Client-side Programs

Until now we've written server-side programs, which return output to the client.

With client-side programs, the program is sent to the user and run on the client computer.

The client browser must understand the language that the program is written in, and we've chosen JavaScript because it's the only language that all the browsers really understand.

Hello World

greetings.js:

```
var now = new Date();
window.alert('Hello world. It is ' + now + ', right now.');
```

page.html must include a `<script>` tag in the header:

```
<script src="greetings.js"></script>
```

When the client requests the webpage, the program will be returned with the page, and then run. A popup will appear with today's date.

Note: concerning the <script> tag

You can include JavaScript code in the script tag (between the <script> and </script> tags), but it's bad practice.

JavaScript (particles.js)

- Designed for writing programs embedded within other software applications.
- Core language is very standard, should be very familiar.
- Client-side JavaScript[1] includes objects to:
  - Control the browser
  - Interact with the user
  - Communicate with the server
  - Alter the document content
- There are server-side extensions, and others.
- JavaScript and Java are both partly inspired by C but are not related.

HTML Canvas

3 Files:

- A HTML web page, particles.html
- A CSS stylesheet, particles.css
- A JS program, particles.js

We use a `<canvas></canvas>` tag, with nothing between the tags, and specify a height and width.

On the CSS

To get the canvas in the centre, we put a margin on the left and on the right, and set them both to auto. Then the browser sorts itself out.

The JavaScript (particles.js)

The initial and ending lines ensure that variables in different files with the same name won't clash.

We select the canvas with the document.querySelector method, and we create a 'context', which will be how we interact with the canvas, like using a cursor for the database.

Declaring variables

Variables in JavaScript should be explicitly declared at the beginning:

```
var hourly_pay
var hours_worked

hourly_pay = 10
hours_worked = 30
```

This can also be done with the initial assignment, as a shortcut:

```
var hourly_pay = 10
…
```

particles.js (cont.)

Making It Bounce (JavaScipt IF Statement)

Here's an example `if` statement in JS:

```
if (x < y) {
    console.log('x is smaller than y')
} else if (x === y) {
    console.log('x is equal to y')
} else {
    console.log('x is larger than y')
}
```

Notes:

- The conditions must be enclosed by ()
- 'and', 'or', and 'not' are represented by: `&&`, `||`, `!`
- We use the three = signs in the else-if check. See below.

Python is Strongly-Typed

JavaScript is weakly-typed and if you try to add e.g. a string to a number, it'll try to convert the number to a string or the string to a number and add them.

This is why above we use `===` instead of `==` in the else if, which avoids some type coercion. It's stricter.

Objects in Javascript

Simple objects look more like dictionaries:

```
var twin A = {
    firstname : 'John',
    surname : 'Grimes',
    age: 24
}
```

You can use dot notation to refer to the variables:

```
twinA.firstname => 'John'
```

In Our Example

We replace our 5 variables by an object with 5 fields in it:

```
var p = {
    x : 250,
    y : 150,
    size : 10,
    xChange : getRandomNumber(-10, 10),
    yChange : getRandomNumber(-10, 10)
}
```

The reason this is good is we can now easily add more particles, by creating more particle objects.

Arrays in JavaScript

Arrays are the JavaScript equivalent to Python's lists:

```
var groceries = ['eggs', 'milk', 'tea'];

groceries.length => 3

groceries.push('bread')

groceries is now ['eggs', 'milk', 'tea', 'bread']
```

In Our Example

We can now use a for loop (see below) to make e.g. 30 particle objects, store them all in an array, and put another for loop in draw which sorts out the motion for all of them.

for Loops in JavaScript

Here's the equivalent of `for i in range(10):` in JavaScript:

```
for (var i = 0; i < 10; i += 1) {
}
```

Since there isn't a nice equivalent of `for item in groceries` in JavaScript, you use array indexing and use a for loop that runs from `0` to the last index of the array.

Calling a Function

Here's how to define a function:

```
function irritate_me() {
    window.alert('Irritating, huh?');
}
```

To call then, you use the name of the function, as in Python.

In JavaScript, though, along with many modern programming languages, you can tell the computer to run a particular function when a particular event happens, e.g.:

```
window.addEventListener('click', irritate_me, false);
```

This code will cause the program to be executed any time the user clicks within the window. You could also try e.g. `canvas.addEventListener()`, or the same with other html elements.

