Animation:
A Compelling Example of Objects and Inheritance

CS111 Computer Programming
Department of Computer Science
Wellesley College

Animation:
cslgraphics

We used cslgraphics earlier this semester to make static scenes.
Now we revisit cslgraphics to make dynamic animations
= scenes that change over time.
It is remarkably easy to use Python’s classes/objects/inheritance to make a simple animation framework from scratch.

Example: Run the file simpleAnimation.py from Canopy.
You can stop the animation by typing Ctrl-C Ctrl-C in the interaction pane.

Animation Parts

There are two main parts to our animation framework:

1. Sprites are the actresses in an animation. We create a cast of sprites to act in our play. Each sprite knows how to perform its own part. In particular it knows how to update its state for each time step of the animation.

2. Animations are the plays in which the sprites act. An animation has a canvas on which visual representations of the sprites are displayed. At each time step of the animation, each sprite is asked to update its state, which often changes how it appears on the canvas. As each sprite changes, we see a “movie” of the sprites’ performances.

cslgraphics.py Details

The version of cslgraphics.py we’ve been using this semester has an undocumented feature we’ll use in our animations: Drawable objects have a getAngle() method that returns the angle by which the object has been rotated.

```python
s = Square(100, Point(250,375))
s.getAngle() -> 0.0 # default initial angle is 0.0
s.rotate(90)
s.getAngle() -> 90.0
s.rotate(-12.5)
s.getAngle() -> 77.5
```

We’ll also use the getReferencePoint() method to get the (x,y) coordinates of the reference point of Drawable objects:

```python
position = s.getReferencePoint()
position.getX() -> 250.0
position.getY() -> 375.0
```
**Just RotatingSquares**

Run the file `rotatingSquaresAnimation.py` from Canopy.

```python
from Animation import *
from RotatingSquare import *

rotatingSquares = Animation(800, 600, 'skyblue', 'RotatingSquares')

rotatingSquares.addSprite(
    RotatingSquare(300, 400, 300, 'orange', 45, 5)
)

rotatingSquares.addSprite(
    RotatingSquare(500, 150, 250, 'green', 30, -1)
)

rotatingSquares.start()
```

**RotatingSquare Animation**

```python
from Animation import *
from RotatingSquare import *

rotatingSquares = Animation(800, 600, 'skyblue', 'RotatingSquares')

rotatingSquares.addSprite(
    RotatingSquare(300, 400, 300, 'orange', 45, 5)
)

rotatingSquares.addSprite(
    RotatingSquare(500, 150, 250, 'green', 30, -1)
)

rotatingSquares.start()
```

**Object-Based Diagram**

**Animations consist of sprites that change state over time**

<table>
<thead>
<tr>
<th>stepNum</th>
<th>Angle of orange rectangle (deltaAngle = 5)</th>
<th>Angle of green rectangle (deltaAngle = -1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>25</td>
</tr>
</tbody>
</table>
Animation Class: First Cut

```python
from cs1graphics import *

class Animation:
    def __init__(self, width, height, color, title):
        # Create canvas for showing the sprites
        self.canvas = Canvas(width, height, color, title)
        self.sprites = [] # Keep track of sprites in animation

    def addSprite(self, sprite):
        self.sprites.append(sprite)
        sprite.addToCanvas(self.canvas)

    def start(self):
        while True:
            # Animation is infinite loop.
            # Stop it using Ctrl-C Ctrl-C.
            for sprite in self.sprites:
                sprite.step() # Tell sprite to change by one step
```

Sprite Class: Parent class of other sprite classes

```python
class Sprite:
    def addToCanvas(self, canvas):
        print("Default addToCanvas method called on sprite.")

    def step(self):
        print("Default step method called on sprite")

    def __repr__(self):
        return '{spriteName}<{stateString}>'.format(
            spriteName = self.__class__.__name__,
            stateString = self.stateString()
        )

    def stateString(self):
        return ""
```

