While-Loops are Fundamental

- They are behind all **long-running applications**
  - Always have a top level while loop
  - And many subtasks are themselves while loops
- The challenge is how these loops fit together
  - Very often you are not writing all of the code
  - Your code has to fit inside "invisible" loops
  - Your code can cause an infinite loop you never saw
- To understand this, we will look at a **case study**

1

The Game of Pig: A Random Game

- Play progresses clockwise
- On your turn, throw the die:
  - If roll 1: lose turn, score zero
  - Anything else: add it to score
    - Can also roll again (and lose)
    - If stop, score is "banked"
- First person to 100 wins



2

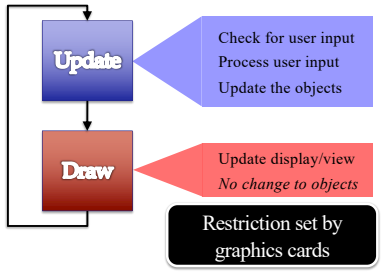
Advantage of Class-Based Applications

- **Function calls** correspond to **call frames**
  - Any work is lost when the game is complete
  - What if you want to keep statistics?
- **Objects** live in the **heap**
  - Any attributes will continue to persist
  - Still have functions (methods) to do stuff
- **Idea**: **Refactor** our function-based apps
  - Turn some local variables into attributes
  - But not all – only those we want to "save"

3

Understanding a GUI Application

Animates the application, like a movie



4

We Have to Change Our Thinking

```
while program_is_running:
    # Get information from mouse/keyboard
    # (Handled
    # Your code go
    application.update()
```

Local variables are **erased** each loop step!

If something needs to survive multiple steps it must be an **attribute**

5

Attributes = Loop Variables

Normal Loops

```
x = 0
i = 2
# x = sum of squares of 2..i-1
while i <= 5:
    x = x + i*i
    i = i +1
# x = sum of squares of 2..5
```

Variables "external" to the loop body

Application

Attributes are the "external" variables

```
while program_running:
    # Get input
    # Your code called here
    application.update()
    # Draw
```

6

Designing a Game Class: Animation

```
class Animation(game2d.GameApp):
    """App to animate an..."""

    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

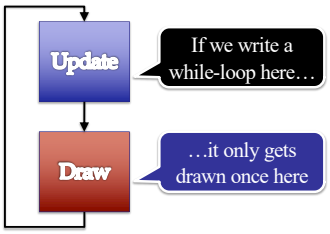
Loop initialization
Do NOT use __init__

Loop body

Use method draw() defined in GObject

7

Understanding a GUI Application

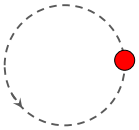


8

Solution: Application State

- **Idea:** *Interleave* loops
  - Write body for each loop
  - Pick right body in update
- **State**: Current loop body
  - Playing game vs. pausing
  - Choice of animation
- Add an attribute state
  - Method update() checks it
  - Executes correct helper

State STATE_CIRCLE



State STATE_HORIZONTAL



See state.py

9

How Do We Represent State?

- Use an **enumeration**
  - List of fixed values
  - Each value a variable
  - Variable is a state name
- In Java/C this is a type
  - But we use int in Python
  - Achieves the same result
- Affects state **invariant**
  - Should not be any int
  - Should be in the list

```
# state before the game has started
STATE_INACTIVE = 0

# state when loading a new wave
STATE_LOADING  = 1

# state when the wave is in play
STATE_ACTIVE   = 2

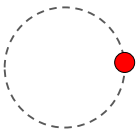
# state when we pause the game
STATE_PAUSED   = 3

# state when the game is complete
STATE_COMPLETE = 5
```

10

When Do We Switch State?

- We switch for one of two reasons
  - Can be the result of an *event* (e.g. game over)
  - Can be the result of an *input* (e.g. a key press)
- Checks for this at the start or end of update()
  - At the **beginning** if **input driven**
  - At the **end** if **event driven**
- **Examples:**
  - Refactored pig3.py is it at the end
  - Animated state.py at the beginning



11

Checking Input

Keyboard

- is_key_down(key)
  - Returns True if key is down
  - key is a string ('a' or 'space')
  - Empty string means *any* key
- is_key_pressed(key)
  - Returns True if key pressed
  - key **not** down prev. frame
- is_key_released(key)
  - Returns True if key released
  - key was down prev. frame

Mouse/Touch

- touch
  - **Attribute** giving a position
  - Stored as a Point2 object
  - But None if no touch
- is_touch_pressed()
  - True if touch pressed
  - touch was None prev. frame
- is_touch_released()
  - True if touch released
  - touch **not** None prev. frame

12