

# Visualizing Function Calls

We can explore how Python manages function calls using the Python Visualizer. (See the Resources page.)

In the example below, function `convert_to_seconds` contains a call on `convert_to_minutes`.

```
def convert_to_minutes(num_hours):
    '''(int) -> int
    Return the number of minutes there are in num_hours hours.
    >>> convert_to_minutes(2)
    120
    '''
    result = num_hours * 60
    return result
```

```
def convert_to_seconds(num_hours):
    '''(int) -> int
    Return the number of seconds there are in num_hours hours.
    >>> convert_to_minutes(2)
    7200
    '''
    return convert_to_minutes(num_hours) * 60
```

```
seconds_2 = convert_to_seconds(4)
```

Here is what the memory model looks like just before the return statement inside function `convert_to_minutes` looks like:

The screenshot shows the Python Visualizer interface. On the left, the code is displayed with line numbers 1 through 18. Line 7 is highlighted with a green arrow, indicating it has just executed. Line 8 is highlighted with a red arrow, indicating it is the next line to execute. The code defines two functions: `convert_to_minutes` and `convert_to_seconds`, and then calls `convert_to_seconds(4)` to assign the result to `seconds_2`.

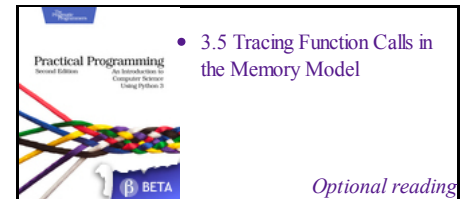
On the right, the memory model is visualized. It shows a stack of frames. The top frame is the 'Global variables' frame, which contains `convert_to_minutes` (id1) and `convert_to_seconds` (id2). Below it is the `convert_to_seconds` frame, which contains `num_hours` (id3) and `result` (id4). The bottom frame is the `convert_to_minutes` frame, which contains `num_hours` (id3) and `result` (id4). The `result` variable in the `convert_to_minutes` frame is highlighted in blue, indicating it is the current focus of the visualization.

Below the code, there is a progress bar and navigation buttons: '<< First', '< Back', 'Step 6 of 8', 'Forward >', and 'Last >>'. A legend at the bottom left indicates that a green arrow points to the line that has just executed, and a red arrow points to the next line to execute.

Note that there are three stack frames on the call stack: the main one, then underneath that a frame for the call on function `convert_to_seconds`, and underneath that the frame for the call on function `convert_to_minutes`.

Here is [a link to the Python Visualizer](#) at this stage of the execution so that you can explore this yourself. **We strongly encourage you to step backward and forward through this program until you understand every step of execution.**

When the return statement is executed, the call on `convert_to_minutes` exits. The bottom stack frame is removed, and execution resumes using the stack frame for `convert_to_seconds`:



```

1 def convert_to_minutes(num_hours):
2     '''(int) -> int
3     Return the number of minutes there are in num
4     >>> convert_to_minutes(2)
5     120
6     '''
7     result = num_hours * 60
8     return result
9
10 def convert_to_seconds(num_hours):
11     '''(int) -> int
12     Return the number of seconds there are in num
13     >>> convert_to_minutes(2)
14     7200
15     '''
16     return convert_to_minutes(num_hours) * 60
17
18 seconds_2 = convert_to_seconds(4)

```

[Edit code](#)

Step 8 of 8

→ line that has just executed  
→ next line to execute

Global variables

convert_to_minutes	id1
convert_to_seconds	id2

Frames

convert_to_seconds	
num_hours	id3
Return value	id5

Objects

id1: function  
convert\_to\_minutes(num\_hours)

id2: function  
convert\_to\_seconds(num\_hours)

id3: int  
4

id5: int  
14400

Jennifer Campbell • Paul Gries  
University of Toronto