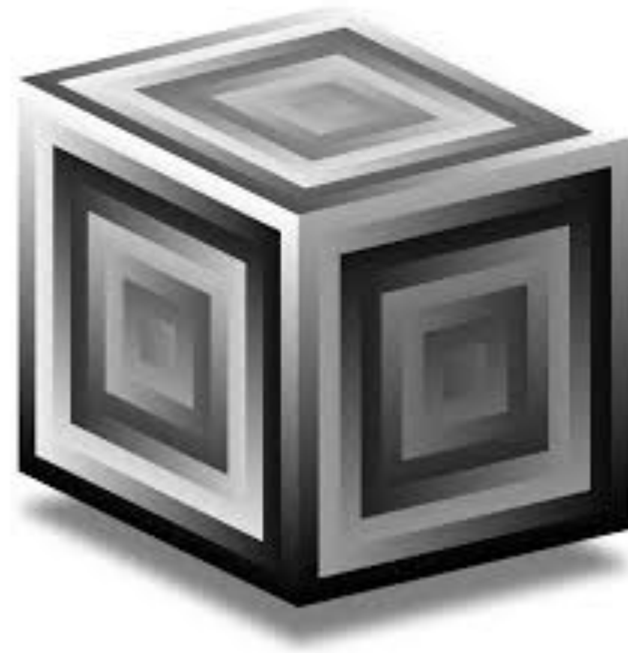
GUI
(Graphical User Interface)

Topics Addressed

- GUI
- Windows/Drawing Basics
- Views
  - Slider
  - Knob
  - StaticText
  - Button
  - RangeSlider
- Decorators
- Layouts



Event-Driven Programming

- Event-Driven programming is a programming paradigm in which the control flow of a program is dictated by user interaction.
  - This is nearly all programs we use on a daily basis
- Control flow is dictated by user interaction on the screen: clicking of buttons, selection from menus, dragging, typing, etc.
- Event-Driven programming is not necessarily restricted to user input. Hardware sensors, program interaction, etc. all use the paradigm.

GUI - Definition

- GUI stands for Graphical User Interface. A GUI is a graphical window or interface that allows users to cleanly and intuitively interact with some technological/electronic component. Examples of GUIs:
    - Desktop
    - Finder
    - Program Interfaces like Microsoft Word/Excel/PowerPoint or Google Maps
    - Most things you interact with on your computer have a GUI
- Concept of GUI developed at Xerox in 1981 to eliminate the need for command line interaction.
- How do you interact with a GUI?
    - Mouse
    - Keyboard
    - Touchscreen

Displaying Information

- All machines place data on the screen by interacting with the computer's CPU/operating system and graphics card/GPU.
- Graphical data is constructed by a program. Then, the program triggers the operating system to take over and transmits the data to your graphics card for rendering and placement in a portion of graphics card memory called a framebuffer for display.
- Managing the interaction between these hardware devices is complicated. Graphics libraries and toolkits help abstract away these details.
- SuperCollider uses Qt, a free and open-source toolkit written in C++, to handle creating GUI windows. More info here: https://wiki.qt.io/About_Qt

Windows

- Creating a GUI starts by creating an instance of `Window`.
  - The class `Window` is the topmost container of a hierarchy of different GUI objects called views which you can add to the window.
- Each window has a `View` that holds other GUI objects.
- To see all windows created, use the class method `Window.allWindows`
- Useful instance methods:
  - `.front` -> brings the window to the front of your screen
  - `.minimize` -> place your window in the taskbar, dock… etc.
  - `.fullScreen`
  - `.close` -> closes and destroys the window

Windows

```
(
w = Window.new("GUI Introduction");
w.front; // Ensures that the screen is on top so we can see immediately
)

Window.allWindows; // An array of all existing windows
w.close; // Hides and destroys (i.e., frees) the window
```

There are several optional parameters for creating a window:

- 1st argument – the title of the window to be displayed at the top of the window.
- 2nd argument – dimension/location (by default 400x400 pixels in the center of the screen)
- 3rd argument – Boolean expressing if the window is resizeable
- 4th argument – Dummy argument that should be ignored (used for compatibility with SwingOSC)
- 5th argument – Boolean expressing if the window should be scrollable. False by default

The dimension/location for the window is expressed with another class called a `Rect`. More shortly.

Drawing On A Window

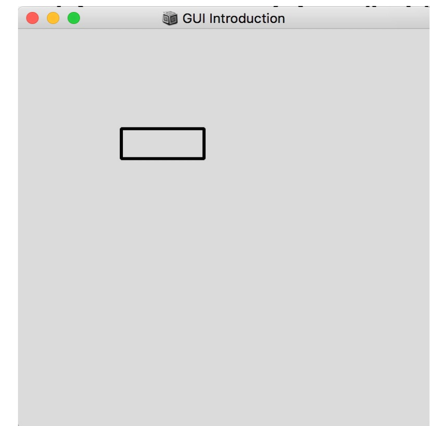
- The class Window has an instance method `.drawFunc` that is used when the window is refreshed to draw objects on the screen.
- The objects are drawn with a class called `Pen`, which has numerous class methods that can draw all sorts of figures (termed paths) on the window.
- Pen has two very important class methods: .fill and .stroke.
  - `.fill` -> fills the object with the color specified by `Pen.color` (black, by default)
  - `.stroke` -> outlines the object with a border specified by `Pen.width`
  - If you fail to use either of these methods, then nothing will be displayed on your screen!
    - Note: there are some class methods which handle this for you. Be sure to read the documentation.
- When objects are placed on a window or any subcontainer in a window, coordinates are relative to the upper left corner of the window. Positive y-coordinates go **down**.

