15-112
Lecture 2

OOP Part 2 & TP Tech

Instructor: Pat Virtue

Announcements

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Week 12		TP0 due 5pm	OOP Part2		🦃		
Week 13		TP1 due **5pm**	Cool Stuff		Cool Stuff	TP2 due **5pm**	
Week 14			TP User Study	TP3 due **5pm**	TP Showcase		
Week 15						Final 1-4 pm	

Topics

OOP and Animations

Events++

Movement: Basic Physics

OOP Inheritance

Images

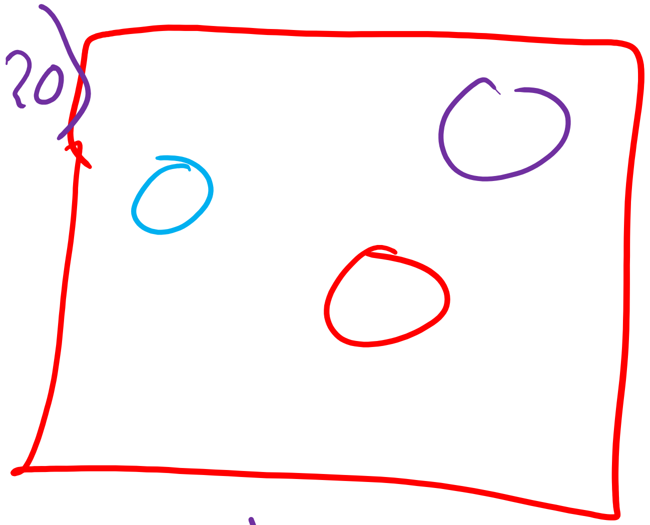
Sprites

Organizing code in separate python files

OOP and Animation

app.c = Circle(10, 20)

Blob class



Attributes

x y r color ... bord

Constructor

__init__(self, ___ ___ ___ ...)

Methods

draw(self)

drawCircle(self .c . x, self.c.y, self.c.r)

OOP and Animation

Blob class

```python
def onAppStart(app):
    app.blobs = []

def onMousePress(app, mx, my):
    newBlob = Blob(mx, my)
    app.blobs.append(newBlob)

def redrawAll(app):
    for blob in app.blobs:
        blob.draw()
```

```python
class Blob:
    def __init__(self, x, y, r=20):
        self.x = x
        self.y = y
        self.r = r
        self.color = 'cornflowerBlue'

    def draw(self):
        drawCircle(self.x, self.y, self.r,
             fill=self.color)
```

OOP and Animation

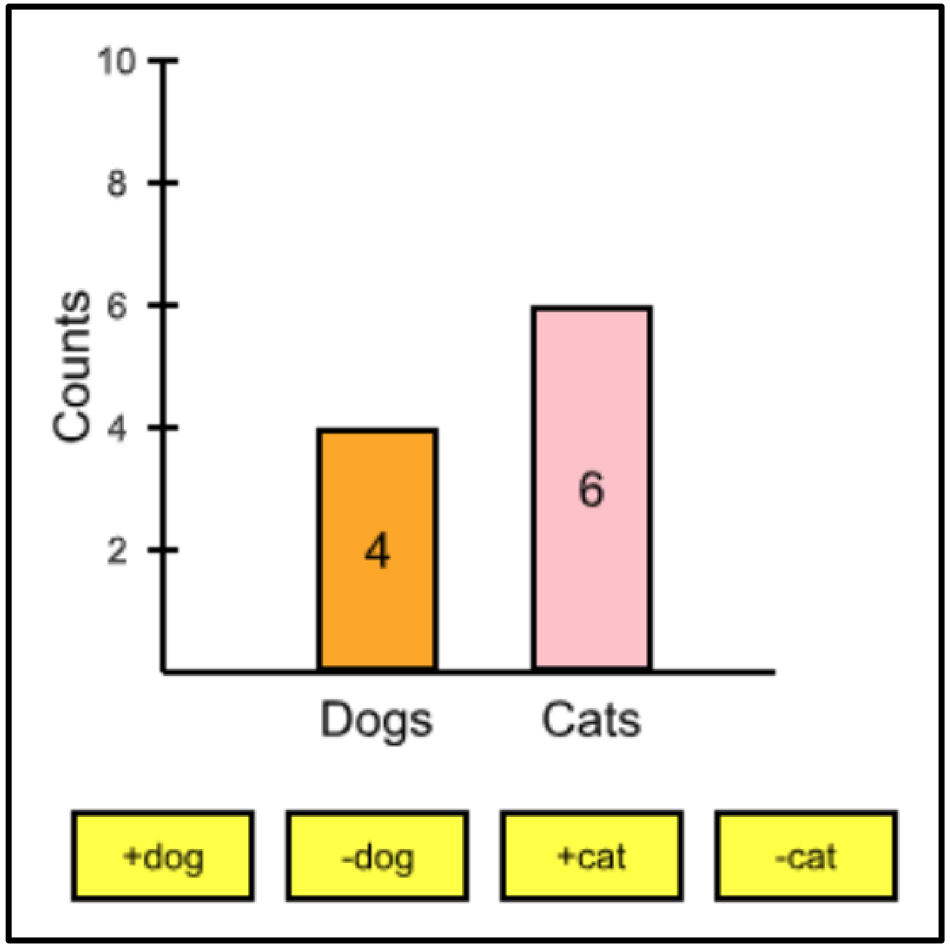
BlinkingBlob class

```
def onAppStart(app):
    app.blobs = []

def onMousePress(app, mx, my):
    newBlob = BlinkingBlob(mx, my)
    app.blobs.append(newBlob)

def onStep(app):
    for blob in app.blobs:
        blob.onStep()

def redrawAll(app):
    for blob in app.blobs:
        blob.draw()
```

```
class BlinkingBlob:
    def __init__(self, x, y, r=20):
        self.x = x
        self.y = y
        self.r = r
        self.color = 'cornflowerBlue'
        self.blinkOn = True

    def onStep(self):
        self.blinkOn = not self.blinkOn

    def draw(self):
        if self.blinkOn:
            fillColor = self.color
        else:
            fillColor = None
        drawCircle(self.x, self.y, self.r,
                   fill=fillColor)
```

