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

```python
def _animate_turn(self, dt):
    """Animates a rotation of the image over SPEED seconds""

    steps = (self._fangle-self._sangle)/SPEED
    amount = steps*FRAME_RATE
    self.image.angle = self.image.angle+amount
    if abs(self.image.angle-self._sangle) >= 90:
        self.image.angle = self._fangle
        self._animating = False
```

Wouldn’t a Loop Be Simpler?

```python
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
    animating = True
    while animating:
        amount = steps*FRAME_RATE
        self.image.angle = self.image.angle+amount
        if abs(self.image.angle-sangle) >= 90:
            self.image.angle = fangle
            animating = False
```

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

Subroutines vs Coroutines

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Coroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runs until completed</td>
<td>Can stop and start</td>
</tr>
<tr>
<td>* Invoked by parent routine</td>
<td>* Runs for a little while</td>
</tr>
<tr>
<td>* Runs until reach the end</td>
<td>* Returns control to parent</td>
</tr>
<tr>
<td>* Returns output to parent</td>
<td>* And then picks up again</td>
</tr>
<tr>
<td>* Just like a function call</td>
<td>* Kind of like a generator</td>
</tr>
<tr>
<td>* Parent is “frozen”</td>
<td>* Starts up at initial call</td>
</tr>
<tr>
<td>* Subroutine/function runs</td>
<td>* Can yield execution</td>
</tr>
<tr>
<td>* Parent resumes when done</td>
<td>* Resumes with full state</td>
</tr>
</tbody>
</table>

Subroutines vs Coroutines

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Coroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>Program 2</td>
</tr>
<tr>
<td>Frozen</td>
<td>call</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Coroutine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>Program 2</td>
</tr>
<tr>
<td>Frozen</td>
<td>yield</td>
</tr>
<tr>
<td>Frozen</td>
<td>next</td>
</tr>
<tr>
<td>Frozen</td>
<td></td>
</tr>
</tbody>
</table>
Same Animation with Generator

```python
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 * FRAME_RATE * dt  # Update the angle
        self.image.angle = self.image.angle + amount
        if abs(self.image.angle - sangle) >= 90:
            self.image.angle = fangle
            animating = False
```

Also Need to Drive The Animation

```python
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')

```

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)`

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

Animation Smoothing with Coroutines

```python
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
    Get the current dt as input each time
    while animating:
        dt = (yield)  # Get time to animate
        amount = steps * dt  # Update the angle
        self.image.angle = self.image.angle - amount
        if abs(self.image.angle - sangle) >= 90:
            self.image.angle = fangle
            animating = False
```

Parent Code Also Needs Tweaking

```python
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 StopIteration:
            self._animator = None  # Stop animating
    elif self.input.is_key_down('left'):  # Start up the animator
        self._animator = self._animate_turn('left')
        next(self._animator)  # Start up the animator
```

Add this one line

```
```