### RotatingSquare Class w/o stateString method

```python
from cs1graphics import *

class RotatingSquare(Sprite):
    '''Colored square that rotates.''
    def __init__(self, centerX, centerY, size, color, initialAngle, deltaAngle):
        sq = Square(size, Point(centerX, centerY))
        sq.setFillColor(color)
        sq.rotate(initialAngle)
        self.square = sq
        self.deltaAngle = deltaAngle
        self.color = color

    def addToCanvas(self, self, canvas):
        canvas.add(self.square)

    def step(self):
        self.square.rotate(self.deltaAngle)
```

### Debugging Information

When debugging is turned on, we can see how sprites change over time. How does this work?

```
RotatingSquares[#sprites=2; step#=0; Sprites:
    RotatingSquare<color=orange; deltaAngle=5; angle=45.0>
    RotatingSquare<color=green; deltaAngle=-1; angle=30.0>]

RotatingSquares[#sprites=2; step#=1; Sprites:
    RotatingSquare<color=orange; deltaAngle=5; angle=50.0>
    RotatingSquare<color=green; deltaAngle=-1; angle=29.0>]

RotatingSquares[#sprites=2; step#=2; Sprites:
    RotatingSquare<color=orange; deltaAngle=5; angle=55.0>
    RotatingSquare<color=green; deltaAngle=-1; angle=28.0>]

RotatingSquares[#sprites=2; step#=3; Sprites:
    RotatingSquare<color=orange; deltaAngle=5; angle=60.0>
    RotatingSquare<color=green; deltaAngle=-1; angle=27.0>]

RotatingSquares[#sprites=2; step#=4; Sprites:
    RotatingSquare<color=orange; deltaAngle=5; angle=65.0>
    RotatingSquare<color=green; deltaAngle=-1; angle=26.0>]
```
RotatingSquare Class: stateString method

def stateString(self):
    return "color={c}; deltaAngle={da}; angle={a}".format(
        c = self.square.getFillColor(),
        da = self.deltaAngle,
        a = self.square.getAngle()
    )

For example:

rs1 = RotatingSquare(300, 400, 300, 'orange', 45, 5)

str(rs1) → 'RotatingSquare<color=orange; deltaAngle=5; angle=45.0>'

rs2 = RotatingSquare(500, 150, 250, 'green', 30, -1)

str(rs1) → 'RotatingSquare<color=green; deltaAngle=-1; angle=30.0>'

Animation 21.13

Animation Class: debugging

def __init__(self, width, height, color, title):
    ... other statements from before ...
    self.title = title # Instance variable for bookkeeping
    self.stepNum = 0; # Keep track of step number
    self.inDebugMode = True; # In debug mode by default

def setDebug(self, bool):
    self.inDebugMode = bool

def start(self): # More full-featured than version in slide 21-9
    while True: # Animation is infinite loop.
        # Stop it using Ctrl-C Ctrl-C.
        if self.inDebugMode:
            print(self)
        for sprite in self.sprites:
            sprite.step()
            self.stepNum += 1

def __repr__(self): # represents current state of whole animation
    spriteStrings = "\n    " + \n    + "Sprites:\n    title=self.title, numSprites=len(self.sprites),
    stepNum=self.stepNum, ss=spriteStrings\n    "
    return "{}\n    "

Animation 21.15

Inheritance at work

# In Sprite

def __repr__(self):
    return '{spriteName}<color={c}; deltaAngle={da}; angle={a}>'.format(  
        spriteName = self.__class__.__name__,
        c = self.color,
        da = self.deltaAngle,
        a = self.square.getAngle()
    )

# In RotatingSquare

def stateString(self):
    return "color={c}; deltaAngle={da}; angle={a}".format(
        c = self.color,
        da = self.deltaAngle,
        a = self.square.getAngle()
    )

rs1 = RotatingSquare(300, 400, 300, 'orange', 45, 5)
str(rs1) → '{spriteName}<color=orange; deltaAngle=5; angle=45.0>'

rs2 = RotatingSquare(500, 150, 250, 'green', 30, -1)
str(rs1) → '{spriteName}<color=green; deltaAngle=-1; angle=30.0>'

Animation 21.14

Add a new sprite

In rotatingSquaresAnimation.py
add a new RotatingSquare sprite and run the app again.