OOP-ing

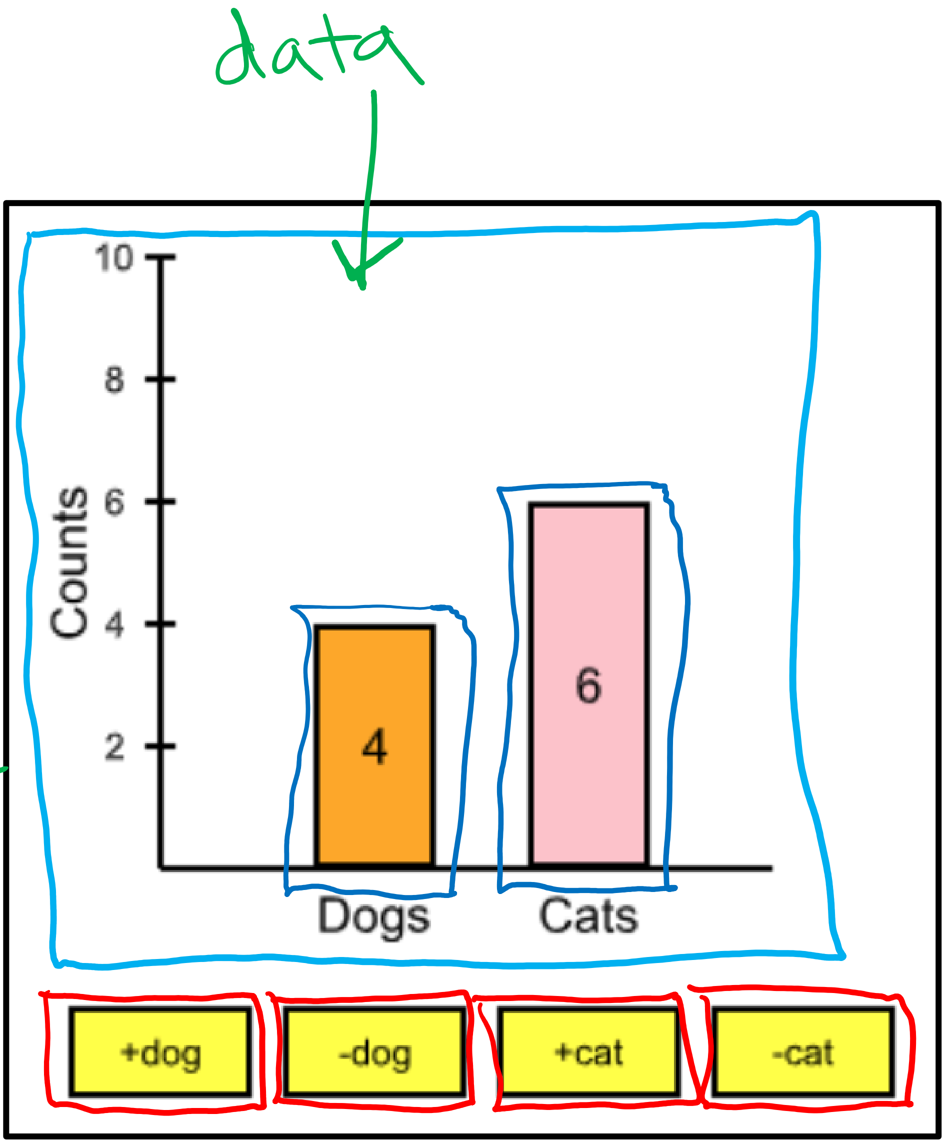
Starting from 3.3.7 exercise



Poll 1

Which of the following classes would be helpful to create?

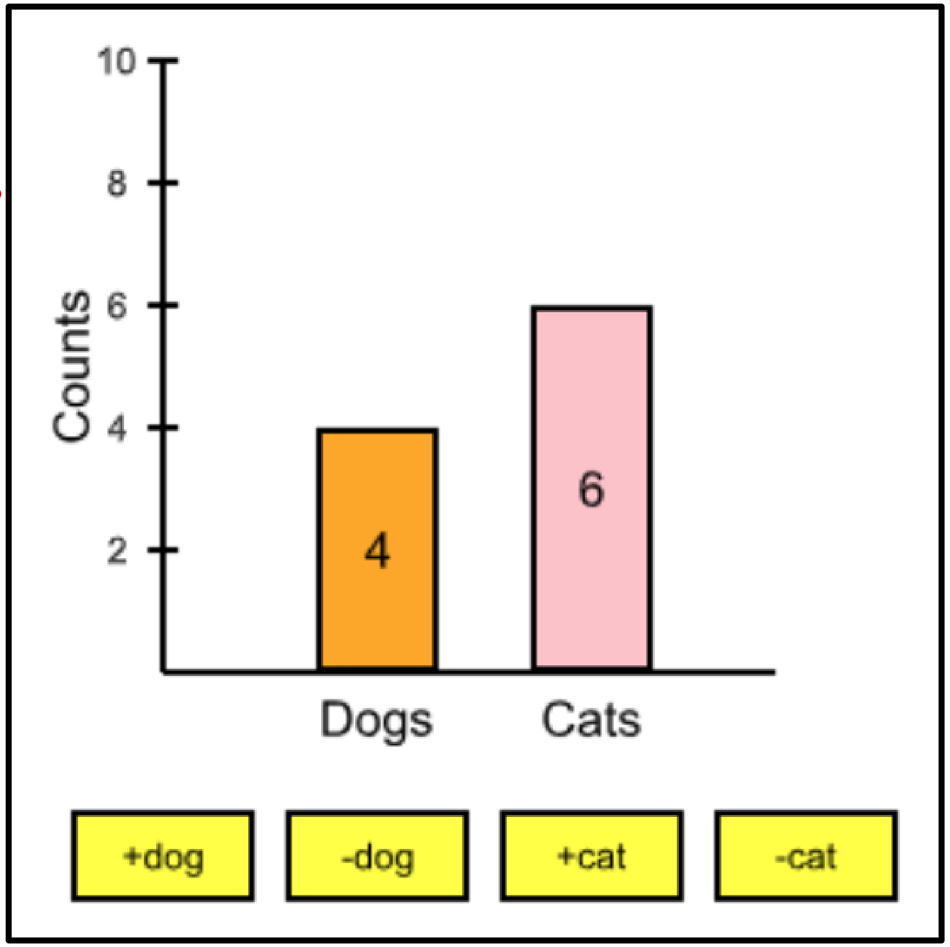
- ✗ A. Dog too specific
- ✗ B. DogCount probably just an int
- C. Animal – name – count • increase() • decrease()
- D. Data ← maybe just a dict
- [✓ E. BarGraph] • BarGraph(data) label: count
- F. Bar ← maybe
- [✓ G. Button]
- ✗ H. DogButton ← too specific
- ✗ I. IncreaseDogButton



