Web Programming

This is CS50. Harvard University. Fall 2015.

Cheng Gong

Table of Contents

Announcements

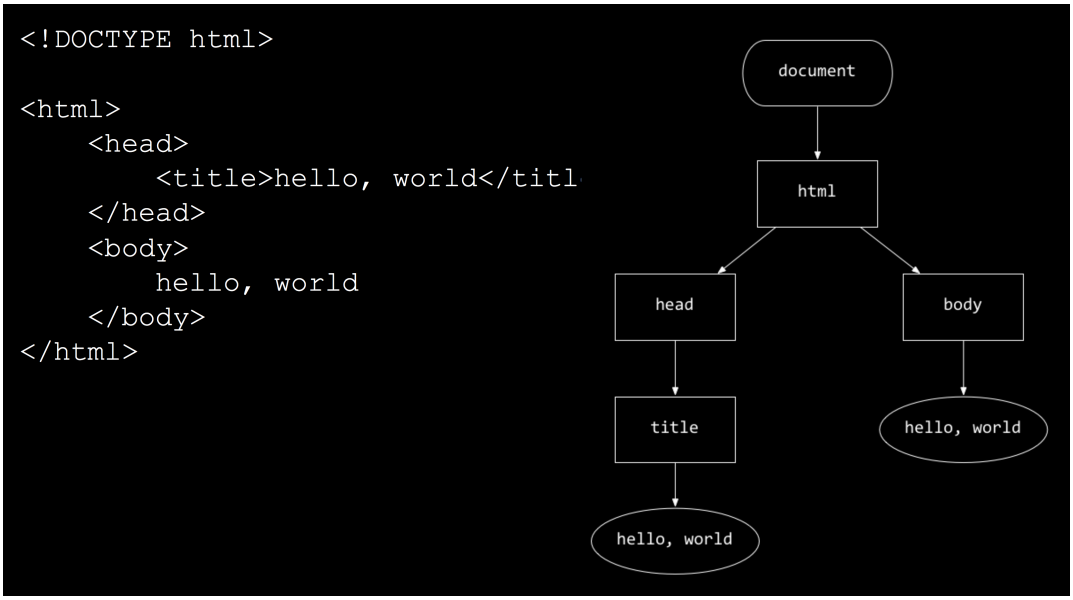
- This story[1] describes a man who wrote a script to programmatically delete files from some folders, and he made a mistake with writing the script, which resulted in all of his files being deleted!
- We'll also have seminars, the times of which are listed here[2].

Web Programming

- HTML is a language (but not programming language) since it doesn't contain much logic in how it displays a page.
- JavaScript, on the other hand, is a programming language. It's used for features we see every day, like showing new notifications or emails without having to refresh the page.
- **APIs**, application programming interfaces, are like a service that someone provides, which you can programmatically access for some functionality. **Libraries** are collections of code that we can download and use in our own programs.
- JavaScript is an **interpreted language**, which means that it is turned into 0s and 1s as the browser executes it. A compiled language, on the other hand, needs to be converted to 0s and 1s all at once, before the operating system can run it.

[1] http://www.independent.co.uk/life-style/gadgets-andtech/news/man-accidentally-deletes-his-entire-company-with-one-line-of-bad-code-a6984256.html
[2] https://cs50.harvard.edu/mba/seminars/

- So a simple webpage might look like this, with code on the left, and a tree-like structure on the right:



  # We see that each **element**, based on its indentation in the code, maps naturally to a hierarchical tree.

- JavaScript relies on manipulating this data structure in memory, through modifying, inserting, or deleting elements.
- In Scratch, we might have used the `say` block, the equivalent of which in JavaScript is something like `window.alert("hello, world");`.
- A `forever` block could be translated to JavaScript with:

```
while (true)
{
    window.alert("hello, world");
}
```

  # Normally we might have some other condition within the parentheses for the `while` loop, but using `true` which will always be true, we can make the loop continue forever.

- A `repeat 10` block might be:

```
for (var i = 0; i < 10; i++)
{
    window.alert("hello, world");
```

```
}
```

This is a little more complicated but within the `for` parentheses, we notice that we're making a new variable called `i`, setting it to `0`, checking that it's less than `10`, and adding one to it (`i++` is equivalent to `i = i + 1`, adding one to `i`).

We could also change it to `var i = 1; i # 10; i++` to achieve the same results, but conventionally we start with 0. And the first element in arrays and other data structures is generally labeled 0.

