

1

### Must We Write this Loop Each Time?

```

while program_is_running:
    # Get information from mouse/keyboard
    # Handled by OS/GUI library
    # Your code (for loop body)
    application.update()
    # Custom Application class with its own attributes
    # OS/GUI handles everything else.
    
```

Method call (for loop body)

- Write loop body in an app class.
- OS/GUI handles everything else.

2

### Programming Animation

Intra-Frame	Inter-Frame
<ul style="list-style-type: none"> <li>Computation within frame                             <ul style="list-style-type: none"> <li>Only need current frame</li> </ul> </li> <li><b>Example:</b> Collisions                             <ul style="list-style-type: none"> <li>Need current position</li> <li>Use to check for overlap</li> </ul> </li> <li>Can use <b>local variables</b> <ul style="list-style-type: none"> <li>All lost at update() end</li> <li>But no longer need them</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Computation across frames                             <ul style="list-style-type: none"> <li>Use values from last frame</li> </ul> </li> <li><b>Example:</b> Movement                             <ul style="list-style-type: none"> <li>Need old position/velocity</li> <li>Compute next position</li> </ul> </li> <li>Requires <b>attributes</b> <ul style="list-style-type: none"> <li>Attributes never deleted</li> <li>Remain after update() ends</li> </ul> </li> </ul>

3

### Designing a Game Class: Animation

```

class Animation(game2d.GameApp):
    """App to animate an ellipse"""
    def start(self):
        """Initializes the game loop."""
        ...
    def update(self,dt):
        """Changes the ellipse position."""
        ...
    def draw(self):
        """Draws the ellipse"""
        ...
    
```

Parent class that does hard stuff

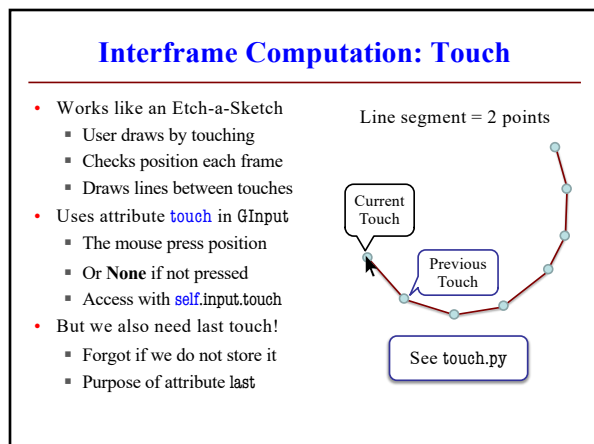
See animation.py

Loop initialization Do NOT use \_\_init\_\_

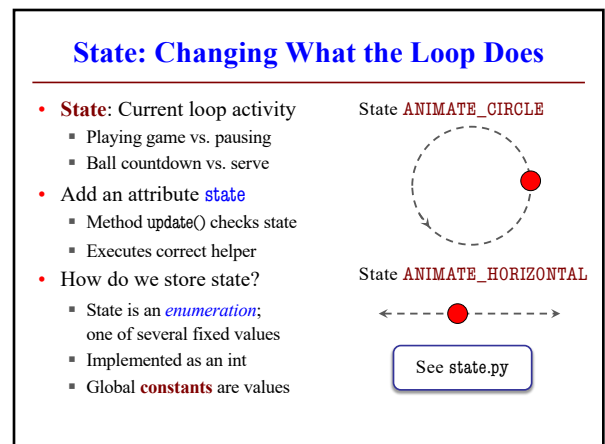
Loop body

Use method draw() defined in GObject

4



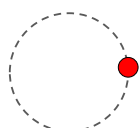
5



6

### States and the Class Invariant

- Think of each state as a mini-program
  - Has its own update functionality/logic
  - Usually separated out as helper to update
  - update uses ifs to send to correct helper
- Need to include in the **class invariant**
  - Some attributes only used in certain states
  - What values must they have in *other* states?
- Also need rules for when we switch states
  - Could be the result of an *event* (e.g. game over)
  - Could be the result of an *input* (e.g. a key press)



See state.py

7

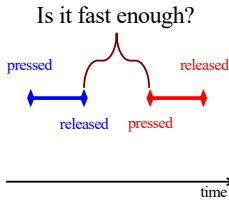
### Checking Input

Keyboard	Mouse/Touch
<ul style="list-style-type: none"> <li><code>is_key_down(key)</code> <ul style="list-style-type: none"> <li>Returns True if key is down</li> <li><code>key</code> is a string ('a' or 'space')</li> <li>Empty string means <i>any</i> key</li> </ul> </li> <li><code>is_key_pressed(key)</code> <ul style="list-style-type: none"> <li>Returns True if key pressed</li> <li><code>key</code> <b>not</b> down prev. frame</li> </ul> </li> <li><code>is_key_released(key)</code> <ul style="list-style-type: none"> <li>Returns True if key released</li> <li><code>key</code> was down prev. frame</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li><code>touch</code> <ul style="list-style-type: none"> <li><b>Attribute</b> giving a position</li> <li>Stored as a Point2 object</li> <li>But None if no touch</li> </ul> </li> <li><code>is_touch_pressed()</code> <ul style="list-style-type: none"> <li>True if touch pressed</li> <li>touch was None prev. frame</li> </ul> </li> <li><code>is_touch_released()</code> <ul style="list-style-type: none"> <li>True if touch released</li> <li>touch <b>not</b> None prev. frame</li> </ul> </li> </ul>

8

### Complex Input: Click Types

- Double click = 2 fast clicks
- Count number of fast clicks
  - Add an attribute `clicks`
  - Reset to 0 if not fast enough
- Time click speed
  - Add an attribute `time`
  - Set to 0 when mouse released
  - Increment when not pressed (e.g. in loop method `update()`)
  - Check time when next pressed

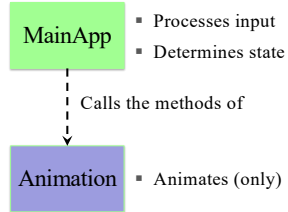


See touch.py

9

### Designing Complex Applications

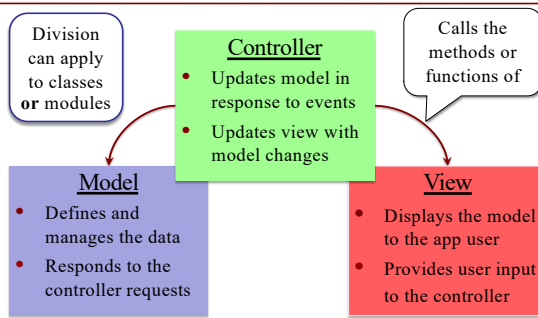
- Applications can become extremely complex
  - Large classes doing a lot
  - Many states & invariants
  - Specification unreadable
- Idea:** Break application up into several classes
  - Start with a "main" class
  - Other classes have roles
  - Main class delegates work



See subcontroller.py

10

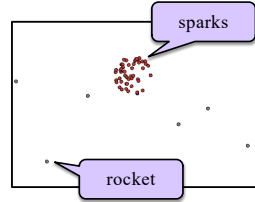
### Model-View-Controller Pattern



11

### Models in Assignment 7

- Often subclass of `GObject`
  - Has built-in draw method
- Includes groups of models
  - Example:** rockets in `pyro.py`
  - Each rocket is a model
  - But so is the entire list!
  - `update()` will change both
- A7:** Several model classes
  - Ship to animate the player
  - Alien to represent an alien



See pyro.py

12