Carnegie Mellon University
Computer Science Academy

CS1: Unit 6 Review

Groups

Groups

- By placing shapes in a group, we can treat them as a single shape
- Ex. `g = Group(Circle(200, 200, 20), Rect(10, 10, 200, 200))` is a group containing a circle and rectangle
- Common errors:
    - Separate each shape by a comma
    - There must be a closing right-parenthesis after all the shapes

Group Properties

- Position Properties:
    - Changing position properties of a group changes the position of every shape in the group
    - Ex. `g = Group(Circle(200, 200, 20), Rect(10, 10, 200, 200))`
        - `g.centerX = 100` will change the `centerX` of the whole group to be 100
- Non-position Properties:
    - We can use groups to set the non-position properties `fill` and `opacity`
    - Groups cannot access or set any other non-position properties
- Shape-specific Properties:
    - Groups cannot access or change shape-specific properties
- Child-specific Properties:
    - Shapes in a group are children of the group
    - The group is the parent of the shapes in the group
    - In order to change the property for only a specific child, we would need to assign it to a variable before and then add that variable into the group

Group Methods

Group Methods

- Shape methods for groups:
    - Groups can use methods like `toFront` and `toBack` or `hitsShape` like shapes can
- Use the `add` method to add a new shape to a group
- Remove all the shapes at once from a group using the `clear` method
- Removing one shape:
    - Use the group's `remove` method and give it the name of the shape you want to remove
    - Remove a shape from a group by setting its `visible` property to `False`
- When not explicitly in a group, shapes are automatically added to `app.group`

Carnegie Mellon University
Computer Science Academy

Step Events and Motion

Step Events

- Sometimes we want our app to change even when the user is not using the mouse or keyboard
- The onStep function takes in no parameters and will run without the user doing anything
- A common use of onStep is to **animate** a shape, so it appears to be moving
- onStep is called 30 times per second by default
    - We can make this faster or slower by changing app.stepsPerSecond

Horizontal Motion

- Only the x coordinates change
- Straight Motion: adding or subtracting from the centerX value on each step
- We use a custom property dx to store the amount that the shape's centerX changes each step
    - Always add dx value and to go left set the dx property to be negative
- Reversing Motion: *change the sign* of dx
    - Ex. Use c.dx = -c.dx to change the sign of c.dx
- Standing Still: set dx to 0, but still add it on every step
- Bounded Motion: stand still when you reach the edge of the canvas
- Wraparound Motion: can have the shape exit on one edge of the canvas and then appear on the other end using conditionals
- Bouncing Motion: when reaching the edge of the canvas, reverse directions by changing the sign of dx

Vertical and Diagonal Motion

- Vertical Motion: same as Horizontal except...
    - use dy instead of dx
    - use top instead of left
    - use bottom instead of right
    - use centerY instead of centerX
- Diagonal Motion: move horizontally and vertically where dy and dx do not have to be the same as we can move along different diagonals
- Smooth Motion:
    - The larger dx is, the faster the shape moves
    - Can speed up the app by increasing app.stepsPerSecond
    - Set both to be larger but not too large that the computer cannot keep up and it does not look smooth