```
###

    Introduction to:

        • Framer Layers
        • Basic event handling

    Preface:

        FramerJS is a library.
        It contains a useful collection of classes and functions for doing UI prototyping.
        Quiet a bit of what's covered here is specific to FramerJS, opposed to programming in general,
        but FramerJS uses techniques, patterns, idioms, or approaches that are similar to other
        UI frameworks such as Apple's UIkit.

###
```

```
######################## P1
# Layers are a powerful FramerJS class used for making visuals.
# Layer instances have properties like x, y, width, height, backgroundColor, etc.
# Think of layer instances as customizable rectangles.
# When we set of modify their properties, the visuals change automatically.

# We create new layers like this.
layer1 = new Layer
layer2 = new Layer


# Layers can be positioned by setting their x and y properties.
layer2.x = 100
layer2.y = 200
# Note: The coordinate 0,0 is in the top left corner,
# with y positive going down and x positive going right.

# Layers can be sized by setting their width and height properties.
layer2.width = 50
layer2.height = 200

# Layers have a backgroundColor property.
# It can be set to a String that contains any valid CSS color value.
layer1.backgroundColor = "white"
layer2.backgroundColor = "rgb(255,0,0)"

# A full list of Layer properties can be found in FramerJS's documentation (cmd+D)

# Technical Note:
# When we create a Layer, FramerJS creates a DIV element in the document.
# FramerJS then uses in-line CSS style properties to control what the DIV looks like.
# It uses absolute positioning so that we can ignore normal flow (the default way elements are placed).
# When we modify a layer's properties FramerJS updates the in-line styles.
# In MOST cases we will never need to access the underlying element.
# We can inject raw HTML into the Div by setting the layer's html property.
#   e.g.   myLayer.html = "<a href="google.com">Link</a>""


```

```
######################## P2
# We can set many of the layer instance's properties right when we
# create it by passing an object with the settings we'd like.

# Here is the very explicit version:
layer1 = new Layer({x:100,y:100,backgroundColor:"black",width:10,height:10})

# It is much more common to use shorthand when constructing a layer in this way.
# For example:
layer2 = new Layer
    x:10
    y:100
    backgroundColor:"orange"
    width:20
    height:20
```

```
######################## P3
# Layers can be nested. Sublayers are called "children".
# Layers that contain children are called "parents".
# Sublayers are positioned based on the top left of the parent layer, not the screen.

container = new Layer
    x:50
    y:50
    width:100
    height:100
    backgroundColor: "black"

sublayer = new Layer
    x:10
    y:10
    backgroundColor:"red"
    opacity:0.5 # Note: opacity goes between 0 and 1

sublayer.parent = container     # Note that sublayer is at x:10,y:10 based on the parent's top left corner.
                                # Note that container's background color doesn't affect the sublayer.
                                # Note that the sublayer is visually on top of container's background color.

print container.height          # Note that the container's width and height are not affected by sublayers.
print container.width

# container.clip = true        # Uncomment this line and watch how the sublayer is clipped to the container's size.
```

```
#######################
####################### (Layer) Events
#######################

###

    Preface

        User interface development is all about running code when something happens.
        Most systems respond to input by "generating events" and/or by calling a
        function that the programmer has set to "handle" the event.

        Most systems will generate events when:

            • A mouse button is pressed down or released.
            • A keyboard button is pressed down or released.
            • The internal clock updates.

        Most user interface frameworks will generate events in more specific cases like:

            • When a cursor moves over or off of a UI element.
            • When a mouse button is pressed or released while the cursor is over a UI element.

###
```

```
######################## P4
# Layer instances "generate events" when something like a click happens.
# We write code that "handles" or "listens for" these events.
# Specifically, we "add" event handling functions to our layers using specific layer methods.
# (Sometimes this is called "binding"—it's like we're connecting things together.)
# Functions that are used by a program to respond to events are called "event handlers", "handlers", or "listeners".
# This term refers to how the programmer is _using_ the function.
# By all other accounts an event handler is just a plain old function.

block = new Layer

# Create the event handler function
handleMouseDown = ->
    print("Ouch!")

# To "add an event handler" to a layer we use one of the layer's event binding methods.
# These methods correspond to a kind of event, and accept one argument, the handler function.
# Here we call the method 'onMouseDown' with the argument 'handleMouseDown'
block.onMouseDown(handleMouseDown) # This "binds" the 'handleMouseDown' function to 'mouseDown' events that block may generate.

# Under the hood, FramerJS notes that the 'handleMouseDown' function should be called
# when it detects a mouse press while the cursor is over the 'block' layer.
```

```
######################## P5
# We can create an add event handlers in one step by using a function literal.
# This shortens things a bit.

block = new Layer
block.onMouseDown( -> print("Ow!") ) # Here the function is written out literally.

# We can shorten things further by using function call shortcuts:
block.onMouseUp -> print "Whew"

# We can also use function literals with multiple lines:
block.onMouseOver ->
    print "?"    # Note the indentation!
    print "!"
    print "?"

# This approach is very common in FramerJS.
# It's very brief, but it can be ambiguous if you don't understand that: 'onMouseOver' is
# a "method" being "called" with an "argument" which happens to be a "function literal".
```

```
######################### P6
# When a handler function is called, something magical happens: in the handler function,
# the keyword 'this' is set to refer to the layer instance that generated the event.
# This behavior makes it easy to refer to the object that was acted on.


block = new Layer
    backgroundColor:"black"

block.onMouseDown ->
    this.backgroundColor = "red"    # 'this' will refer to 'block' when the function is called.

block.onMouseUp ->
    @backgroundColor = "black"  # We can also use CoffeeScript's @ shortcut.


######################### P7
# This program creates a button-like entity that responds to the mouse in several ways.

block = new Layer
    backgroundColor:"grey"

block.onMouseOver ->
    @backgroundColor = "orange"

block.onMouseOut ->
    @backgroundColor = "grey"

block.onMouseDown ->
    @backgroundColor = "red"

block.onMouseUp ->
    @backgroundColor = "orange"
```

```
######################## P8
# When an event handler is called the computer tries to pass it an "event object".
# In other words, it tries to call the function with a single argument.
# This event object has many properties with data about what happened exactly.
# If our handler does not accept any arguments our function will not receive the event object.
# To access the event object, our handler must accept one argument.

block = new Layer
    x:100

# The explicit way
downHandler = (eventObject)->
    print "down"
    print eventObject.offsetX  # the x location of the cursor in the layer
    print eventObject.offsetY  # the y location of the cursor in the layer

block.onMouseDown(downHandler)

# A shorter way of doing the same thing.
block.onMouseUp (eventObject)->
    print "up"
    print eventObject.offsetX
    print eventObject.offsetY

# Note that we're still calling the method 'onMouseUp' with
# one argument: a function that itself accepts one argument.
```

```
######################## P9
# In this next example, we want to listen to events on the background.
# To do this, we can create a layer that is the size of the screen.

bg = new Layer
    width:Screen.width
    height:Screen.height
    backgroundColor:"skyblue"

box = new Layer
    width:10
    height:10
    backgroundColor:"white"

bg.onMouseDown (e)->
    box.midX = e.offsetX
    box.midY = e.offsetY


########################
######################## End
########################
###

    Epilogue.

        Events are not unique to CoffeeScript and FramerJS.

        The events that your handlers respond to are the same as, if not similar to, the events that
        a pages html elements generate.

        While the syntax may be different when using raw HTML+JavaScript, you can still create
        interactivity by creating event handler functions and binding them to events from specific
        elements in the document.


###
```