Copying and slicing lists

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 15, Fall 2020

Monday September 28
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, Michael and Mike refer to the SAME PERSON!
  • Mike is NOT a clone of Michael!
  • `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*:

```python
da = [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)

```python
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]
>>> b
[1, 2]
>>> a[0] = 99
>>> b[1] = -500
>>> a  # What does this print?
[99, 2, 3, 4, 5]
>>> b  # What does this print?
[1, 2]
```
Example of independence rule (2/2)

```python
>>> 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:

turt1 = turtle.Turtle()
turt1.goto(0, 100)
turt2 = turtle.Turtle()
turt2.goto(0, 200)
turt3 = turtle.Turtle()
turt3.goto(0, 300)
turtles = [turt1, turt2, turt3]
turtSlice = turtles[:2]

Let's run the code in lecture_15_objects_in_lists.py
List symbol tables

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

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

and one made by slicing a list of Turtle objects:

```python
turtles = [turt1, turt2, turt3]
turtSlice = turtles[: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.

```python
turt1 = turtle.Turtle()
turt2 = turtle.Turtle()
turt3 = turtle.Turtle()
turtles = [turt1, turt2, turt3]
turtSlice = turtles[:]  # NOT independent
```

- Solution?
- Would have to loop thru `turtles`, clone each one, append to another list `turtSlice`.
Lists and symbol tables

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

turt1 = turtle.Turtle()
turt2 = turtle.Turtle()
turt3 = turtle.Turtle()
turtles = [turt1, turt2, turt3]
turtSlice = turtles[:2]
turtSlice[0].color('yellow')
turt3 = turtles[1]