Animating Objects

- **Naïve** animations are easy
  - Look at the key input right now
  - Move the objects based on the keys
  - Redraw the moved objects
- **Timed** animations are harder
  - Press a key to start the animation
  - Animation continues for X seconds
  - Animation stops automatically when done

animate1.py

animate2.py

1

Animation Needs Many Attributes

```
def _animate_turn(self,dt):
    """Animates a rotation of the image over SPEED seconds"""
    # Compute degrees per second
    steps = (self._fangle-self._sangle)/SPEED
    amount = steps*FRAME_RATE
    # Update the angle
    self.image.angle = self.image.angle+amount
    # If we go to far, clamp and stop animating
    if abs(self.image.angle-self._sangle) >= 90:
        self.image.angle = self._fangle
        self._animating = False
```

2

Wouldn't a Loop Be Simpler?

```
def _animate_turn(self,direction):
    """Animates a rotation of the image over SPEED seconds"""
    sangle = self.image.angle
    fangle = sangle+90 if direction == 'left' else sangle-90
    steps = (fangle-sangle)/ANIMATION_SPEED      # Degrees per second
    animating = True
    while animating:
        amount = steps*FRAME_RATE
        self.image.angle = self.image.angle+amount      # Update the angle
        if abs(self.image.angle-sangle) >= 90:
            self.image.angle = fangle
            animating = False
```

3

But This is Not Going to Work

- **This won't actually draw anything!**
  - This function is a helper to `update()`
  - Keeps running until animation done
  - Method `draw()` only called **at the end**
- Cannot `draw()` inside of `update()`
  - All drawing must be at **same time**
  - What about all the other animations?
- Need some way to "break up" the loop

4

Subroutines vs Coroutines

Subroutine

- Runs until completed
  - Invoked by parent routine
  - Runs until reach the end
  - Returns output to parent
- Just like a function call
  - Parent is "frozen"
  - Subroutine/function runs
  - Parent resumes when done

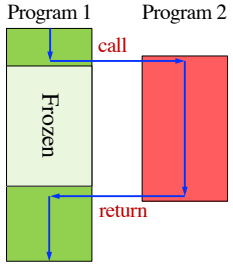
Coroutine

- Can stop and start
  - Runs for a little while
  - Returns control to parent
  - And then picks up again
- *Kind of* like a generator
  - Starts up at initial call
  - Can yield execution
  - Resumes with full state

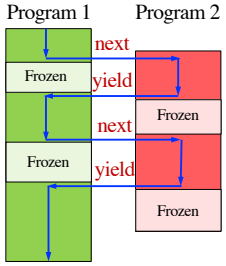
5

Subroutines vs Coroutines

Subroutine



Coroutine



6

Same Animation with Generator

```
def _animate_turn(self,direction):
    """Animates a rotation of the image over SPEED seconds"""
    sangle = self.image.angle
    fangle = sangle+90 if direction == 'left' else sangle-90
    steps = (fangle-sangle)/ANIMATION_SPEED        # Compute degrees per second
    animating = True
    while animating:
        amount = steps*FRAME_RATE
        self.image.angle = self.image.angle+amount      # Update the angle
        if abs(self.image.angle-sangle) >= 90:
            self.image.angle = fangle
            animating = False
        yield                                           # Pause to draw
```

Add this one line

7

Also Need to Drive The Animation

```
def update(self,dt):
    """Animates the image."""
    if not self._animator is None:        # Something to animate
        try:
            next(self._animator)          # Step animation forward
        except StopIteration:
            self._animator = None         # Stop animating

    elif self.input.is_key_down('left'):  # Start animation on press
        self._animator = self._animate_turn('left')
    ...
```

8

Generators Have a **send** Method

- Generators have a send() method
  - a = mygenerator()
  - b = next(a) # progress and get a value
  - a.send(val) # sends a value back
- Sends to a **yield expression**
  - **Format**: (yield) # parentheses are necessary
  - Typically used in an assignment
  - **Example**: value = (yield)

9

Can Do Both Ouput and Input

- **Format:** var = (yield expr)
  - Coroutine evaluates expr and outputs it
  - Coroutine stops and lets parent resume
  - When coroutine resumes, new value in var
- **Example:**

```
def give_receive(n):
    """Receives n values as input and prints them"""
    for x in range(n):
        value = (yield x)
        print('Received '+repr(value))
```

10

Animation Smoothing with Coroutines

```
def _animate_turn(self,direction):
    """Animates a rotation of the image over SPEED seconds"""
    sangle = self.image.angle
    fangle = sangle+90 if direction == 'left' else sangle-90
    steps = (fangle-sangle)/ANIMATION_SPEED        # Compute degrees per second
    animating = True
    while animating:
        dt = (yield)                                # Get time to animate
        amount = steps*dt
        self.image.angle = self.image.angle+amount      # Update the angle
        if abs(self.image.angle-sangle) >= 90:
            self.image.angle = fangle
            animating = False
```

Get the current dt as input each time

11

Parent Code Also Needs Tweaking

```
def update(self,dt):
    """Animates the image."""
    if not self._animator is None:        # Something to animate
        try:
            self._animator.send(dt)       # Tell it secs to animate
        except:
            self._animator = None         # Stop animating
    elif self.input.is_key_down('left'):
        self._animator = self._animate_turn('left')
        next(self._animator)              # Start up the animator
        ...
```

12