1 Administrative Topics

• None today.

2 Lists in functions

Because lists live outside the table and the values are just arrows to the lists, we need to look at memory when we call functions that examine or manipulate lists.

Let’s start with a very simple example:

```python
def mean(nums):
    sum = 0.0
    for num in nums:
        sum += num
    return sum/len(nums)
```

```python
a = [1, 2, 3, 4]
b = mean(a)
print(b)
```

There is one list in memory. The main function has a name `a` that points to it and, while `mean` is executing, it has a name `nums` that points to it.

One nice consequence of lists living outside the table is that we don’t use too much memory – sending in copies of arrows is much more space-efficient.
than sending in a copy of the list.

Another consequence is that functions can modify lists that were created by the main code. This can be good and bad. It is efficient, but it means that the values in a main table’s lists can be changed, which means that you really need to step through all the code to figure out what is in a main table.

```python
def double(nums):
    for i, num in enumerate(nums):
        nums[i] = num*2
a = [1, 2, 3]
b = double(a)
print a
print b
```

There is only one list, its entries are doubled.

Notice this is different from a function that creates a new list, with its entries double those of the old list.

```python
def double_copy(nums):
    doubled = []
    for num in nums:
        doubled.append(num*2)
    return doubled
a = [1, 2, 3]
b = double_copy(a)
print a
print b
```

There are two lists in this example. The quiz and homework are modeled after this second example.

We should model our code after `double` (i.e. edit the original list) when when don’t need to keep around the original copy of the list. We should model our code after `double_copy` when we do need to keep around an original copy of the list. Notice that with `double_copy`, we must return the new list and with `double` we should not return anything.
3 Functions calling functions

So far, we have had main code call functions, but we can also have functions call functions.

```python
1 def square( x ):
2     return x**x

4 def sum_of_squares( x, y, z );
5     x2 = square( x )
6     y2 = square( y )
7     z2 = square( z )
8     return x2 + y2 + z2

10 print sum_of_squares( 2, 3, 1 )
```

We have up to three symbol tables at a time.

When Python executes line 1, it adds `square` to the main table (note that `<fcn>` really belongs outside the table, but I am drawing it inside because my typesetting software makes it difficult to draw it outside).

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td><code>&lt;fcn&gt;</code></td>
</tr>
</tbody>
</table>

When Python executes line 4, it adds `sum_of_squares` to the main table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td><code>&lt;fcn&gt;</code></td>
</tr>
<tr>
<td>sum_of_squares</td>
<td><code>&lt;fcn&gt;</code></td>
</tr>
</tbody>
</table>

When Python executes line 10, it calls `sum_of_squares`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td><code>&lt;fcn&gt;</code></td>
</tr>
<tr>
<td>sum_of_squares</td>
<td><code>&lt;fcn&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2&lt;int&gt;</td>
</tr>
<tr>
<td>y</td>
<td>3&lt;int&gt;</td>
</tr>
<tr>
<td>z</td>
<td>1&lt;int&gt;</td>
</tr>
</tbody>
</table>

print `sum_of_squares( 2, 1, 3 )`
Then, one line 5, `sum_of_squares` calls `square`, which associates the value 2 with its parameter `x`.

```
main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>&lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>&lt;fcn&gt;</td>
</tr>
</tbody>
</table>

**print** `sum_of_squares( 2, 1, 3 )`
```

The value is returned, and the `square` symbol table is deleted: Then, on line 5, `sum_of_squares` calls `square`, which associates the value 2 with its parameter `x`.

```
main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>&lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>&lt;fcn&gt;</td>
</tr>
</tbody>
</table>

**print** `sum_of_squares( 2, 1, 3 )`
```

The assignment statement from line 5 is complete and the `sum_of_squares` symbol table is updated:
main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>&lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>&lt;fcn&gt;</td>
</tr>
</tbody>
</table>

print sum_of_squares( 2, 1, 3 )

The line 6 is executed. It results in a new square symbol table:

main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>&lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>&lt;fcn&gt;</td>
</tr>
</tbody>
</table>

print sum_of_squares( 2, 1, 3 )

The square function returns and the assignment statement from line 6 is complete and the sum_of_squares symbol table is updated:

main

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>&lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>&lt;fcn&gt;</td>
</tr>
</tbody>
</table>

print sum_of_squares( 2, 1, 3 )

Line 7 causes a similar change:
The square function returns and the assignment statement from line 7 is complete and the sum_of_squares symbol table is updated:

```
main

Name  Value
square → <fcn>
sum_of_squares → <fcn>

print sum_of_squares(2, 1, 3 )
z2 = square(z1)

return x*x 1*1
```

Line 8 sums the squares

```
main

Name  Value
square → <fcn>
sum_of_squares → <fcn>

print sum_of_squares(2, 1, 3 )
return x2 + y2 + z2 4+9+1 14
```

Line 8 returns that sum
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>square</td>
<td>→ &lt;fcn&gt;</td>
</tr>
<tr>
<td>sum_of_squares</td>
<td>→ &lt;fcn&gt;</td>
</tr>
</tbody>
</table>

print sum_of_squares(2, 1, 3)

and 14 is printed.