

Shallow copy, deep copy, and lists

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 13, Spring 2021

Friday March 12

Warm-up

Find bugs in turtle code that uses objects.

Project 4

Make a race track scene and have two turtles race each other around a track.

- Shape of race course is up to you. Circle is the simplest/recommended.
- One `Turtle` object with dedicated job of drawing the background scene (race track, audience, etc.)
- Two different `Turtle` objects are the racers.
- Two other `Turtle` objects display their score ("scorekeepers") — how many laps Turtle 1 and Turtle 2 complete.
- **All drawing should use objects! No code that looks like `turtle.function()`**

Project 4 organization

- `object_shape.lib.py`: Like `better_shape.lib.py`. All your shape and scene drawing functions. One `Turtle` object created in `object_shape.lib.py` takes care of all this drawing.
 - Running `object_shape.lib.py` should draw your race track scene
- `race.py`: Make the `Screen` object, 2 racing turtles, 2 score keeper turtles. Moves turtles around the race course.
 - Running `race.py` should draw your race track scene, have the turtles race and display their current score.

Finish Turtle shallow and deep copy examples

What is a **shallow copy**?

Shallow copy

```
turt1 = turtle.Turtle()  
turt1.color('red')  
turt2 = turt1  # This does a shallow copy.  
# NOT what you want in above scenario
```

- A **shallow copy** creates a *nickname* — two names for the same data.
- Change turt2 color to red? turt1 also becomes red.
- Change turt1 shape to a turtle? turt2 also gets a turtle shape.
- Let's look at example code then show what happens in the symbol table.

Deep copy (1/2)

Often you want an *independent* (**deep copy**) copy of an object: all data are copied from object to another. Modifying one does not modify the other.

- Means we make a NEW symbol table for 2nd object. Copy all entries of one symbol table to the other.
- Also called **cloning** an object.

Deep copy (2/2)

```
turt1 = turtle.Turtle()  
turt1.color( 'red' )  
turt2 = turt1.clone()  # turt2 is a deep, independent copy
```

- `turtle` provides the `clone` method to do a deep copy (e.g. of `turt1`).
- Changes to `turt2` will not affect `turt1`.
- Let's look at example code then show what happens in the symbol table.

Lists

Why lists?

Situation you may have found yourself in the project:

```
turt1 = make_turtle()  
turt2 = make_turtle()  
turt3 = make_turtle()  
turt3 = make_turtle()
```

```
turt1.move()  
turt2.move()  
turt3.move()  
turt4.move()
```

Tedious!

Lists group multiple things together using one name

Examples:

```
turtles = [turt1, turt2, turt3, turt4]
```

```
numbers = [1, 2, 3, 4, -1, -100, 999]
```

Let's explore lists together in code.