17.1. Introduction: Nested Data and Nested Iteration

17.1.1. Lists with Complex Items

The lists we have seen so far have had numbers or strings as items. We've snuck in a few more complex items, but without ever explicitly discussing what it meant to have more complex items.

In fact, the items in a list can be any type of python object. For example, we can have a list of lists.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
2 print(nested1[0])
3 print(len(nested1))
4 nested1.append(['i'])
5 print("-------")
6 for L in nested1:
7     print(L)
8
```

```
['a', 'b', 'c']
3
-------
['a', 'b', 'c']
['d', 'e']
['f', 'g', 'h']
['i']
```

Activity: 1 -- ActiveCode (ac17_1_1)

Line 2 prints out the first item from the list that `nested1` is bound to. That item is itself a list, so it prints out with square brackets. It has length 3, which prints out on line 3. Line 4 adds a new item to `nested1`. It is a list with one element, 'i' (it a list with one element, it's not just the string 'i').

Codelens gives a you a reference diagram, a visual display of the contents of nested1.

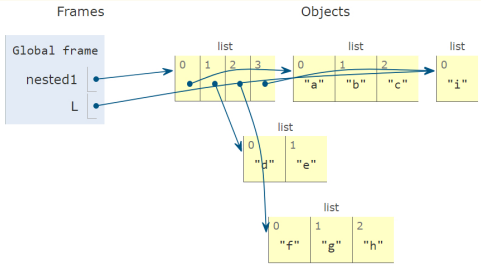
Python 3.3

```
1 nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
2 print(nested1[0])
3 print(len(nested1))
4 nested1.append(['i'])
5 for L in nested1:
6     print(L)
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

```
['a', 'b', 'c']
3
['a', 'b', 'c']
['d', 'e']
['f', 'g', 'h']
['i']
```

Activity: 2 -- CodeLens: (clens_1_1)

When you get to step 4 of the execution, take a look at the object that variable nested1 points to. It is a list of three items, numbered 0, 1, and 2. The item in slot 1 is small enough that it is shown right there as a list containing items "d" and "e". The item in slot 0 didn't quite fit, so it is shown in the figure as a pointer to another separate list; same thing for the item in slot 2, the list `['f', 'g', 'h']`.

There's no special meaning to whether the list is shown embedded or with a pointer to it: that's just CodeLens making the best use of space that it can. In fact, if you go on to step 5, you'll see that, with the addition of a fourth item, the list ['i'], CodeLens has chosen to show all four lists embedded in the top-level list.

With a nested list, you can make complex expressions to get or set a value in a sub-list.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h']]
2 y = nested1[1]
3 print(y)
4 print(y[0])
5
6 print([10, 20, 30][1])
7 print(nested1[1][0])
8
```

```
['d', 'e']
d
20
d
```

Activity: 3 -- ActiveCode (ac17_1_2)

Lines 1-4 above probably look pretty natural to you. Line 5 illustrates the left to right processing of expressions. `nested1[1]` evaluates to the second inner list, so `nested1[1][1]` evaluates to its second element, `'e'`. Line 6 is just a reminder that you index into a literal list, one that is written out, the same way as you can index into a list referred to by a variable. `[10, 20, 30]` creates a list. `[1]` indexes into that list, pulling out the second item, 20.

Just as with a function call where the return value can be thought of as replacing the text of the function call in an expression, you can evaluate an expression like that in line 7 from left to right. Because the value of `nested1[1]` is the list `['d', 'e']`, `nested1[1][0]` is the same as `['d', 'e'][0]`. So line 7 is equivalent to lines 2 and 4; it is a simpler way of pulling out the first item from the second list.

At first, expressions like that on line 7 may look foreign. They will soon feel more natural, and you will end up using them a lot. Once you are comfortable with them, the only time you will write code like lines 2-4 is when you aren't quite sure what your data's structure is, and so you need to incrementally write and debug your code. Often, you will start by writing code like lines 2-4, then, once you're sure it's working, replace it with something like line 7.

You can change values in such lists in the usual ways. You can even use complex expressions to change values. Consider the following

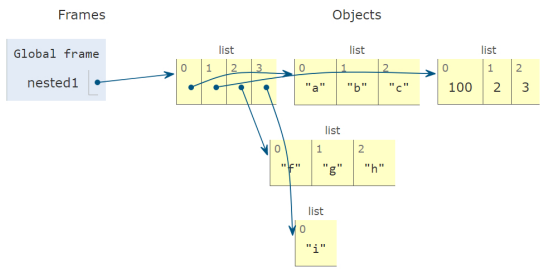
Python 3.3

```
1 nested1 = [['a', 'b', 'c'],['d', 'e'],['f', 'g', 'h'], ['i']]
2 nested1[1] = [1, 2, 3]
3 nested1[1][0] = 100
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