Poll 2

Which of the following methods would be helpful for a Button class?

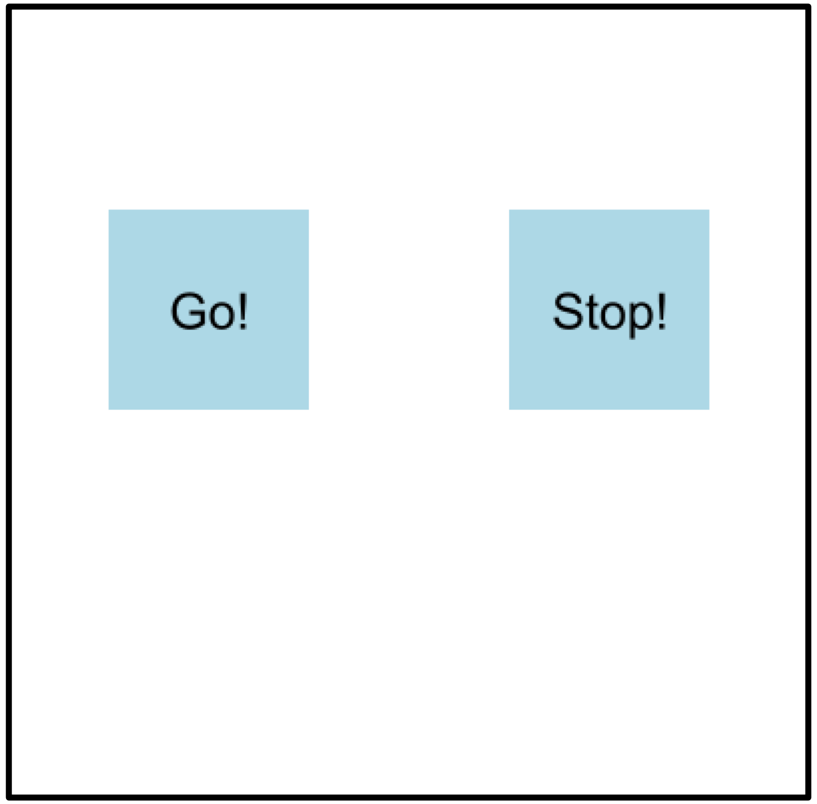
✓ A. draw
✓ B. onMousePress
C. onButtonPress ← kinda backwalds
✓ D. getLeftTopRightBotton ← probably just used in Button onMousePress
E. getCount
F. getAnimal ← too specific



OOP and Animations

Example: Button Class

```
def onAppStart(app):
    button1 = Button(50, 200, 100,
                     'Go!')
    button2 = Button(250, 200, 100,
                     'Stop!')
    app.buttons = [button1, button2]

def onMousePress(app, mx, my):
    for button in app.buttons:
        button.onMousePress(mx, my)

def redrawAll(app):
    for button in app.buttons:
        button.draw()
```



```
>>> Go! button clicked!
Stop! button clicked!
Go! button clicked!
Stop! button clicked!
Go! button clicked!
Stop! button clicked!
```

OOP and Animations

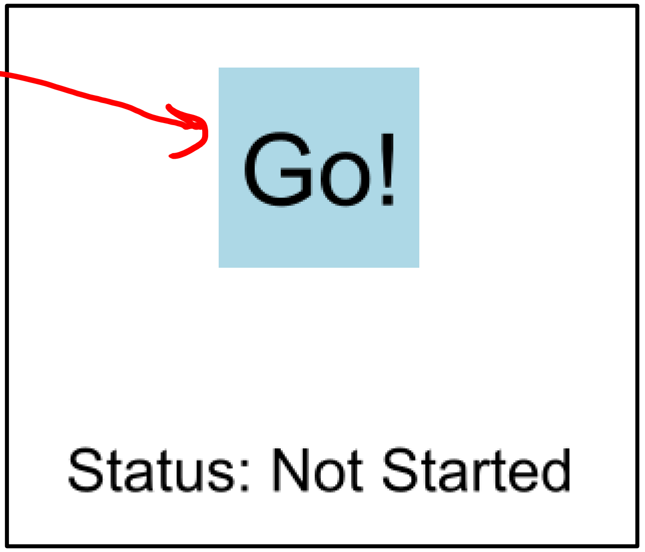
Example: Button Class

```
def onAppStart(app):
    button1 = Button(50, 200, 100,
                     'Go!')
    button2 = Button(250, 200, 100,
                     'Stop!')
    app.buttons = [button1, button2]

def onMousePress(app, mx, my):
    for button in app.buttons:
        button.onMousePress(mx, my)

def redrawAll(app):
    for button in app.buttons:
        button.draw()
```

```
class Button:
    def __init__(self, x, y, size, text):
        self.x = x
        self.y = y
        self.text = text
        self.size = size
        self.color = 'lightBlue'

    def pointInBounds(self, px, py):
        ...

    def onMousePress(self, mx, my):
        if self.pointInBounds(mx, my):
            print(f'{self.text} button clicked!')

    def draw(self):
        drawRect(self.x, self.y, self.size, self.size,
                 fill=self.color)

        cx = self.x+self.size/2
        cy = self.y+self.size/2
        drawLabel(self.text, cx, cy, align='center',
                  size=self.size/4)
```

