Getting Started with JS-Eden

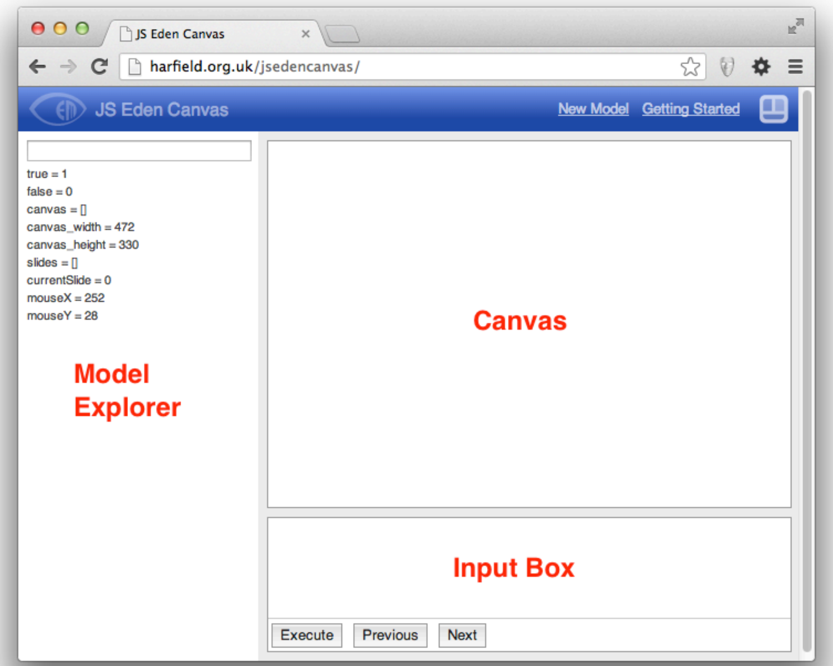
This activity will guide you through an introduction to JS-Eden.

Starting JS-Eden

You can open JS-Eden by pointing your web browser to:

http://harfield.org.uk/jsedencanvas/

When you first start JS-Eden, you will see a top toolbar and three parts to the window. On the left is the Model Explorer, on the right is the Canvas, and below this the Input Box.



```
true = 1
false = 0
canvas = []
canvas_width = 472
canvas_height = 330
slides = []
currentSlide = 0
mouseX = 252
mouseY = 28
```

In the JS-Eden environment, we can construct models using the principles of observables, dependencies and agency. Let's get started by showing how to create observables and dependencies through the input window.

The Input Box

The Input Box is the primary place to create the observables, dependencies and agency that makes up a model.

Creating observables

Type in the input box:

```
a = 5;
b = 6;
```

Then click on 'Execute'. To inspect the value of a and b, you can view the values of all observables in the Model Explorer. These are integer observables, here are some other types of observable:

```
myReal = 1.234;
myString = "Hello World";
myList = [myReal, myString];
```

There are these 4 basic types of observables in JS-Eden: integer, real, string, list.

JS-Eden has operators and functions like most programming languages. Here are some mathematical functions to try:

```
x = pow(2,8);
y = sqrt(16);
z = floor(5.9);
```

Task 1: Write the answer and the mathematical meaning of the three functions.

pow(2,8)	**result ....**	**meaning ....**
sqrt(16)	**result ....**	**meaning ....**
floor(5.9)	**result ....**	**meaning ....**

Another function we will use is the random function:

```
r = random();             // Random real between 0 and 1
s = floor(random() * 10); // Random integer between 0 and 9
```

Creating dependencies

As shown above, you create a new observable called c:

```
c = a + b;
```

The observable c should now have the value of a+b. There is another way of calculating a+b, using dependency:

```
d is a + b;
```

Now c and d should have the same value. Try changing the value of a:

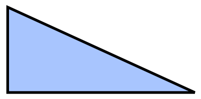
```
a = 1;
```

Are c and d still the same? They are not, because d has changed. It has updated to maintain the 'dependency' of d is a+b. In JS Eden, 'is' is used to create dependencies between observables, much like the dependencies between cells in a spreadsheet.

