

# Functions, Variables, and the Call Stack

## Understanding Scope

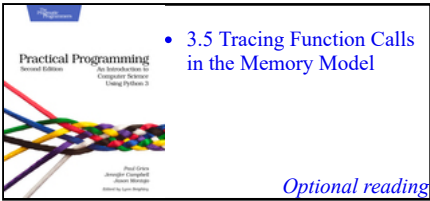
Below is an explanation and review of the example used in the video.

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

def convert_to_seconds(num_hours):
    """ (int) -> int
    Return the number of seconds there are in num_hours hours.
    """
    minutes = convert_to_minutes(num_hours)
    seconds = minutes * 60
    return seconds

seconds = convert_to_seconds(2)
```

Python defines the first two functions, creates objects for them in the heap, and, in the stack frame for the main program, creates variables that refer to those function objects.



```
1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
→ 8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
→ 16 seconds = convert_to_seconds(2)
```

[Edit code](#)

Frames		Objects
Global frame		id1:function convert_t
convert_to_minutes	id1	
convert_to_seconds	id2	id2:function convert_t

<< First

< Back

Step 3 of 10

Forward >

Last >>

- line that has just executed
- next line to execute

After that, it executes the assignment statement on line 16. The right-hand side of the assignment statement is a function call so we evaluate the argument, 2, first. The frame for convert\_to\_seconds will appear on the call stack. The parameter, num\_hours, will refer to the value 2.

```

1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)

```

[Edit code](#)

Step 4 of 10

→ line that has just executed  
→ next line to execute

Frames		Objects
Global frame		id1: function convert_t...
convert_to_minutes	id1	
convert_to_seconds	id2	id2: function convert_t...
convert_to_seconds		
num_hours	id3	id3: int 2

The first statement in function `convert_to_seconds` is an assignment statement. Again, we evaluate the expression on the right-hand side. This is a function call so we evaluate the argument, `num_hours`. This produces the value 2. A stack frame for function `convert_to_minutes` is created on the call stack. Python stores the memory address of 2 in the parameter for `convert_to_minutes`, which also happens to be called `num_hours`.

```

1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)

```

[Edit code](#)

Step 5 of 10

→ line that has just executed  
→ next line to execute

Frames		Objects
Global frame		id1: function convert_t...
convert_to_minutes	id1	
convert_to_seconds	id2	id2: function convert_t...
convert_to_seconds		
num_hours	id3	id3: int 2
convert_to_minutes		
num_hours	id3	

We now see that there are two variables called `num_hours` in the call stack; one is in `convert_to_minutes` and the other is in `convert_to_seconds`.

The next line of code Python executes is `minutes = num_hours * 60`. However, which instance of `num_hours` will be used? Python always uses the variable in the current stack frame. With an assignment statement, if the variable does not exist in the current stack frame, Python creates it. So, once `num_hours * 60` is evaluated,

variable `minutes` is created in the current stack frame.

1def convert\_to\_minutes(num\_hours):  
2 """ (int) -> int  
3 Return the number of minutes there are in num\_hours hours.  
4 """  
5 minutes = num\_hours \* 60  
6 return minutes  
7  
8def convert\_to\_seconds(num\_hours):  
9 """ (int) -> int  
10 Return the number of seconds there are in num\_hours hours.  
11 """  
12 minutes = convert\_to\_minutes(num\_hours)  
13 seconds = minutes \* 60  
14 return seconds  
15  
16 seconds = convert\_to\_seconds(2)

[Edit code](#)

<< First< BackStep 6 of 10Forward >Last >>

→ line that has just executed  
→ next line to execute

Frames

Global frame  
convert\_to\_minutes id1  
convert\_to\_seconds id2  
convert\_to\_seconds  
num\_hours id3  
convert\_to\_minutes  
num\_hours id3  
minutes id4

Objects

id1:function  
convert\_t  
id2:function  
convert\_t  
id3:int  
2  
id4:int  
120

The last line of the function is `return minutes`. Once this statement is complete, Python will return to the frame just underneath the top of the call stack.

1def convert\_to\_minutes(num\_hours):  
2 """ (int) -> int  
3 Return the number of minutes there are in num\_hours hours.  
4 """  
5 minutes = num\_hours \* 60  
6 return minutes  
7  
8def convert\_to\_seconds(num\_hours):  
9 """ (int) -> int  
10 Return the number of seconds there are in num\_hours hours.  
11 """  
12 minutes = convert\_to\_minutes(num\_hours)  
13 seconds = minutes \* 60  
14 return seconds  
15  
16 seconds = convert\_to\_seconds(2)

[Edit code](#)

<< First< BackStep 7 of 10Forward >Last >>

→ line that has just executed  
→ next line to execute

Frames

Global frame  
convert\_to\_minutes id1  
convert\_to\_seconds id2  
convert\_to\_seconds  
num\_hours id3  
convert\_to\_minutes  
num\_hours id3  
minutes id4  
Return value id4

Objects

id1:function  
convert\_t  
id2:function  
convert\_t  
id3:int  
2  
id4:int  
120

So, Python is going to produce the value 120, remove the current stack frame, create a new variable called `minutes` in the stack frame for `convert_to_seconds`, and store the memory address of 120 in that variable.

https://d3c33hcgiewv3.cloudfront.net/\_a12e9792972ebacec647b7aaed264070\_scope.html?Expires=1520380800&Signature=SxUsgFyLfIEX-uwD... 3/5