OOP Events

source



Connecting controls with models

updateStatus

Callback pattern

Provide the source with a function (a callback function)

When the event happens in the source, the source will call the callback function

status object status.update()

Observer (listener) pattern

Provide the source with an object that contains a specific update function

When the event happens in the source, the source will call the object's update function.

Callback Pattern

Example

Click Go button to change status from "Not Started" to "Started"

Application code

```
def onAppStart(app):
    app.status = StatusData()
    app.button = Button(150, 100, 100, 'Go!',
                        app.status.update)
    # Note: No () after update. We want to
    # pass the update function not call it

def onMousePress(app, mx, my):
    app.button.onMousePress(mx, my)

def redrawAll(app):
    app.button.draw()
    drawLabel(app.status.text, 200, 300)
```

Source

```
class Button:
    def __init__(self, x, y, size, text,
                 callbackFunction):
        ...
        self.callback = callbackFunction

    def onMousePress(self, mx, my):
        if self.pointInBounds(mx, my):
            # Call callback function
            self.callback()
    ...
```

Callback function will be the update function

```
class StatusData:
    def __init__(self):
        self.text = 'Status: Not Started'

    # The callback function given to source
    def update(self):
        self.text = 'Status: Started!'
```

Observer Pattern

Example

Click Go button to change status from "Not Started" to "Started"

Source

```
class Button:
    def __init__(self, x, y, size, text):
        ...
        self.observers = set()

    def addObserver(self, observer):
        self.observers.add(observer)

    def onMousePress(self, mx, my):
        if self.pointInBounds(mx, my):
            for observer in self.observers:
                observer.update()
    ...
```

Application code

```
def onAppStart(app):
    app.status = StatusData()
    app.button = Button(150, 100, 100, 'Go!')

    app.button.addObserver(app.status)

def onMousePress(app, mx, my):
    app.button.onMousePress(mx, my)

def redrawAll(app):
    app.button.draw()
    drawLabel(app.status.text, 200, 300)
```

Observer object

```
class StatusData:
    def __init__(self):
        self.text = 'Status: Not Started'

    # Listens for update from source
    def update(self):
        self.text = 'Status: Started!'
```

More Events

CS Academy Docs: Advanced Events

```
def onKeyPress(app, key, modifiers)
```

```
def onMousePress(app, mouseX, mouseY, button)
```

Topics

OOP and Animations

Events++

Movement: Basic Physics

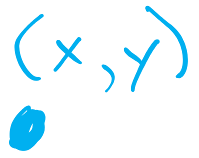
OOP Inheritance

Images

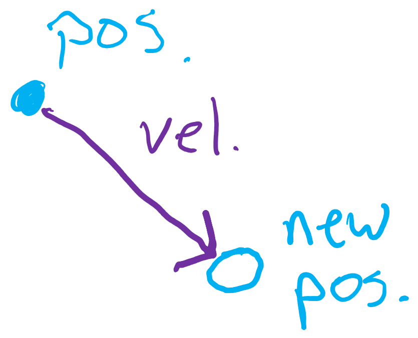
Sprites

Organizing code in separate python files

Movement: Basic Physics



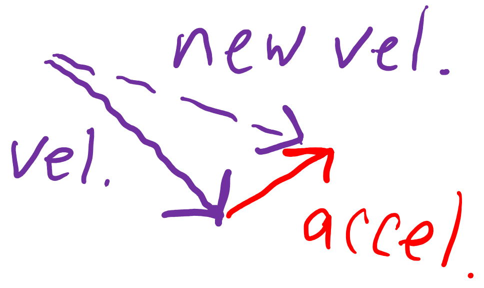
Position x, y



Velocity vx, vy $\leftarrow \frac{\text{dist}}{\text{time}}$

x = x + vx # Every time unit

y = y + vy



Acceleration ax, ay $\leftarrow \frac{\text{change vel}}{\text{time}} = \frac{\text{dist}}{\text{time}^2}$

