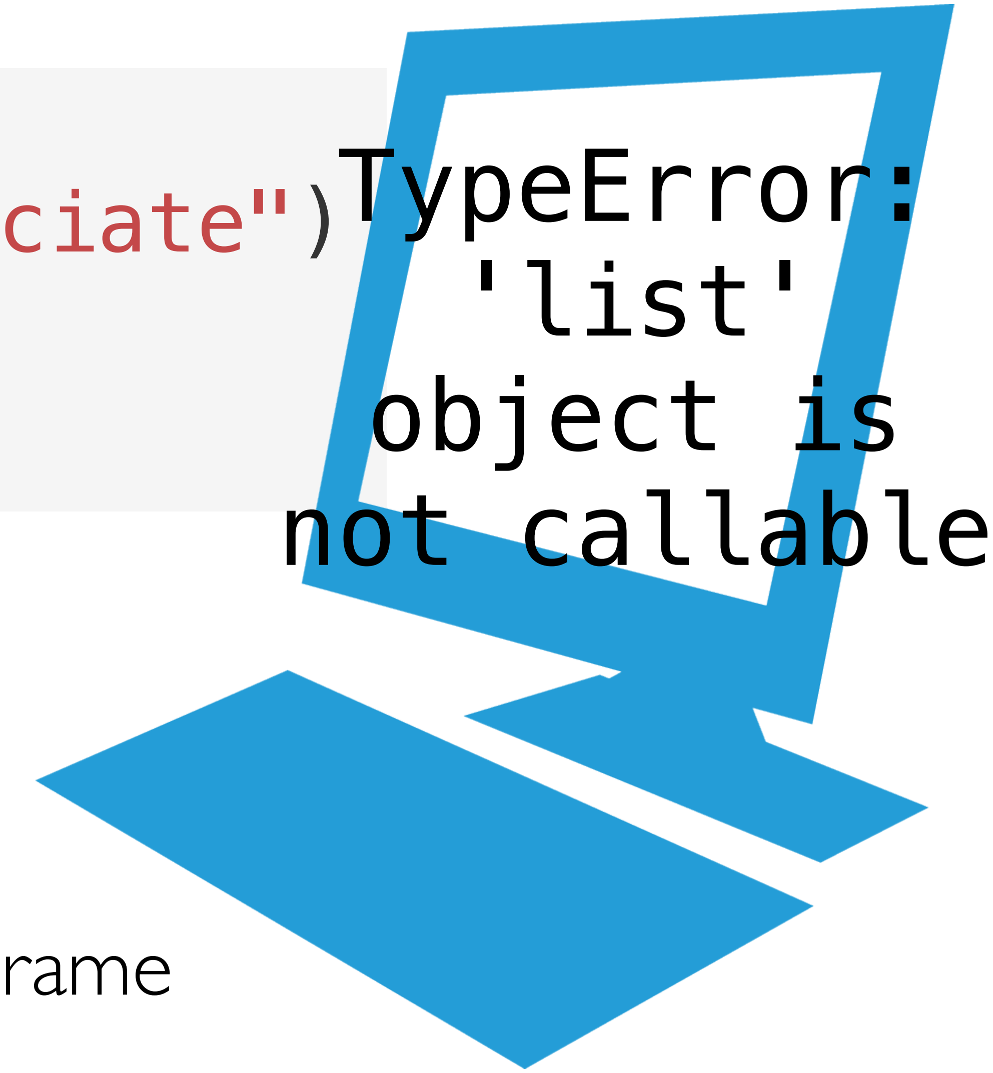
What gets printed to the screen?

```python
list = 2468
list_str = list("whodoweappreciate")
print(list, list_str)
```



What gets printed to the screen?

```
list = 2468
list_str = list("whodoweappreciate")
print(list, list_str)
```