Rect

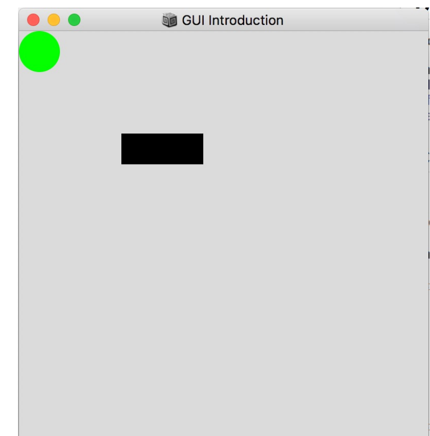
- Rectangles are some of the simplest figures we can draw but also are useful as a way of expressing dimensions for other GUI objects.
- Rectangles take four arguments:
  - 1st argument: left -> expresses how far from the left the rectangle should be placed.
  - 2nd argument: top -> expresses how far from the top the rectangle should be placed.
  - 3rd argument: width
  - 4th argument: height
- When placing rectangles in windows or other GUI objects, these coordinates are relative to the upper-left corner.

Simple Drawing Code

```
// Draw a rectangle
// Coordinates are relative to the upperleft corner of the window
~r1 = Rect(100, 100, 80, 30);
w.drawFunc = {
    Pen.addRect(~r1); // A path for a rectangle
    Pen.width = 3;
    Pen.stroke; // Add a stroke (i.e., border) to most recent path
};
w.refresh; // Need to redraw the window
```




```
// Circles/Ovals are defined by a rectangle that can enclose it
~r2 = Rect(0, 0, 40, 40);
w.drawFunc = {
    Pen.fillRect(~r1); // Add and fill a rectangle
    Pen.addOval(~r2); // Add an oval to the screen
    Pen.color_(Color.green); // Set the pen color to green
    Pen.fill; // Fill the most recently added path (i.e., the oval)
};
w.refresh;
```




```
// Clear the screen from drawings
w.drawFunc_(nil); // No draw function
w.refresh;
```

Views

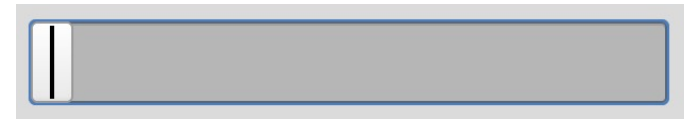
- In SuperCollider, views are the GUI objects that are placed on windows.
  - Some views (called containers) are capable of holding other views.
    - Example: Each window has a view that is used to hold all of the GUI objects within itself
  - When a view "A" holds a view "B", we say that "A" is the parent view and "B" is the child view.
- All views inherit from the basic `View` class. The view occupies a rectangular space of the window within which it draws itself to display some data or to indicate a mode of interaction between the user and the program.
- Views receive keyboard and mouse events generated by the user and respond to them by controlling the behavior of the program. They also display information about the state of the program and the data on which it operates.

Slider

- The class `Slider` is a view for a moveable slider that can be mapped to values for some other aspect of your program.
- The values for a slider can be got/set through `.value` method
- Mapping values to a slider can be written with an action function via the `.action` method.
  - Every view has an action method. The action method allows for the view to be passed in as an argument so that the data can be retrieved from it.
- See lecture code for more examples of sliders and mapping to SynthDefs.

```
(
~s1 = Slider.new(w, Rect(10, 10, 100, 40));
~s1.action = {
      arg view;
      view.value.postln;
};
w.front;
)
```

Posts the value of the slider to the screen. Any update to the slider triggers the action method.



Picture of a slider

Knob

- The class Knob, like Sliders, ranges from 0 to 1.0.
- Knobs, like Sliders, have an `.action` method and the value of the slider can be extracted with `.value`.
- Knobs are controlled with mouse motion, either circular or linear. This property can be set to `.mode`.