vx = vx + ax

vy = vy + ay

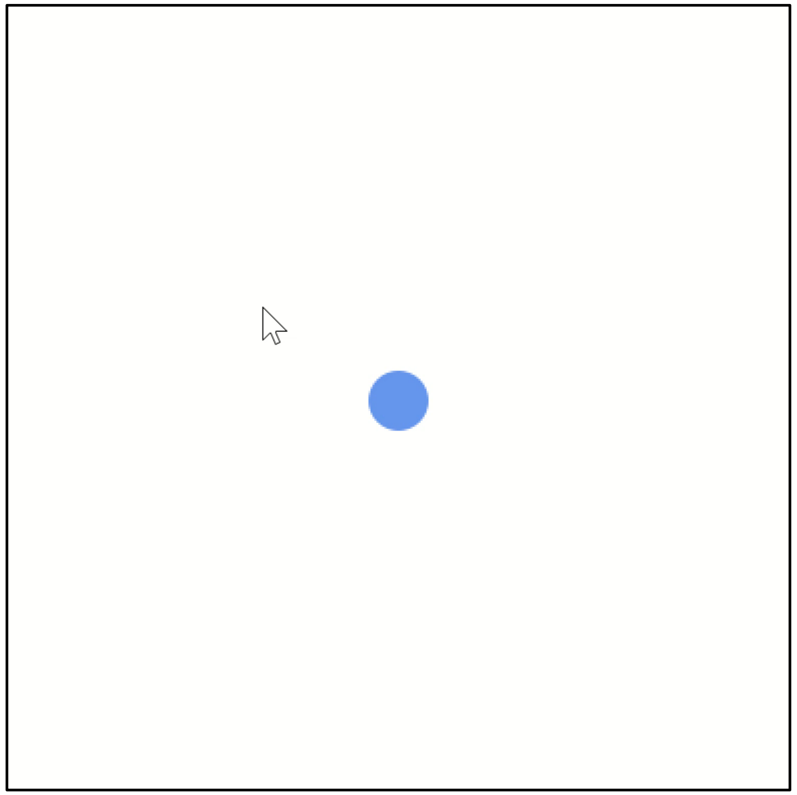
Increase/decrease speed

Also, changes direction

Movement: Basic Physics

onStep $\longrightarrow$ app.takeStep()

```
class Thing
    __init__(self):
        x = 100
        y = 100
        vx = 0
        vy = 0

    takeStep(self):
        x += vx
        y += vy

    accelerate(self, ax, ay)
        vx += ax
        vy += ay
```



Movement: Example

```python
def onAppStart(app):
    app.thing = Thing(200, 200)

def onStep(app):
    app.thing.takeStep()

def onMousePress(app, mx, my):
    # Accelerate towards mouseClick
    # Scale it down to be resonable
    scale = 0.01
    app.thing.accelerateTowardsPoint(
        mx, my, scale)

def redrawAll(app):
    app.thing.draw()
```

```python
class Thing:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.vx = 0
        self.vy = 0

    def takeStep(self):
        self.x += self.vx
        self.y += self.vy

    def accelerate(self, ax, ay):
        self.vx += ax
        self.vy += ay

    def accelerateTowardsPoint(self, x, y, scale):
        # Accelerate (change velocity) towards point
        ax = (x - self.x) * scale
        ay = (y - self.y) * scale
        self.accelerate(ax, ay)

    def draw(self):
        drawCircle(self.x, self.y, 15)
```

OOP Inheritance

From Notes

```
class FarmAnimal:
    def __init__(self, name):
        self.name = name
        self.says = 'Generic Animal Sound'

    def speak(self):
        return f'{self.name} says {self.says}'

class Pig(FarmAnimal):
    def __init__(self, name):
        super().__init__(name)
        self.says = 'Oink'

class Cow(FarmAnimal):
    def __init__(self, name):
        super().__init__(name)
        self.says = 'Moo'

animal1 = FarmAnimal('Fred')
animal2 = Pig('Barney')
```

OOP Inheritance

Dragon example

class Thing
→ core movemet physics

class Dragon(Thing)
→ Draw as image
→ Add chase(Thing)

class BabyDragon(Dragon)
→ Hatch timer
→ Stationary egg until hatch timer is zero

