TI-nspire™

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 11**

Scripting Tutorial - Lesson 11: Advanced: Introducing Classes

Download supporting files for this tutorial

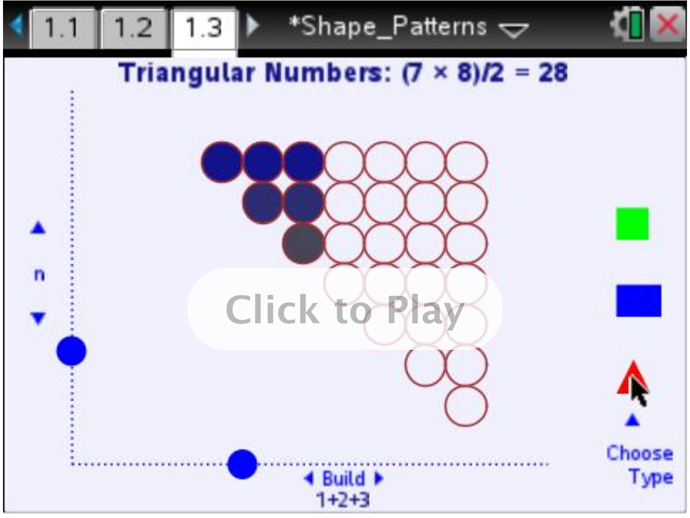
Texas Instruments TI-Nspire Scripting Support Page

In lesson 9 and lesson 10 we created a workable document for visualizing shape numbers. Like many of the documents before this, the user controls the action using arrow keys, enter, escape and tab keys. This sort of keyboard control works very well when using the handheld - it can mean that there is no need for students to have to grab and drag anything - they just start using arrow keys and the result is immediate.

We have already noted, however, that this approach is useless if the document is intended for use with the Player. We have retained TI-Nspire native sliders to support such use, but this writing of variables back and forth between Lua and Nspire is probably not the most efficient way to work in terms of performance and ease of use.

In fact, you might have noticed that a pretty important UI component has been missing in our introduction to Lua to this point - how can we use Lua to control and respond to mouse actions? Clearly, this is the preferred way of operating when using a computer (as opposed to the handheld). Wouldn't it be ideal if documents we developed were actually able to be optimised for all platforms – supporting keyboard control for easy handheld access, and also working with mouse control for use with computers? As something of a bonus, if we no longer need Nspire sliders, then we probably no longer need to transfer variables and can work entirely within Lua, which must be a simpler approach for most problems.

Click to see what such a document might look like



Launch Player

Click on the screen shot shown to view a short video of a document created in this way. Then try it using the TI-Nspire Player by clicking on the red button beneath the image.

In order to realize this goal, we need to move into the next level of Lua scripting and introduce the important and powerful tool of **classes**.

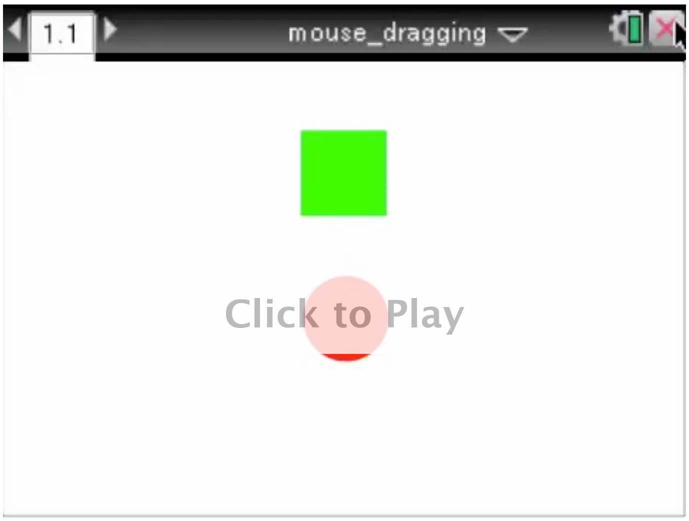
Back to Top

Lesson 11.2: A Class of its Own

We will begin with something a little less ambitious.