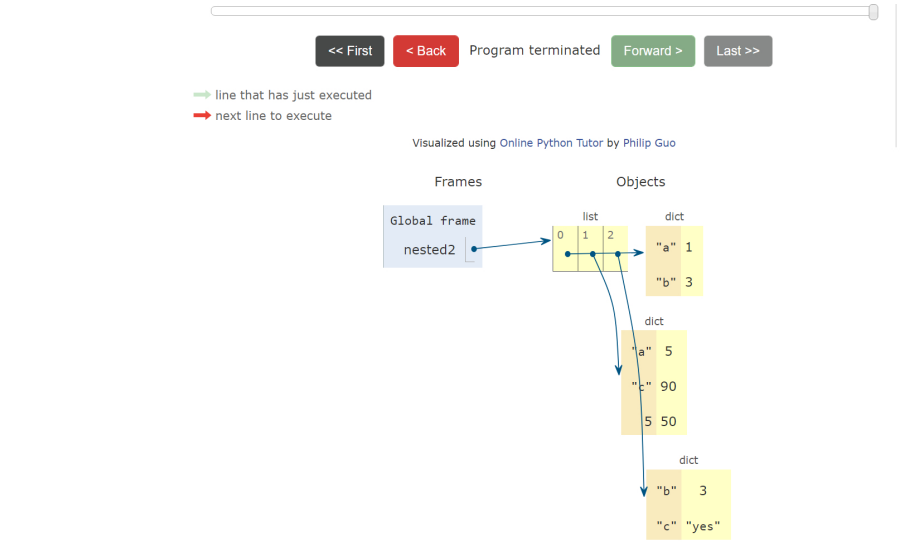
Program output:

Activity: 4 -- CodeLens: (clens_1_2)

The complex items in a list do not have to be lists. They can be tuples or dictionaries. The items in a list do not all have to be the same type, but you will drive yourself crazy if you have lists of objects of varying types. Save yourself some headaches and don't do that. Here's a list of dictionaries and some operations on them. Take a look at its visual representation in codelens.

Python 3.3

```
1 nested2 = [{'a': 1, 'b': 3}, {'a': 5, 'c': 90, 5: 50}, {'b': 3, 'c': "yes"}]
```



Program output:

Activity: 5 -- CodeLens: (clens_1_3)

Try practicing some operations to get or set values in a list of dictionaries.

Save & Run | Show Feedback | Show Code | Show CodeLens

Activity: 6 -- ActiveCode (ac17_1_3)

You can even have a list of functions (!).

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 def square(x):
2     return x*x
3
4 L = [square, abs, lambda x: x+1]
5
6 print("****names****")
7 for f in L:
8     print(f)
9
10 print("****call each of them****")
11 for f in L:
12     print(f(-2))
13
14 print("****just the first one in the list****")
15 print(L[0])
```

```
****names****
<function square>
<built-in function abs>
<function <lambda>>
****call each of them****
4
2
-1
****just the first one in the list****
<function square>
9
```

Activity: 7 -- ActiveCode (ac17_1_4)

Here, L is a list with three items. All those items are functions. The first is the function square that is defined on lines 1 and 2. The second is the built-in python function abs. The third is an anonymous function that returns one more than its input.

In the first for loop, we do not call the functions, we just output their printed representations. The output <function square> confirms that square truly is a function object. For some reason, in our online environment, it's not able to produce a nice printed representation of the built-in function abs, so it just outputs <unknown>

In the second for loop, we call each of the functions, passing in the value -2 each time and printing

whatever value the function returns.

The last two lines just emphasize that there's nothing special about lists of functions. They follow all the same rules for how python treats any other list. Because L[0] picks out the function square, L[0](3) calls the function square, passing it the parameter 3.

Step through it in Codelens if that's not all clear to you yet.

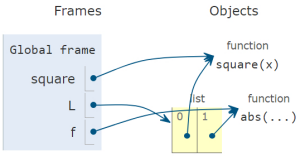
Python 3.3

```
1  def square(x):
2      return x*x
3
4  L = [square, abs]
5
6  print("****names****")
7  for f in L:
8      print(f)
9
10 print("****call each of them****")
11 for f in L:
12     print(f(-2))
13
14 print("****just the first one in the list****")
15 print(L[0])
16 print(L[0](3))
```

<< First | < Back | Program terminated | Forward > | Last >>

line that has just executed
next line to execute

Visualized using Online Python Tutor by Philip Guo



Program output:

```
****names****
<function square at 0x7fa6fc270d90>
<built-in function abs>
****call each of them****
4
2
****just the first one in the list****
<function square at 0x7fa6fc270d90>
9
```

Activity: 8 -- CodeLens: (clens_1_4)

Check Your Understanding

1. Below, we have provided a list of lists. Use indexing to assign the element 'horse' to the variable name `idx1`.

Save & Run | 5/14/2021, 8:33:57 PM - 2 of 2 | Show in CodeLens

```
1
2 animals = [['cat', 'dog', 'mouse'], ['horse', 'cow', 'goat'], ['cheetah', 'giraffe'
3
4 idx1 = animals[1][0]
5 print(idx1)
6
```

```
horse
```

Activity: 9 -- ActiveCode (ac17_1_5)

Result	Actual Value	Expected Value	Notes
Pass	'horse'	'horse'	Testing that idx1 was assigned correctly.

You passed: 100.0% of the tests

2. Using indexing, retrieve the string 'willow' from the list and assign that to the variable `plant`.

Save & Run | 5/14/2021, 8:35:00 PM - 2 of 2 | Show in CodeLens

```
1
2 data = ['bagel', 'cream cheese', 'breakfast', 'grits', 'eggs', 'bacon', [34, 9, 73,
3 plant = data[7][0][0]
4 print(plant)
5
```

```
willow
```

Activity: 10 -- ActiveCode (ac17_1_6)

Result	Actual Value	Expected Value	Notes
Pass	'willow'	'willow'	Testing that plant has the correct value.

You passed: 100.0% of the tests

You have attempted 10 of 10 activities on this page

17. Nested Data and Nested Iteration">
17.2. Nested Dictionaries">
ted Data and Nested Iteration">
✔ Completed. Well Done!
17.2. Nested Dictionaries">Next Section - 17.2. Nested Dictionaries

| Back to top