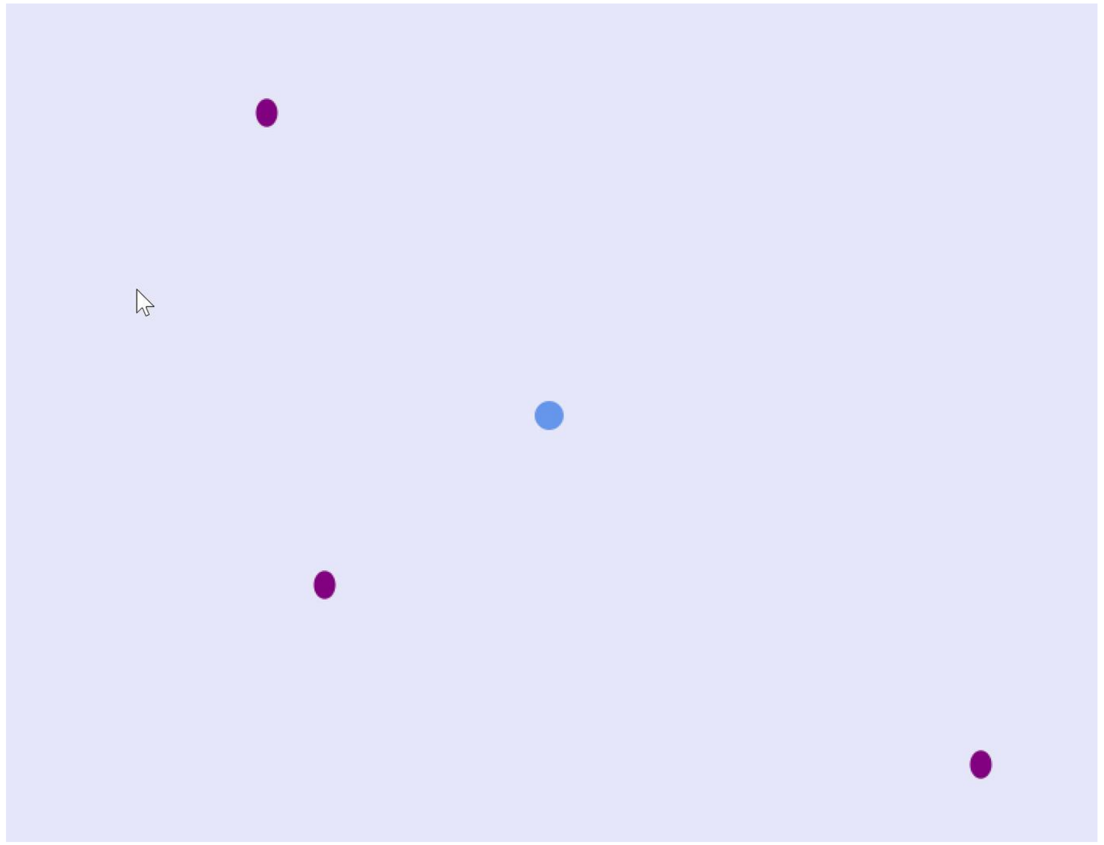


Image source: https://www.silhouette.pics/tags/overhead-dragon.php

OOP Inheritance

Dragon example

```python
class Dragon(Thing):
    def __init__(self, x, y, size):
        super().__init__(x, y, size)
        imFilename = 'dragon.png'
        imPIL = Image.open(imFilename)
        self.image = CMUImage(imPIL)
        self.target = None

    def chase(self, thing):
        self.target = thing

    def takeStep(self):
        # Use superclass to move with physics
        super().takeStep()

        if self.target is not None:
            self.accelerateTowardsPoint(
                self.target.x, self.target.y)

    def draw(self):
        angle = calcAngle(self.vx, self.vy)

        drawImage(self.image,
            self.x, self.y, align='center',
            width=self.size, height=self.size,
            rotateAngle=angle)
```

OOP Inheritance

Dragon example

```python
class BabyDragon(Dragon):
    def __init__(self, x, y, mother):
        super().__init__(x, y, mother.size/4)
        self.eggColor = 'purple'

        hatchTime = 10
        self.hatchTimer = hatchTime
        self.chase(mother)

    def takeStep(self):
        if self.hatchTimer > 0:
            self.hatchTimer -= 1
        else:
            # Use superclass to move
            super().takeStep()

    def drawEgg(self):
        width = 0.3 * self.size
        height = 1.3 * width
        drawOval(self.x, self.y,
                 width, height,
                 fill=self.eggColor)

    def draw(self):
        if self.hatchTimer > 0:
            self.drawEgg()
        else:
            # Use superclass to draw
            # as a dragon
            super().draw()
```

Topics

OOP and Animations

Events++

Movement: Basic Physics

OOP Inheritance

Images

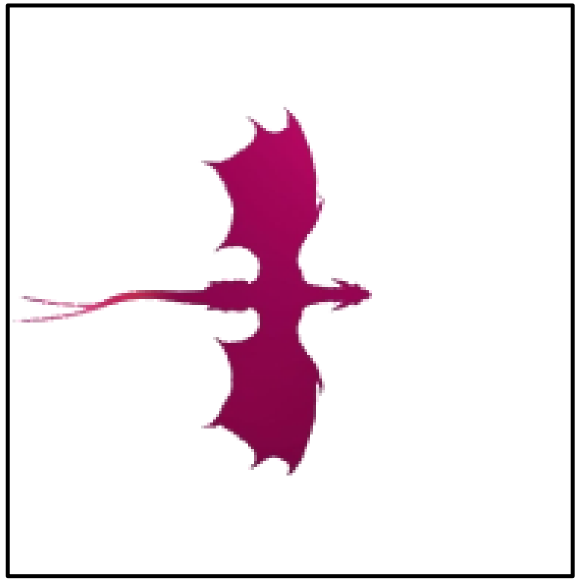
Sprites

Organizing code in separate python files

Images

Pillow (PIL) Image allows for a ton of image operations

```
def onAppStart(app):
    imageFilename = 'dragon.png'
    imPIL = Image.open(imageFilename)

    # insert any PIL image manipulation here, e.g. crop
    imPIL = imPIL.crop((200, 100, 340, 425))

    app.imCMU = CMUImage(imPIL)

def redrawAll(app):
    drawImage(app.imCMU, 100, 100)
```



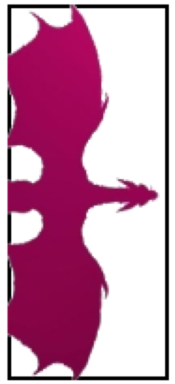


Image source: https://www.silhouette.pics/tags/overhead-dragon.php

Images

Aspect ratio to display image with desired size without distortion

```
def onAppStart(app):
    imageFilename = 'dragon.png'
    imPIL = Image.open(imageFilename)
    app.imCMU = CMUImage(imPIL)

    aspectRatio = imPIL.width / imPIL.height
    app.imHeight = 200
    app.imWidth = rounded(app.imHeight * aspectRatio)

def redrawAll(app):
    drawImage(app.imCMU, 100, 100,
              width=app.imWidth, height=app.imHeight)
```

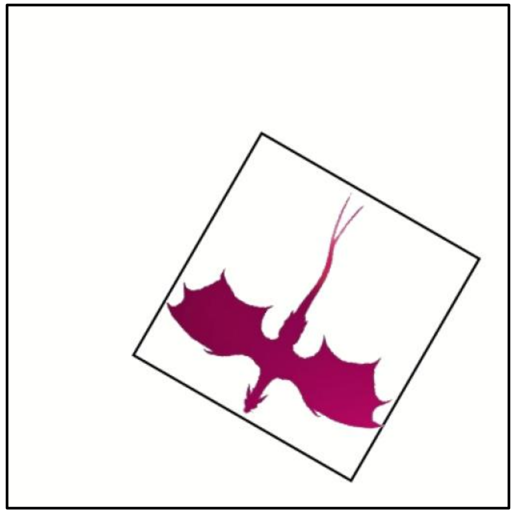
Images

Rotation using rotateAngle parameter

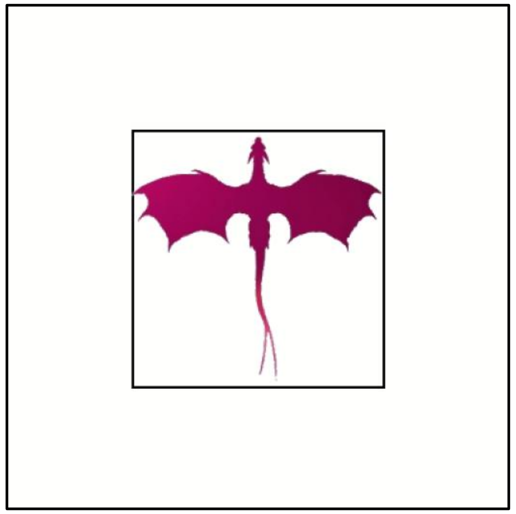
```
    drawImage(app.image, app.x, app.y, align='center',
                      rotateAngle=angle)
```