Check the debugging information.
Does it make sense?

rotatingSquares = Animation(800, 600, 'skyblue', 'RotatingSquares')

rotatingSquares.addSprite(  
    RotatingSquare(300, 400, 300, 'orange', 45, 5))

rotatingSquares.addSprite(  
    RotatingSquare(500, 150, 250, 'green', 30, -1))

rotatingSquares.start()
Try it out: Rotating squares that move horizontally

Run the file `movingRotatingSquaresAnimation.py` from Canopy.

```python
from Animation import *
from MovingRotatingSquare import *

timeUnit = 0.01

timelapse = 15

movingRotatingSquares = Animation(800, 600, 'skyblue', 'MovingRotatingSquares')

movingRotatingSquares.addSprite(
    MovingRotatingSquare(300, 400, 300, 'orange', 45, 5, 3))

movingRotatingSquares.addSprite(
    MovingRotatingSquare(500, 150, 250, 'green', 30, -1, -2))

movingRotatingSquares.start()
```

### Changing State in `movingRotatingSquaresAnimation`

<table>
<thead>
<tr>
<th>step</th>
<th>angle of orange rect (deltaAngle = 5)</th>
<th>x coord of orange rect (deltaX = 3)</th>
<th>angle of green rect (deltaAngle = -1)</th>
<th>x coord of Green rect (deltaX = -2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>300</td>
<td>30</td>
<td>500</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>303</td>
<td>29</td>
<td>498</td>
</tr>
<tr>
<td>2</td>
<td>55</td>
<td>306</td>
<td>28</td>
<td>496</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>309</td>
<td>27</td>
<td>494</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
<td>312</td>
<td>26</td>
<td>492</td>
</tr>
<tr>
<td>5</td>
<td>70</td>
<td>315</td>
<td>25</td>
<td>490</td>
</tr>
</tbody>
</table>

### Sprite Inheritance: `MovingRotatingSquare`

```python
class MovingRotatingSquare(RotatingSquare):
    '''Colored square that rotates and moves horizontally with speed deltaX.'''
    def __init__(self, centerX, centerY, size, color, initialAngle, deltaAngle, deltaX):
        # Explicitly invoke superclass constructor
        RotatingSquare.__init__(self, centerX, centerY, size, color, initialAngle, deltaAngle)
        self.deltaX = deltaX  # remember deltaX

    def addToCanvas(self):
        # Inherited canvas method
        RotatingSquare.addToCanvas(self)

    def step(self):
        # Inherited rotate and move
        RotatingSquare.step(self)
        self.square.move(self.deltaX, 0)  # move the square

    def stateString(self):
        return RotatingSquare.stateString(self) + '
        ; deltaX={dx}; x={x}'.format(dx=self.deltaX, x=self.square.getReferencePoint().getX())
```

### Debugging info for `MovingRotatingSquares`

New state string info shown in magenta:

```
MovingRotatingSquares[#sprites=2; step#=0; Sprites:
    MovingRotatingSquare=color=orange; deltaAngle=5; angle=45.0; deltaX=3; x=300.0
    MovingRotatingSquare=color=green; deltaAngle=-1; angle=30.0; deltaX=-2; x=498.0]

MovingRotatingSquares[#sprites=2; step#=1; Sprites:
    MovingRotatingSquare=color=orange; deltaAngle=5; angle=50.0; deltaX=3; x=303.0
    MovingRotatingSquare=color=green; deltaAngle=-1; angle=29.0; deltaX=-2; x=498.0]

MovingRotatingSquares[#sprites=2; step#=2; Sprites:
    MovingRotatingSquare=color=orange; deltaAngle=5; angle=55.0; deltaX=3; x=306.0
    MovingRotatingSquare=color=green; deltaAngle=-1; angle=28.0; deltaX=-2; x=496.0]

MovingRotatingSquares[#sprites=2; step#=3; Sprites:
    MovingRotatingSquare=color=orange; deltaAngle=5; angle=60.0; deltaX=3; x=309.0
    MovingRotatingSquare=color=green; deltaAngle=-1; angle=27.0; deltaX=-2; x=494.0]
```
Try it out: Rotating moving squares that bounce