This is event-driven programming.

Event-driven Programming

In JS, events include: * the webpage has finished loading * the user presses a particular key * the user moves the mouse into or out of a certain region of the screen

When an event occurs, information about the event is stored in an object. In the case of a mouse click, for example, the event has properties `clientX` and `clientY`, the co-ordinates of the mouse click.

The event handler can then access that info:

```
function irritate_me(event) {
    window.alert('Your click was at ' + event.clientX + ' ' +
event.clientY);
}
```

Note the event is passed through into the function.

We've already done this twice:

```
document.addEventListener('DOMContentLoaded', init, false);
```

This was waiting for the whole the webpage and all the stylesheets and javascript to arrive, at which point it started running the javascript.

The second example:

```
window.setInterval(draw, 33);
```

This code called draw() every 33 milliseconds.

- **Note: Look up 'Event Bubbling' and 'Event Capturing' to explain the 'false' constant in the .addEventListener() function every time so far.**

Completing an Incomplete Program

Lines surrounded by ** were added in-lecture:

```
(function() {

var canvas;
var context;
var width;
var height;

var interval_id;

var ps = [];

var player = {
    x : 0,
    y : 150,
    size : 10
};

var moveRight = false;
var moveUp = false;
var moveDown = false;

document.addEventListener('DOMContentLoaded', init, false);

function init() {
    canvas = document.querySelector('canvas');
    context = canvas.getContext('2d');
    width = canvas.width;
    height = canvas.height;

    /* The two lines below allow the program to detect when the user presses
a key and when they let go a key.*/
    **window.addEventListener('keydown', activate, false);**
    **window.addEventListener('keyup', deactivate, false);**

    /* Note the line below this has to be a variable assignment for the
stop() function to work */
```

```
    interval_id = window.setInterval(draw, 33);
}

function draw() {
    if (ps.length < 10) {
        var p = {
            x : width,
            y : getRandomNumber(0, height),
            size : 10,
            xChange : getRandomNumber(-10, -1),
            yChange : 0
        };
        ps.push(p);
    }
    context.clearRect(0, 0, width, height);
    context.fillStyle = 'yellow';
    for (var i = 0; i < ps.length; i += 1) {
        context.fillRect(ps[i].x, ps[i].y, ps[i].size, ps[i].size);
    }
    context.fillStyle ='cyan';
    context.fillRect(player.x, player.y, player.size, player.size);
    if (player.x + player.size >= width) {
        stop();
        window.alert('YOU WIN!');
        return;
    }
    for (var i = 0; i < ps.length; i += 1) {
        if (collides(ps[i])) {
            stop();
            window.alert('YOU LOSE!');
            return;
        }
    }
    for (var i = 0; i < ps.length; i += 1) {
        ps[i].x = ps[i].x + ps[i].xChange;
        ps[i].y = ps[i].y + ps[i].yChange;
        if (ps[i].x <= -ps[i].size) {
            ps[i].x = width;
        }
    }
    if (moveRight) {
        player.x += 3;
    }
    if (moveUp) {
        player.y -= 3;
    }
    if (moveDown) {
        player.y += 3;
    }
}
```

```
function getRandomNumber(min, max) {
    return Math.round(Math.random() * (max - min)) + min;
}

function collides(p) {
    if (player.x + player.size < p.x ||
        p.x + p.size < player.x ||
        player.y > p.y + p.size ||
        p.y > player.y + player.size) {
        return false;
    } else {
        return true;
    }
}

function stop() {
    clearInterval(interval_id);

    /* These two lines remove the event listeners */
    **window.removeEventListener('keydown');**
    **window.removeEventListener('keyup');**

}

/* These functions below allow the user to control the movement of the cyan
rectangle */

**function activate(event) {
    var keyCode = event.keyCode;
    if (keyCode === 38) {
        moveUp = true;
    } else if (keyCode === 39) {
        moveRight = true;
    } else if (keyCode === 40) {
        moveDown = true;
    }
}**

**function deactivate(event) {
    var keyCode = event.keyCode;
    if (keyCode === 38) {
        moveUp = false;
    } else if (keyCode === 39) {
        moveRight = false;
    } else if (keyCode === 40) {
        moveDown = false;
    }
}**

})();
```

1. These client-side extensions are less standardised, and support can differ from browser to browser.↩