You can certainly do this, but you only have to pay attention to the align parameter (and it has limitations)

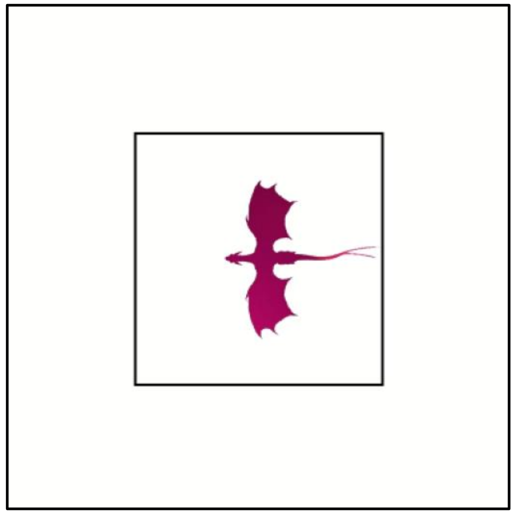
Changed image file to put the dragon's neck at the center



`x=100, y=100, align='left-top'`



`x=200, y=200, align='center'`



`x=200, y=200, align='center'`

Image source: https://www.silhouette.pics/tags/overhead-dragon.php

Images

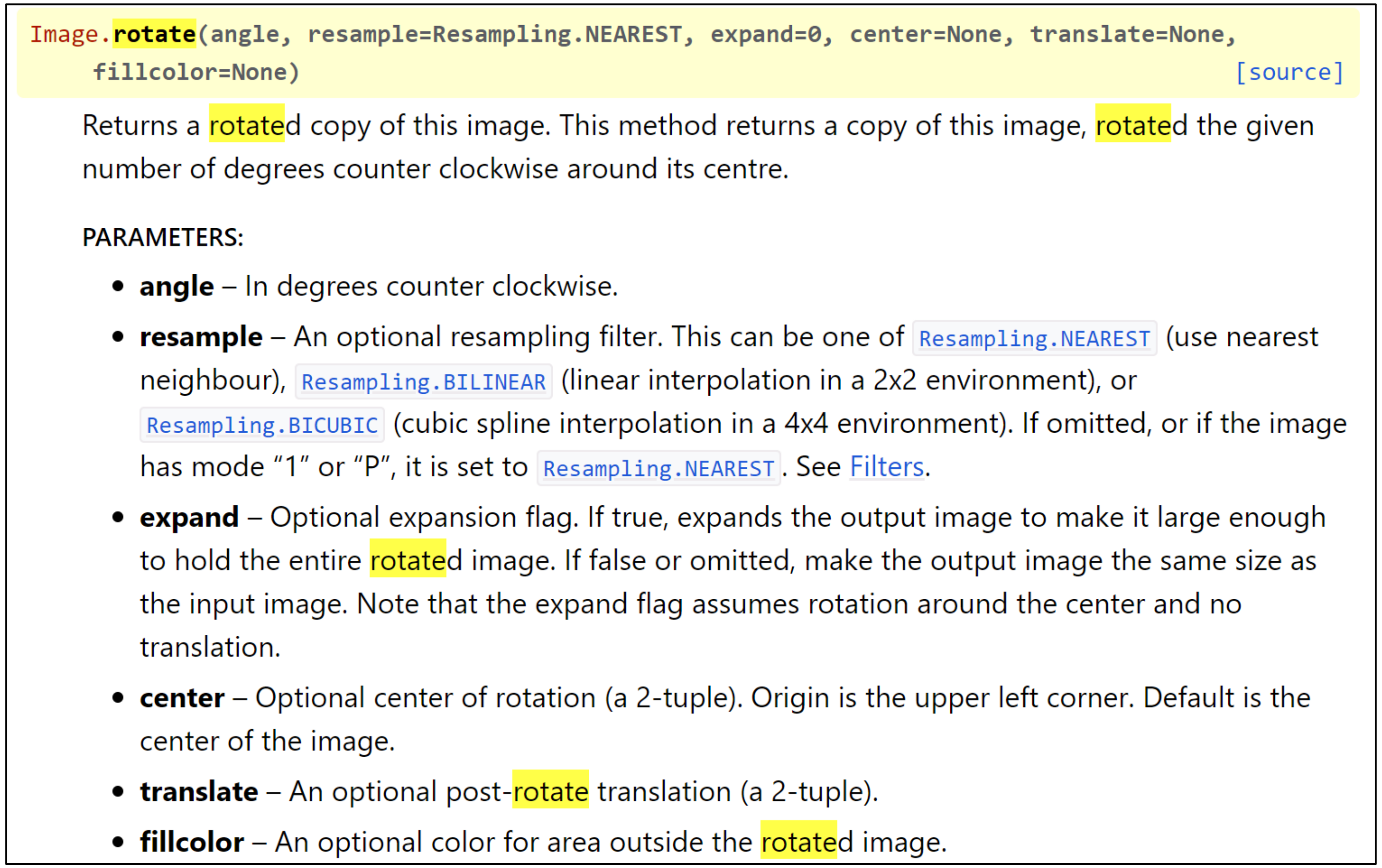
Rotation using PIL rotate

More flexibility

But more complex

Directly rotate the image before drawing

```
imPIL = imPIL.rotate(angle)
```



```
Image.rotate(angle, resample=Resampling.NEAREST, expand=0, center=None, translate=None,
    fillcolor=None)
```
[source]

Returns a rotated copy of this image. This method returns a copy of this image, rotated the given number of degrees counter clockwise around its centre.