Run the file `bouncingRotatingSquaresAnimation.py` from Canopy.

```python
from Animation import *
from BouncingRotatingSquare import *

bouncingRotatingSquares = Animation(800, 600, 'skyblue', 'BouncingRotatingSquares')
bouncingRotatingSquares.addSprite( BouncingRotatingSquare(300, 400, 300, 'orange', 45, 5, 3))
bouncingRotatingSquares.addSprite( BouncingRotatingSquare(500, 150, 250, 'green', 30, -1, -2))
bouncingRotatingSquares.start()
```

Sprite Inheritance: `BouncingRotatingSquare`

```python
class BouncingRotatingSquare(MovingRotatingSquare):
    # __init__ method inherited
    def addToCanvas(self, canvas):
        # Override inherited method
        MovingRotatingSquare.addToCanvas(self, canvas)
        self.maxX = canvas.getWidth()

    def step(self):
        # Rotate & move square
        pos = self.square.getReferencePoint()
        centerX = pos.getX()
        if centerX < 0 or centerX > self.maxX:
            self.deltaX = -self.deltaX # Change direction

    def stateString(self):
        return MovingRotatingSquare.stateString(self) + "; maxX={mx}".format(mx=self.maxX)
```

**Inherited behavior**
- `__init__` method inherited
- `addToCanvas` method overridden
- `step` method modified
- `stateString` method modified

**Additional behavior for this class**
- Change direction when `x > maxX`

Anatomy of the first bounce of the orange square

(New state string info shown in magenta)

BouncingRotatingSquares[sprites=2; step#=165; Sprites:
BouncingRotatingSquare[color=orange; deltaAngle=5; angle=150.0;
deltaX=3; x=795.0; maxX=800]
]

BouncingRotatingSquares[sprites=2; step#=166; Sprites:
BouncingRotatingSquare[color=orange; deltaAngle=5; angle=155.0;
deltaX=3; x=798.0; maxX=800]
]

BouncingRotatingSquares[sprites=2; step#=167; Sprites:
BouncingRotatingSquare[color=orange; deltaAngle=5; angle=160.0;
deltaX=-3; x=801.0; maxX=800]
]

BouncingRotatingSquares[sprites=2; step#=168; Sprites:
BouncingRotatingSquare[color=orange; deltaAngle=5; angle=165.0;
deltaX=-3; x=798.0; maxX=800]
]

BouncingRotatingSquares[sprites=2; step#=169; Sprites:
BouncingRotatingSquare[color=orange; deltaAngle=5; angle=170.0;
deltaX=-3; x=795.0; maxX=800]
]

Pulsing Circles

Flesh out the `step` method for the `Pulsar` class in the `PulsarSkeleton.py` file (on the next slide) in two stages.

1. A growing circle should stop growing at `maxRadius` (and stay there), while a shrinking circle should stop shrinking at `minRadius` (and stay there).

2. A growing circle should start shrinking when it reaches `maxRadius`, while a shrinking circle should start growing when it reaches `minRadius`.