We say that "c is a plain observable", and "d is a dependency".

Task 2: Can you model a right angled triangle with sides a, b and c? The value of c should always be the hypotenuse (longest side) and should be calculated as a dependency on a and b. Hint: use the sqrt function.

If a = 3 and b = 4, then c = ...



What is your definition of c? c is ....

You can create dependencies on any type of observable!

Try executing the following script:

```
name is "Ant";
welcome is "Hello " // name;
```

Note: the // is string concatenation in Eden.

Now try to change the value of name:

```
name is "James";
```

The Model Explorer

The Model Explorer is the list of observables on the left pane of JS-Eden. When you create a new observable/dependency then you will see it's name and current value shown in the Model Explorer.

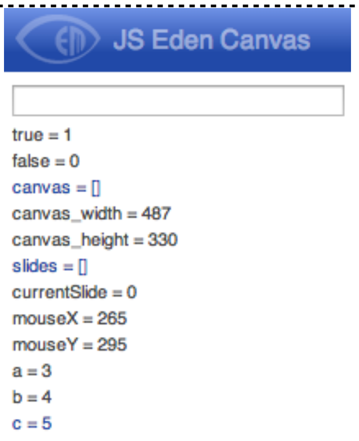
You saw the observables a, b and c being created when you defined them via the Input Box. Note that c shows the current value (not the dependency definition). Dependencies are shown in dark blue to differentiate them from plain observables.

Redefining observables and dependencies

If you want to change an observable value, click on it in the Model Explorer and its definition will be copied to the Input Box.

If you want to see the definition of a dependency or change it's definition, you can also click on it in the Model Explorer and its definition will be copied to the Input Box.

You can change observables and dependencies at any time!



```
true = 1
false = 0
canvas = []
canvas_width = 487
canvas_height = 330
slides = []
currentSlide = 0
mouseX = 265
mouseY = 295
a = 3
b = 4
c = 5
```

The Canvas

The Canvas is currently a large empty white space on the right hand side of the screen. Now that we have a basic knowledge of observables and dependencies, we will explore how to build models using simple graphics.

Creating a line

```
lineA is Line(0, 0, 50, 100);
canvas is [lineA];
```

The first part creates the line observable, with the parameters x1, y1, x2, y2 representing the two points between which the straight line is drawn.

The second part defines the canvas observable as a list containing the line observable. (Without this, you would not see the line in the Canvas.)

As with any observable/dependency, you can modify the line:

```
    lineA is Line(100, 50, 100+10*a, 50);
```

Now the line is dependent on the value of the observable ‘a’.

Task 3: What happens when you change 'a'?

Answer **...**

Create a new line called lineB and add it to the canvas:

```
    lineB is Line(100, 50, 100, 50+10*b);
    canvas is [lineA, lineB];
```

Task 4: Create lineC to make a right-angled triangle with lineA and lineB. When you change the values of a and b then the triangle should change.

lineC is ....

Another optional parameter of Line is the colour:

```
    col is "red";
    lineA is Line(100, 50, 100+10*a, 50, col);
```

Now you can change the col observable:

```
    col is "green";
```

Or you could make the col observable a dependency:

```
    col is a > 10 ? "red" : "green";
```

Task 5: What happens when you change the value of 'a' to be more than 10?

Answer **...**

Explaining the in-line if else

```
answer is [boolean] ? [true_value] : [false_value];
```

Many programming languages have an “in-line if else” construct, but it is rarely used. In Eden we use it very often to create dependencies. The meaning is “if [boolean] is true then the answer is [true_value] else the answer is [false_value].

Text on the Canvas

The Canvas is still quite empty with only lines, so let’s create some text, and then put it on the canvas:

```
textA is Text("Hello folks", 50, 50, "blue");
canvas is [lineA, lineB, lineC, textA];
```

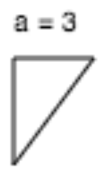
Note: the four parameters for Text are the text string, x position, y position, and text colour.