PARAMETERS:

- **angle** – In degrees counter clockwise.
- **resample** – An optional resampling filter. This can be one of `Resampling.NEAREST` (use nearest neighbour), `Resampling.BILINEAR` (linear interpolation in a 2x2 environment), or `Resampling.BICUBIC` (cubic spline interpolation in a 4x4 environment). If omitted, or if the image has mode “1” or “P”, it is set to `Resampling.NEAREST`. See Filters.
- **expand** – Optional expansion flag. If true, expands the output image to make it large enough to hold the entire rotated image. If false or omitted, make the output image the same size as the input image. Note that the expand flag assumes rotation around the center and no translation.
- **center** – Optional center of rotation (a 2-tuple). Origin is the upper left corner. Default is the center of the image.
- **translate** – An optional post-rotate translation (a 2-tuple).
- **fillcolor** – An optional color for area outside the rotated image.

https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image.rotate

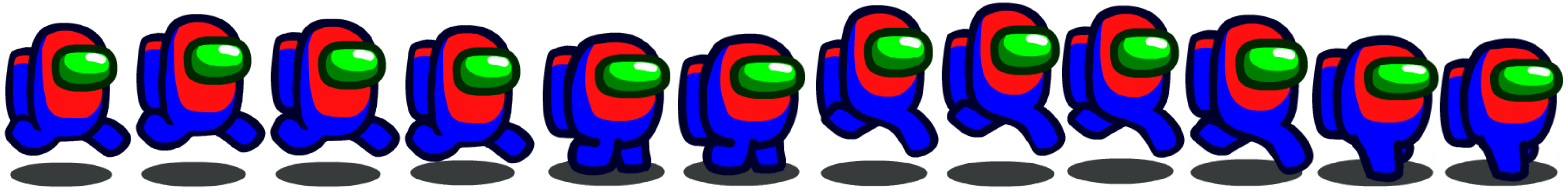
Sprite images: https://www.spriters-resource.com/pc_computer/amongus/sheet/157129/

Sprites

AmongUs Example

```
def onAppStart(app):
    numImages = 12
    app.images = []
    for i in range(numImages):
        filename = f'images/im{i}.png'
        imPIL = Image.open(filename)
        imCMU = CMUImage(imPIL)
        app.images.append(imCMU)

    app.imageIndex = 0
```

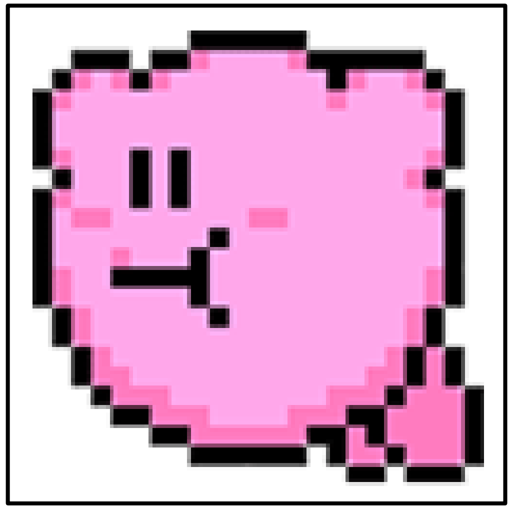
```
def onStep(app):
    app.imageIndex += 1
    app.imageIndex %= len(app.images)

def redrawAll(app):
    image = app.images[app.imageIndex]
    drawImage(image, 200, 200)
```


im0.png im1.png im2.png im3.png im4.png im5.png im6.png im7.png im8.png im9.png im10.png im11.png

Sprites

Images from gif file

```
def onAppStart(app):
    myGif = Image.open('kirby.gif')
    app.images = []
    for frameIndex in range(myGif.n_frames):
        # Set the desired frame in the gif
        myGif.seek(frameIndex)
        # Copy the image frame from the gif
        imPIL = myGif.copy()
        imCMU = CMUImage(imPIL)
        app.images.append(imCMU)

    app.imageIndex = 0
```



Topics

OOP and Animations

Events++

Movement: Basic Physics

OOP Inheritance

Images

Sprites

Organizing code in separate python files

Organizing code in different files

Term projects certainly get large enough that putting all of your code in one file gets a bit nutty. You can organize your code in separate *.py files to add some sanity to your life ☺

- One convention (though not necessarily a Python convention) is to put one class per file. You can also put more than one class in a file.
- The key to this organization is to name your files such that it is easy to infer what is inside it.

Mechanics

Let's say file2.py has class A in it, you can add the following line to file1.py to access class A within file1.py:

```
from blob import Blob
```

Organizing code in different files

Importing * (everything) is actually problematic and should be avoided

Example

blob.py

```
from cmu_graphics import *

class Blob:
    def __init__(self, x, y, r=20):
        self.x = x
        self.y = y
        self.r = r
        self.color = 'lightBlue'

    def draw(self):
        drawCircle(self.x, self.y,
            self.r, fill=self.color)
```

main_app.py

```
from cmu_graphics import *
from blob import Blob

def onAppStart(app):
    app.blobs = []

def onMousePress(app, mx, my):
    newBlob = Blob(mx, my)
    app.blobs.append(newBlob)

def redrawAll(app):
    for blob in app.blobs:
        blob.draw()

def main():
    runApp()

main()
```