```
1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)
```

[Edit code](#)

<< First < Back Step 8 of 10 Forward > Last >>

→ line that has just executed

→ next line to execute

Frames

Global frame

convert\_to\_minutes

convert\_to\_seconds

convert\_to\_seconds

num\_hours

minutes

Objects

id1:function  
convert\_t

id2:function  
convert\_t

id3:int  
2

id4:int  
120

Python then executes `seconds = minutes * 60`. Python evaluates the right-hand side, which produces `7200`, and stores the memory address of that value in variable `seconds`. Since this variable does not exist yet, Python creates it in the current stack frame.

```
1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)
```

[Edit code](#)

<< First < Back Step 9 of 10 Forward > Last >>

→ line that has just executed

→ next line to execute

Frames

Global frame

convert\_to\_minutes

convert\_to\_seconds

convert\_to\_seconds

num\_hours

minutes

seconds

Objects

id1:function  
convert\_t

id2:function  
convert\_t

id3:int  
2

id4:int  
120

id5:int  
7200

Next is a return statement. Like we saw above, that is going to return control back to the the main module.

```

1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)

```

[Edit code](#)

Step 10 of 10

→ line that has just executed  
→ next line to execute

Frames

Global frame

convert_to_minutes	id1
convert_to_seconds	id2

convert\_to\_seconds

num_hours	id3
minutes	id4
seconds	id5
Return value	id5

Objects

id1: function	convert_t...
id2: function	convert_t...
id5: int	7200
id3: int	2
id4: int	120

Once the frame for `convert_to_seconds` is removed, the assignment statement on line 16 (which has been paused a long time!) is completed, and a new variable `seconds` is created in the stack frame for the main program.

```

1 def convert_to_minutes(num_hours):
2     """ (int) -> int
3     Return the number of minutes there are in num_hours hours.
4     """
5     minutes = num_hours * 60
6     return minutes
7
8 def convert_to_seconds(num_hours):
9     """ (int) -> int
10    Return the number of seconds there are in num_hours hours.
11    """
12    minutes = convert_to_minutes(num_hours)
13    seconds = minutes * 60
14    return seconds
15
16 seconds = convert_to_seconds(2)

```

[Edit code](#)

Program terminated

→ line that has just executed  
→ next line to execute

Frames

Global frame

convert_to_minutes	id1
convert_to_seconds	id2
seconds	id5

Objects

id1: function	convert_t...
id2: function	convert_t...
id5: int	7200

## Notes and assignment and return statements

### Assignment statement and computer memory.

variable = expression

If a variable does not exist in the current stack frame, Python creates it.

### Return statement and computer memory.

return expression

In addition to evaluating the expression and yielding its value, `return` also erases the stack frame on top of the call stack.