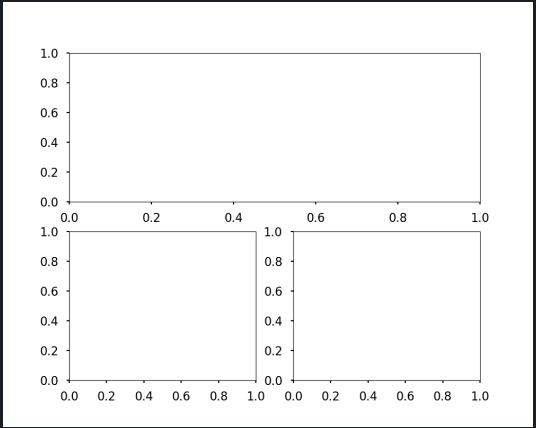
Announcements

- Homework
  - Homework 10 posted and due on Friday!
  - You are going to start getting a bunch of graded things back this weekend. I'm almost through my backlog in my other classes. Soooo sorry for the delay. :(
- Project info coming Friday
  - Poll will be going out to you to get some feedback on potential groups
- Still figuring out exactly what the final is going to look like as well
- Polling: `rembold-class.ddns.net`

Review Question

Which block of code to the right will produce the arrangement of axes below?



A)
```
ax1 = f.add_subplot(222)
ax2 = f.add_subplot(221)
ax3 = f.add_subplot(212)
```

B)
```
ax1 = f.add_subplot(223)
ax2 = f.add_subplot(221)
ax3 = f.add_subplot(122)
```

C)
```
ax1 = f.add_subplot(224)
ax2 = f.add_subplot(223)
ax3 = f.add_subplot(211)
```

D)
```
ax1 = f.add_subplot(213)
ax2 = f.add_subplot(211)
ax3 = f.add_subplot(222)
```

Solution: C

A Classy Arcade

- We've been using `arcade` throughout the semester, but not to its full potential
- Like Matplotlib, Arcade is comprised of multiple classes that define specific objects
- We can inherit from those objects and then make small changes to get large amounts of functionality with relatively little effort!

Window to the Future

- The primary class we can inherit from and use is `arcade.Window`
  - The same class that was being formed when you used to call `arcade.open_window()`
- The window class has a huge amount of predefined methods that we can override to provide almost any sort of flexibility
  - `on_draw` for basic drawing
  - `on_update` for animation
  - Methods to get keyboard input
  - Methods to get location and input from the mouse
  - Methods to control what happens if the window is resized
  - etc

Same Game, New Tricks

Previously...

```
import arcade

arcade.open_window(
            500,
            500,
            'Hi')

arcade.start_render()
arcade.draw_circle_filled()
arcade.finish_render()

arcade.run()
```

Now...

```
import arcade as arc

class MyPicture(arc.Window):
   def __init__(self):
      arc.Window.__init__()
      arc.run()
```

Same Game, New Tricks

Previously...

```
import arcade

arcade.open_window(
            500,
            500,
            'Hi')

arcade.start_render()
arcade.draw_circle_filled()
arcade.finish_render()

arcade.run()
```

Now...

```
import arcade as arc

class MyPicture(arc.Window):
   def __init__(self):
      arc.Window.__init__()
      arc.run()
```

- So initially this is just organizing code
- But inheriting from the `arcade.Window` class gives us many more options.

On Draw!

- We can override or redefine the `on_draw` method to specify what should happen whenever the window needs to draw something!
- Generally where you will put all of your `start_render` and `draw` commands
- You won't need to call the method yourself! The class knows what to do!

```
def on_draw(self):
    arcade.start_render()
    arcade.draw_circle_filled(
                        200,
                        200,
                        100,
                        arcade.color.YELLOW)
```

On Update!

- Can override the `on_update` method to control what happens when the window tries to update itself (usually about 60 times a second)
- Generally where animation controls should go
- Again, you won't need to call this method manually!

```
def on_update(self, dt):
    self.x += 1 # Assuming some circle uses self.x
    # Any other animation code you need to update
```

On Key Press!

- Override `on_key_press` to handle what should happen when any key is initially pressed
- `on_key_release` also exists if you want different functionality for that
- Takes two inputs: the key (as an integer code) and a modifier
- Can use the `arcade.key.<some key>` to easily look up the integer codes

```
def on_key_press(self, key, modifier):
    if key == arcade.key.Q:
        arcade.close_window()
```

On Mouse Press!

- Override `on_mouse_press` to handle anything that should happen when any mouse button is clicked
- Also options for release, drag, motion, etc
- `on_mouse_press` takes 4 arguments: x,y,button, and modifier
- Button codes are available from `arcade.MOUSE_<something>`

```
def on_mouse_press(self, x, y, button, modifier):
    if button == arcade.MOUSE_BUTTON_LEFT:
        print('Click!')
```