We can make the text dependent on another observable:

```
textA is Text(stringA, 50, 50, "blue");
stringA = "I am a label";
```

Task 6: Put textA next to lineA and make stringA a dependency on the observable ‘a’. If a has the value of 3 then textA should show “a = 3”.

Answer...



Mouse observables

Take a look at the Model Explorer again and you might notice some observables that you did not create. For example, "mouseX" and "mouseY" are automatically provided by JS-Eden and they correspond to the mouse pointer position in the canvas. Try this:

```
myLine is Line(0, 0, mouseX, mouseY);
```

There are another 2 observables for the last position of the mouse click ("mouseClickX"), and another observable for the current state of the mouse ("mouseDown").

Task 7: Can you make the line turn red when the mouse button is being pressed?

Hint: Use an "in-line if else" expression.

Answer **..**

Task 8: Can you modify the line so that the origin moves to the position of the last mouse click?

Answer **..**

Shapes

There are several different shape type observables you can create.

Circles

To make a circle:

```
myCircle is Circle(100,100,50);
```

The minimum parameters for a Circle consist of x, y, radius. Don't forget to add it to the Canvas. There are 2 optional parameters for Circle: fill colour and outline colour. Try this:

```
myCircle is Circle(mouseClickX, mouseClickY, 50, "orange",
"black");
```

Task 9: Draw a vertical line down the middle of the canvas. If the user moves the circle to the left side of the browser then it should show a yellow circle. If they move to the right then it should be a green circle.

***Answer* ....**

Rectangles

To make a rectange:

```
myRect is Rectangle(20,20,220,100);
```

You will also need to add the rectangle to the canvas. The minimum parameters for a rectangle are x, y, width and height. Rectangle has 2 optional parameters which are similar to the Circle properties: fill colour and outline colour.

Task 10: Can you make a simple game that the user controls the position of a circle by clicking on the Canvas. If the circle is inside a rectangle then colour of the circle will be green, else it is red.

Divs (a special kind of text)

The Text observable is for simple labels. For richer text there is the Div:

```
myDiv is Div("myDiv", 0, 100, 300, 200,
     "<h1>Title</h1><p>Some text</p>", "");
```

Parameters: name, x, y, width, height, html, style.

A clever way to use a Div might be to show the value of several observables, e.g.:

```
myDiv is Div("myDiv", 0, 100, 300, 200,
     "<p>a = "//a//"</p><p>b = "//b//"</p>", "");
```

Remember that double-slash (//) is the string concatenation operator.

Buttons

Another important user interface component is the Button. It is different from other elements in that clicking a button usually involves triggering an action.

To create a button:

```
    myButton is Button("myButton","Press me",50,300,true);
```

Parameters: name, text, x, y, enabled

The first parameter name is required because when there is user interaction with a button then it will update an observable. In the above example, the name is "myButton" and so when the user clicks the button there is an observable "myButton_clicked" that is updated. This means we can write a trigger:

```
    proc myButtonAction : myButton_clicked {
      label = "Button was clicked!";
    }
```

A trigger is (or triggered procedure) can be thought of as an agent that constantly watches one or more observables, and when they change it executes a change to the state of the model.

Consider the following code:

```
buttonState = "Off";
onOffButton is Button("onOffButton", buttonState, 20, 20, true);
canvas is [onOffButton];

proc onOffButton_trigger : onOffButton_clicked {
  if (onOffButton_clicked == 1) {
    if (buttonState == "On")
      buttonState = "Off";
    else
      buttonState = "On";
  }
}
```

Task 11: Create a button that counts the number of times it is clicked. Every time it is clicked, increase the click count by 1. Show the click count on the button.

More exercises

Task 12: Create a circle anywhere on the screen with the coordinates x and y and radius r. Use mouseClickX and mouseClickY to fill the circle yellow when the user has clicked inside the circle.

Task 13: Create a button that moves to a random position when clicked.

Task 14: Can you make a plan of your bedroom including a bed, a table, chairs, windows and doors? Use dependency so that the plan resizes to the size of the canvas.