Try these out by running `pulsingCirclesAnimation.py`. 
class Pulsar(Sprite):
    '''Colored circle that grows/shrinks between min and max radius.''
    def __init__(self, centerX, centerY, initialRadius, color, minRadius, maxRadius, deltaRadius):
        circ = Circle(initialRadius, Point(centerX, centerY))
        circ.setFillColor(color); self.disc = circ;
        self.minRadius = minRadius; self.maxRadius = maxRadius
        self.deltaRadius = deltaRadius
    def stateString(self):
        return "color={c}; minRadius={minR}; maxRadius={maxR}; "
        + "deltaR={dr}; radius={r}".format(  
            c = self.disc.getFillColor(), minR = self.minRadius,  
            maxR = self.maxRadius, dr = self.deltaRadius,  
            r = self.disc.getRadius())
    def addToCanvas(self, canvas):
        canvas.add(self.disc)
    def step(self):
        currentRadius = self.disc.getRadius()
        # flesh out the rest of this method

Try it out: bouncing images
Run the file
bouncingImagesAnimation.py
from Canopy.

Nonmoving ImageSprite
from cs1graphics import *
from Sprite import *

class ImageSprite(Sprite):
    '''Sprite that has an nonmoving image at a given position.  
    This is a superclass for sprites with moving images.''
    def __init__(self, centerX, centerY, imageFile):
        img = Image(imageFile)
        img.moveTo(centerX, centerY)
        self.image = img
        self.imageName = imageFile.split('.')[0]
    def addToCanvas(self, canvas):
        canvas.add(self.image)
    def stateString(self):
        return "image='{n}'".format(n = self.imageName)

BouncingImage Sprite [1]
class BouncingImage(ImageSprite):
    '''Horizontally moving image bouncing off 
    vertical canvas edges'''
    def __init__(self, centerX, centerY, imageFile, deltaX):
        ImageSprite.__init__(self, centerX, centerY, imageFile)
        self.deltaX = deltaX  
        # Change in x position for each step().
        # If deltaX is positive, moves right;
        # if deltaX is negative, moves left
    def addToCanvas(self, canvas):
        ImageSprite.addToCanvas(self, canvas)
        # Remember maximium x value of right edge for step method
        self.maxX = canvas.getWidth() - self.image.getWidth()/2.0
    def step(self):
        # flesh out the rest of this method
BouncingDropper Class [1]

```
from BouncingImage import *
from FallingImageThatSticks import *
from FallingImageThatDisappears import *
import randomclass

class BouncingDropper(BouncingImage):
    def __init__(self, centerX, centerY, picfile, deltaX, animation, droppedImage, dropProbability, doesDroppedImageStick):
        BouncingImage.__init__(self, centerX, centerY, picfile, deltaX)
        self.animation = animation
        self.droppedImage = droppedImage
        self.dropProbability = dropProbability
        self.doesDroppedImageStick = doesDroppedImageStick

        # should be between 0.0 and 1.0
        self.droppedImageSticky += self.doesDroppedImageStick

        # Inherit addToCanvas from BouncingImage

        # Implementation continued on the next slide
```

---

BouncingDropper Class [2]

```
def step(self):
    # Move the image
    if random.random() < self.dropProbability:
        # random.random()) returns random num between 0.0 and 1.0.
        # Add new FallingImage to animation, with random
        # falling speed between 5 and 15.
        pos = self.image.getReferencePoint()
        self.doesDroppedImageStick = False
        self.droppedImageSticky = False
        self.droppedImage = False
        self.dropProbability = random.randint(0.01, 0.1)  # random.randint(5, 15))
        self.doesDroppedImageStick = doesDroppedImageStick
    else:
        self.animation.addSprite(FallingImageDisappears(
            self.animation, pos.getX(), self.image.getHeight()/2.0,
            self.droppedImage, random.randint(5, 15))))

    def stateString(self):
        return ImageSprite.stateString(self) + '
        "; deltaX={dx}; maxX={mx}; x={x}".format(<
            dx = self.deltaX,
            mx = self.maxX,
            x = self.image.getReferencePoint().getX()
        )
```

---

BouncingDropper that Stick

```
from Animation import *
from BouncingDropper import *

bouncingDroppers = 
Animation(800, 600, 'skyblue',
    'BouncingDroppersSticking')

bouncingDroppers.addSprite(
    BouncingDropper(200, 50, 'cloud.gif', 15, bouncingDroppers,
        'raindrop.gif', 0.1, True))

bouncingDroppers.addSprite(
    BouncingDropper(600, 75, 'alien.gif', 2, bouncingDroppers,
        'flame.gif', 0.02, True))

bouncingDroppers.start()
```