Study the video opposite and have a play with the document using the Player. You will see that it is simply two shapes which can be grabbed and dragged around using the mouse – but these shapes can also be selected using TAB and moved using the arrow keys.

Click to see a short video demonstration



The shapes, circle and square, have been defined as **classes**. For the moment, think of a class as more or less a "super-function". Just as we have used functions previously to define all sorts of useful things, that is what we will do with our square and circle. But the power of classes lies in the fact that they bring with them some useful bonus properties. For example, an object defined in this way knows where it is on the screen, it knows whether it has been selected or not, and what color it is meant to be, along with potentially much more. It can tell if it contains another object or coordinate position. Can you begin to see how this could be useful?

Back to Top

Lesson 11.3: Class: init?

Begin by defining the empty class, Square.

Next, the class must be initialized. The various properties that this class is to possess are defined here. Our class Square has position (x and y coordinates), dimensions (width and height), color and the knowledge of whether it has been selected or not (this will become clearer soon.)

If we are to control the position of our

Square with a mouse, then we need to know when we click inside the Square. This is defined by the "contains" function. The function "contains" takes as input an ordered pair (x, y) and returns a Boolean value true or false if the ordered pair falls within the bounds of the Square.

Finally, we need to paint the Square to the screen.

In the usual way, this will require graphics context commands (gc). The first defines the color for this object, and an interesting approach is used here. At the beginning of the script, color is defined as a table as follows:

```
Color = {
        red = {0xFF, 0x00, 0x00},
        green = {0x00, 0xFF, 0x00},
}
```

Since the color of Square has already been specified (in the init function),

```
Square = class()

function Square:init(x, y, width, height)
        self.x = x
        self.y = y
        self.width = width or 20
        self.height = height or 20
        self.color = Color.green
        self.selected = false
end
```

```
function Square:contains(x, y)
        local sw = self.width
        local sh = self.height
        return x >= self.x – sw/2 and x <= self.x + sw/2 and
                y >= self.y – sh/2 and y <= self.y + sh/2
end
```

```
function Square:paint(gc)
        gc:setColorRGB(unpack(self.color))
        gc:fillRect(self.x – self.width / 2, self.y – self.height / 2, self.width, self.height)
        if self.selected then
                gc:setPen("medium","smooth")
                gc:setColorRGB(0, 0, 0)
```

the **unpack** command simply grabs the RGB definition for green from the table, color. (In more detail: The "unpack" function takes as input a table and returns each table element as multiple return values. "gc:setColorRGB" expects three parameters for red, green, and blue, but Color.green is one value, a table of three elements. "unpack" turns the elements of the table into the three parameters expected by setColorRGB.)

Draw the square in the usual way – notice the "self" references used throughout these definitions. This is a simple and effective way for a class object to refer to its own properties.

Finally, a little routine that draws a black border around the square IF it is selected. Neat.

Back to Top

So now how do we see this Square that we have defined?

First, we need to actually call the function Square

```
            gc:drawRect(self.x – self.width / 2, self.y – self.height / 2, self.width, self.height)

        end

    end
```

along with some parameters. Remember that the init routine required x and y coordinates, width and height (even though these last will be the same for a square).

```
Sq = Square(80, 80, 40, 40)
```

Then all that remains is to use the old **on.paint(gc)** function and to call the paint routine that we have defined for this class. We now have our square displayed. NOTE that it will, at present, just sit and look at you – we have not scripted any instructions to make things happen just yet.

```
function on.paint(gc)

        Sq:paint(gc)

end
```

Next we will learn how to make things happen with it.

This seems like a reasonable place to stop for this tutorial. We have introduced this key idea of classes and shown how we might create and display something in this way. Before the next lesson you might try this out and then define your own class to draw a red circle.

In our next lesson we will see how to control such an object using mouse commands

Back to Top

Home ← TI-Nspire Authoring ← TI-Nspire Scripting HQ ← **Scripting Tutorial - Lesson 11**