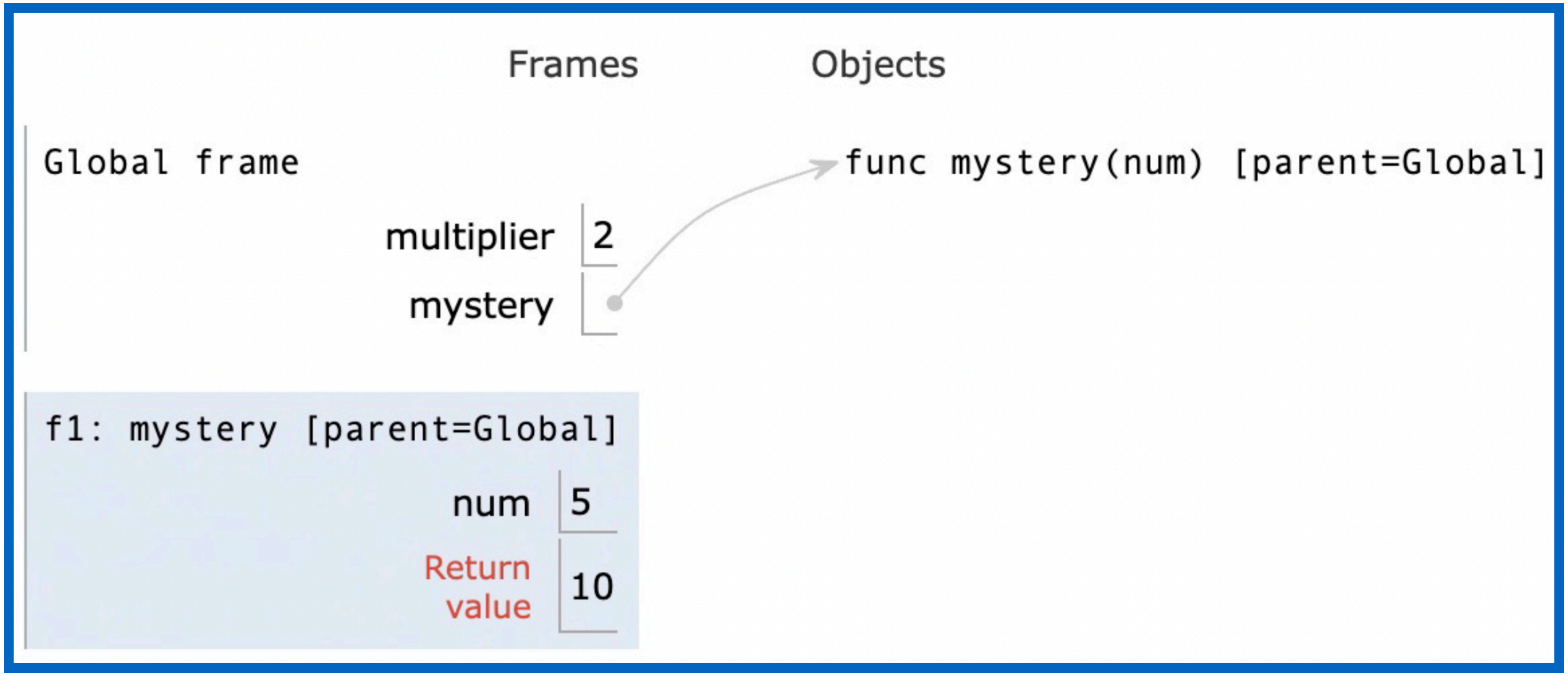


```
TypeError:
 'list'
object is
not callable
```

- `list` is a python keyword, in the Global Frame
- `list = ...` reassigns the value of list in the Global Frame
    - It's no longer the keyword, it's now an integer object
- So you can't call `list(..)` as the built-in list-casting function!
- ...This is why we don't use python keywords as variable names.

Helpful Tool for Learning How python Executes Code

- https://pythontutor.com/cp/composingprograms.html



The end!