

Copying and slicing lists

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 16, Spring 2021

Friday March 19

Project 5 overview

- Create an animated scene using Zelle (any theme).
- Create 2+ compound shapes (like the snowman).
- Animate 1+ of your compound shapes (like the snowman; motion OR color change).
- Like snowman, each compound shape needs three functions: `shapeName_init`, `shapeName_test`, `shapeName_animate` (for your animated shapes only).
- Test each shape as you go (i.e. call `shapeName_test` in `if __name__ == '__main__':`).
- Build an animated scene using all your shapes in `main` function, which should create a canvas, call your `shapeName_init` functions, draw them to canvas, have animation loop, etc.
- Make a video to show me and your family and friends!

Summary: Deep vs. Shallow Copy

- Deep copy is a true clone.
 - New, independent symbol table.
 - `b = a.clone()`. Changes to b don't affect a. Changes to a don't affect b.
- Shallow copy is a reference (nickname) for the same data.
 - For example, Robert and Bob refer to the SAME PERSON!
 - Bob is NOT a clone of Robert!
 - `b = a`. Changes to b DO CHANGE a. Changes DO CHANGE b.

Warm-up: Bug hunting

Let's try and find some copy bugs!

Copying lists

Just like objects, the assignment operator results in a *shallow copy*:

```
a = [1, 2, 3]
```

```
b = a # shallow copy! b is a nickname for a
```

```
b[0] = 99 # a and b are the same and now: [99, 2, 3]
```

The independence rule for slicing lists

Slicing a list creates a second, independent list of the original values, when the list contains ints, floats, strings, and booleans (basic data types).

- If the slice contains an *object*, the slice is NOT an independent copy.
- In other words: **independence rule does not apply when a list has an object.**

Example of independence rule (1/2)

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> b
[1, 2]
>>> a[0] = 99
>>> b[1] = -500
>>> a # What does this print?

>>> b # What does this print?
```

Example of independence rule (2/2)

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2] # INDEPENDENT copy of sublist
>>> b
[1, 2]
>>> a[0] = 99 # changing a doesn't influence b
>>> b[1] = -500 # changing b doesn't influence a
>>> a
[99, 2, 3, 4, 5]
>>> b
[1, -500]
```

Slicing a list that has objects

Scenario:

Create 3 Zelle Rectangles. Add to list. Create a second list by slicing (`lecture_16_objects_in_lists_zelle.py`).

List symbol tables

Let's draw the symbol tables for a list of basic data types:

```
nums = [1, 2, 3]  
sliced = nums[:2]
```

and one made by slicing a list of Turtle objects:

```
shapes1 = [rec1, rec2, rec3]  
shapes2 = shapes1[:2]
```

How to make an deep copy of a list

```
newIndependentList = originalList[:].
```

- Example: `a = [1, 2, 3, 4, 5]`

```
b = a[:]
```

- Will `b[0] = 999` affect `a`?
- NO! They are independent :)
- `b = a[:]` only makes a deep copy if `a` contains strings, ints, floats, and booleans.

Colon and assignment does not work for lists with objects

Slicing a list with an object DOES NOT create an independent copy.

```
rec1 = gr.Rectangle(gr.Point(25, 25), gr.Point(75, 75))
rec2 = gr.Rectangle(gr.Point(125, 25), gr.Point(175, 75))
rec3 = gr.Rectangle(gr.Point(225, 25), gr.Point(275, 75))
shapes1 = [rec1, rec2, rec3]
shapes2 = shapes1[:] # NOT independent
```

- Solution?
- Would have to loop thru shapes1, clone each one, append to another list shapes2.

Lists and symbol tables

Let's draw a symbol table for the following main code:

```
rec1 = gr.Rectangle(gr.Point(25, 25), gr.Point(75, 75))
rec2 = gr.Rectangle(gr.Point(125, 25), gr.Point(175, 75))
rec3 = gr.Rectangle(gr.Point(225, 25), gr.Point(275, 75))
shapes1 = [rec1, rec2, rec3]
setColor(shapes1, ['green', 'red', 'blue'])
shapes2 = shapes1[:2]
shapes2[0].setFill('yellow')
rec4 = shapes1[1]
```