Introduction to Programming

Andrew Butterfield

Department of Computer Science
School of Computer Science and Statistics
Trinity College, Dublin

January 21st, 2008
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code I).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!

- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code!).
General observations on Project 1:

- I am now sick of “X’s and O’s”!
- Marks awarded for:
  - rendering the grid
  - gameplay
  - recognising win/lose and draw!
- Marking emphasis on final program behaviour, rather than code quality (however, Project 2 marking will pay more attention to the code !).
Noted “Features” of Project 1:

- many prompts per move
- grid not shown at start
- only checking win/lose/draw after both moves
- scrolling issue
  
  ```javascript
  use window.scrollBy after every move
  ```
Noted “Features” of Project 1:

- many prompts per move
- grid not shown at start
- only checking win/lose/draw after both moves
- scrolling issue
  
  use `window.scrollBy` after every move
Noted “Features” of Project 1:

- many prompts per move
- grid not shown at start
- only checking win/lose/draw after both moves
- scrolling issue
  
  use `window.scrollBy` after every move
Project 1 Features

Noted “Features” of Project 1:

- many prompts per move
- grid not shown at start
- only checking win/lose/draw after both moves
- scrolling issue

```javascript
use window.scrollBy after every move
```
Project 1 Features

Noted “Features” of Project 1:

- many prompts per move
- grid not shown at start
- only checking win/lose/draw after both moves
- scrolling issue
  - use `window.scrollBy` after every move
Object prototypes

We have seen one way to add methods to an object; assigning to an attribute in the constructor function will ensure that every instance object that is created using that function will have the method.

To do this we need to have a function, and then mention it in the constructor. There is another way, which will not require the creation of a separate function.
If there are objects being created from some constructor function then we can say that they form a class of objects – for example all objects created using the `Rectangle` constructor share some properties. They are all of the same class, “Rectangle”.

Javascript provides a specialised way of adding a method to all objects of the `Rectangle` class using a single assignment to the `prototype` of the class:

```javascript
Rectangle.prototype.area = function() {
  return this.length * this.width;
}
```
A useful trick when writing code that involves objects is being able to control how they are converted to strings when printed.

Supplying a method named `toString` allows you to specify exactly how the object should be converted to text when required (for example, when being printed, or when being added onto another string, etc). The method should return a single string.

```javascript
Rectangle.prototype.toString = function() {
    return "Rectangle of "+this.width+"x"+
            this.height+" dimensions";
};
```
After your Javascript program has exited it is still possible to have the user take action that will run some of your code. In fact, this is the technique behind most interactive web pages.

The trick is to get the browser to run your code when certain events take place. For instance, when the user clicks on a part of the page. Rather than have the program execute all in one go we will rely on the browser to run sections of our code when it is appropriate.

The general name for this kind of programming is event-driven programming.
Many HTML elements have an attribute named onclick. You can place some javascript in this attribute, and it will be run when the element is clicked by the mouse. For example, we can create a button and execute some code to display a dialog box:

```html
<script>
    function myalert() { alert("Yeah, right") }
</script>
<form>
    <input type="button" value="Click me, it’s fun" onclick="myalert();" >
</form>
```
We can combine this with a feature of the DOM to gather some input. If the form, and the appropriate elements in the form, are given name attributes then when the DOM object is constructed there will be convenient ways to access the values in them:

```
<script>
  function myalert()
    alert("Thanks "+document.EventForm.UserName.value+".")
</script>
<form name="EventForm">
  Enter your name here:
  <input type="text" size="12" name="UserName" />
  <input type="button" value="Hi" onclick="myalert()" />
</form>
```

Because it is very cumbersome to write complex code in the onclick attribute it is more usual to write functions within a regular JavaScript program, and just invoke them from within the onclick attribute.
Most HTML elements (paragraphs, spans, etc.) can be given an `onclick` attribute.

Some other events include:

- `onclick`
- `onfocus`
- `onselect`
- `onload` (this can be applied to any element which is progressively loaded, such as the HTML body, or images. The code is executed when the item has fully loaded.
- `onmouseover`. This code is executed when the mouse enters the element. It must terminate with a `return true;` to activate the code.
- `onmouseout`
Here we show how to result in the background changing to black when the mouse rolls over an element, and then reverting back to white once we leave it.

```html
<script>
    function myblackout () {
        document.bgColor = "black";
        return true
    }

    function mylightup () {
        document.bgColor = "white";
        return true
    }
</script>

<h2 onmouseover="myblackout();" onmouseout="mylightup()">blackout header</h2>
```