---

Bouncing Image Sprite [2]

```
def step(self):
    # Override Sprite’s default step method
    imagePos = self.image.getReferencePoint()
    imageWidth = self.image.getWidth()
    nextX = imagePos.getX() + self.deltaX
    if nextX <= imageWidth/2.0:
        nextX = imageWidth/2.0  # Never let image go off left edge
    self.deltaX = -self.deltaX  # change deltaX from - to +
    elif nextX >= imageWidth:
        nextX = self.maxX:
        self.deltaX = -self.maxX:
        nextX = self.deltaX:  # Never let image go off right edge
    self.deltaX = -self.deltaX  # # change deltaX from + to -
    self.image.moveTo(nextX, imagePos.getY())  # Move to new pos

    def stateString(self):
        return ImageSprite.stateString(self) + '
        "; deltaX={dx}; maxX={mx}; x={x}".format(<
            dx = self.deltaX,
            mx = self.maxX,
            x = self.image.getReferencePoint().getX()
        )
```
**FallingImageThatSticks Class [1]**

class FallingImageThatSticks(ImageSprite):
    '''Vertically falling image that sticks to bottom of canvas.'''
    def __init__(self, animation, centerX, centerY, imageFile, deltaY):
        '''Assume deltaY is positive'''
        ImageSprite.__init__(self, animation, centerX, centerY, imageFile)
        self.deltaY = deltaY
    def addToCanvas(self, canvas):
        ImageSprite.addToCanvas(self, canvas)
        self.maxY = canvas.getHeight() - self.image.getHeight()/2.0
    def step(self):
        '''sprite falls and sticks to bottom of canvas'''
        imagePos = self.image.getReferencePoint()
        centerX = imagePos.getX()
        centerY = imagePos.getY() + self.deltaY
        if centerY > self.maxY:
            centerY = self.maxY # Don’t let img go beyond canvas bottom
        self.image.moveTo(centerX, centerY) # Stick at bottom

**FallingImageThatSticks Class [2]**

def stateString(self):
    imagePos = self.image.getReferencePoint()
    posTuple = (imagePos.getX(), imagePos.getY())
    return ImageSprite.stateString(self) + 
        f'; deltaY={dy}; pos={pt}'.format(dy=self.deltaY, pt=posTuple)

Problem: Dropped Sticking Sprites never go away!

Animation gets slower and slower as more sprites are added.

Fortunately, the **Animation** class also contains the following method for removing sprites:

def removeSprite(self, sprite):
    self.sprites.remove(sprite) # Remove sprite from list
    sprite.removeFromCanvas(self.canvas) # Sprite determines how to remove itself from canvas

Try this out in **bouncingDroppersDisappearingAnimation.py**

**FallingImageThatDisappears Class**

Try this out in **bouncingDroppersDisappearingAnimation.py**

class FallingImageDisappears(FallingImageThatSticks):
    '''Vertically falling image that disappears when it hits bottom of canvas.'''
    def step(self):
        '''sprite falls and disappears'''
        FallingImageSticks.step(self)
        if self.image.getReferencePoint().getY() >= self.maxY:
            self.animation.removeSprite(self)
    def removeFromCanvas(self, canvas):
        canvas.remove(self.image)
from Animation import *
from BouncingDropper import *

bouncingDroppers = Animation(800, 600, 'skyblue', 'BouncingDroppersDisappearing')

bouncingDroppers.addSprite(BouncingDropper(200, 50, 'cloud.gif', 15, bouncingDroppers, 'raindrop.gif', 0.1, False))

bouncingDroppers.addSprite(BouncingDropper(600, 75, 'alien.gif', 2, bouncingDroppers, 'flame.gif', 0.02, False))

bouncingDroppers.start()