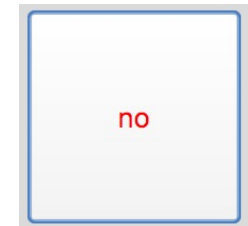
StaticText

- The class `StaticText` is the quick and dirty way to put a view for text inside windows.
- Like many other views, the bounds for static text is defined through an instance of the class Rect.
- See documentation for the useful instance methods that can change font, size, background color, text color… etc.

```
~staticText = StaticText.new(w, Rect(10, 110, 100, 30));
~staticText.string = "This is some text"; // set the text
~staticText.align = \center; // center the text
~staticText.stringColor = Color.blue; // Set the text color to blue
~staticText.background = Color.grey; // Set the background to grey
```

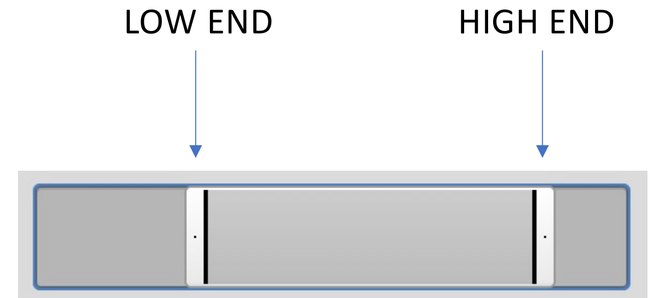
Button

- The class `Button` provides any number of n-clickable states.
  - To set the states of the button, you must use the instance method .states to set the states with an array of arrays.
  - Each inner array is a three element array of text, text color and background color.
  - Each of the *n* inner array represents a state of the button as integer indexed from 0 to *n* - 1.
- Like other views, you must provide the button a parent view like a window's `TopView` and optionally a bound defined through `Rect`.
- A button's `.action` method also passes the view as an argument. The view's instance method `.value` provide the integer state.



RangeSlider

- The class `RangeSlider` is a more sophisticated slider (really a composite of two sliders) that constitutes a low end and high end.
  - The low end and high end provide a range which is the difference between the two values
  - The low end and high end range between 0 and 1.0
- Range sliders can be useful for visually representing audio data that relies upon ranges like bandpass filters, notch filters, … etc.
  - Can also be useful for restricting synthesizers to certain ranges of pitches if the values of the slider are mapped, to say, midi note numbers
- The values of the slider can be extracted with the methods `.lo` for the low end, `.hi` for the high end, and `.range` for the range between the low end and high end.



Defining Window Bounds

- The bounds of a window can be passed in as a rectangle.
- IMPORTANT: the rectangle passed in defines a bounds relative to the **lower** lefthand corner of the screen.
  - Recall that for all views placed inside a window, the bounds are relative to the **upper** lefthand corner
- How to consider the arguments to the rectangle passed into Window
  - 1st argument: distance from the left boundary of the screen
  - 2nd argument: distance from the **bottom** boundary of the screen
  - 3rd argument: width of the window
  - 4th argument: length of the window
- To see the boundaries of the screen, use `Window.screenBounds`

View Resizing

- Views can automatically resize or move when their parent is resized, in one of the nine different ways that define how each of the view's edges will move along with the parent's edges.
  - This parameter is set through a view’s `.resize` method.
- Relevant documentation can be found by looking at the document Resize Behaviour.

```
w = Window.new("GUI Introduction", Rect(200,200,200,200));
TextField.new(w, Rect(0,0,200,30)).resize_(2);
Slider.new(w, Rect(0,30,30,170)).resize_(4);
w.front;
```

1 - fixed to left, fixed to top
2 - horizontally elastic, fixed to top
3 - fixed to right, fixed to top
4 - fixed to left, vertically elastic
5 - horizontally elastic, vertically elastic
6 - fixed to right, vertically elastic
7 - fixed to left, fixed to bottom
8 - horizontally elastic, fixed to bottom
9 - fixed to right, fixed to bottom

CompositeView

- The class Window contains a container view called TopView
  - Container views are views that are capable of holding other views
  - The TopView class is only for Window. If you want a parent view to organize several children views, use the class CompositeView
- Properties of CompositeView:
  - It is the parent of any number of children views
  - It can be the child of other container views, like Window's view or other CompositeView
- Advantages:
  - CompositeView is a way to organize views and to abstract and modularize GUI objects
  - Methods for composite views can often cleanly and succinctly change its child views.
- Composite views do not have any special/unique methods. They are simply a container/organizer for other views.

Decorators

- We have seen how views can be arranged by hardcoding position and sizes within container views like `Window`'s container view `TopView` and `CompositeView`.
- Decorators automatically handle the positioning of views within a container view
- There is **one** decorator in SuperCollider called `FlowLayout`.
- A decorator can be assigned to a view through the view's `.decorator` method or by using the instance method `.addFlowLayout` for `CompositeView` or `Window`.

Layouts

- Layouts offer a middle ground, providing the user more flexibility than a decorator but automating some of the process.
- Layouts distribute the amount of space given to the view on which they are installed among the children of that view.
- Types of layouts:
  - `LineLayout`: places views either horizontally or vertically. Need actually to use the child classes `VLayout` and `HLayout` for vertical and horizontal layouts, respectively. Do not use `LineLayout`.
  - `GridLayout`: place views in a grid system that can hold views, other layouts or nothing
  - `StackLayout`: place views on top of each other and control the visibility of each.
- Layouts can be added to a view by setting the layout with `.layout` method for the specific view.
- You should **not** combine decorators and layouts. Behavior is undefined.