- We could translate these blocks to:

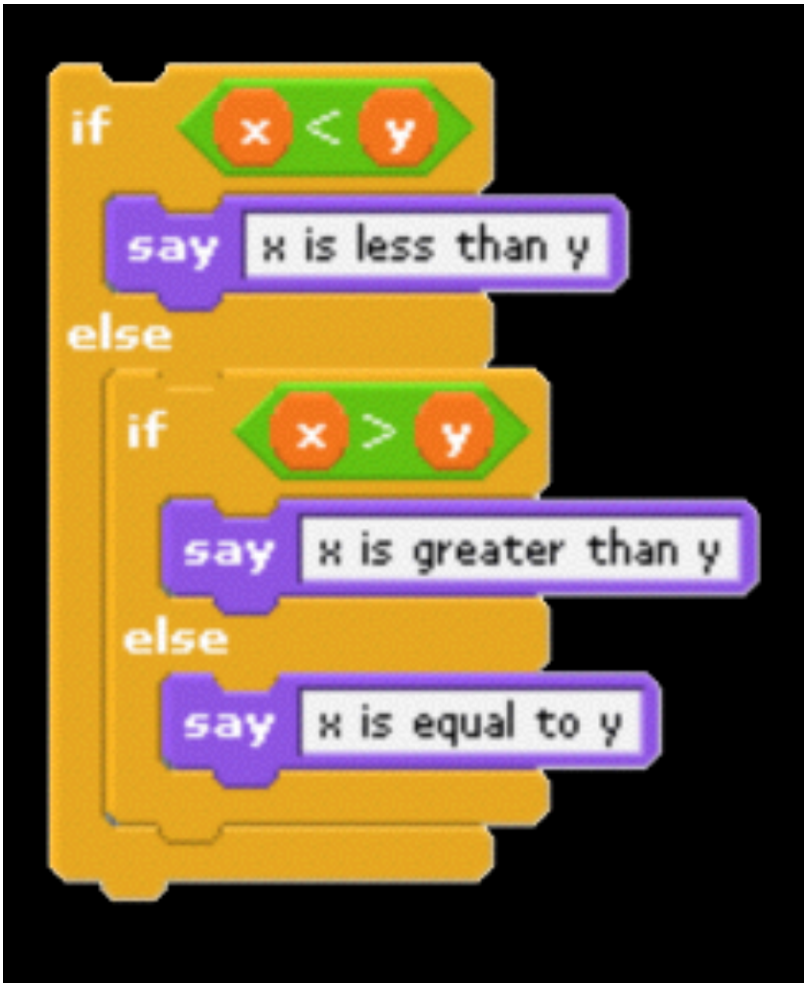

```
var counter = 0;
while (true)
{
    window.alert(counter);
    counter++;
}
```

We notice that `window.alert` can take in a variable, but there might be a bug where eventually the variable `counter` overflows and behaves incorrectly, since there is some maximum value that it can hold.

- Boolean expressions from Scratch also map fairly easily:

```
(x < y)
((x < y) && (y < z))
```

- Even though the punctuation and syntax might not be familiar, the underlying ideas of JavaScript are quite close to those we used in Scratch.
- We can have conditions, too, and many forks without having to nest indentations like we did in Scratch:




```
if (x < y)
{
    window.alert("x is less than y");
}
else if (x > y)
{
```

```
    window.alert("x is greater than y");
}
else
{
    window.alert("x is equal to y");
}
```

Without the `else if`, we would end up with something happening twice if `x < y`, since the first `if` statement would be true, then the next `if` and `else` statements would be treated as separate, and the `else` would be true. But with `else if`, only one of the three possibilities will be considered, as only the first true condition will be executed.

- We take a look at `dom-0.html`[3], where we have a form:

```
<!DOCTYPE html>

<html>
    <head>
        <script>

            function greet()
            {
                alert('hello, ' + document.getElementById('name').value +
 '!');
            }

        </script>
        <title>dom-0</title>
    </head>
    <body>
        <form id="demo" onsubmit="greet(); return false;">
            <input id="name" placeholder="Name" type="text"/>
            <input type="submit"/>
        </form>
    </body>
</html>
```

[3] http://cdn.cs50.net/2016/mba/classes/8/src8/dom-0.html.src

    # But within the code, we have `onsubmit="greet(); return false;"` instead of a URL that the browser should take us to, which seems to call the function `greet()` and then stop (which we accomplish with `return false`). The open and close parentheses mean that this function has no inputs.

    # And then inside the `head`, we have the `script` element that alerts us with whatever we typed into the `name` box inside the form. We get that by accessing a `document` object (which is built-in to JavaScript) and call the `getElementById` function on that, passing in the ID of the element we want (in this case, `name` which is what we called the input text box), and finally accessing the `value` of, or what we typed into, that element. And the `+` is just adding the strings together so we can alert the user with something like `hello, Marta!`.

- Let's take a look at `dom-2.html`[4]:

```
<!DOCTYPE html>

<html>
    <head>
        <script src="http://code.jquery.com/jquery-latest.min.js"></
script>
        <script>

            $(document).ready(function() {
                $('#demo').submit(function(event) {
                    alert('hello, ' + $('#name').val() + '!');
                    event.preventDefault();
                });
            });

        </script>
        <title>dom-2</title>
    </head>
    <body>
        <form id="demo">
            <input id="name" placeholder="Name" type="text"/>
            <input type="submit"/>
        </form>
  </body>
```

[4] http://cdn.cs50.net/2016/mba/classes/8/src8/dom-2.html.src

```
</html>
```

Here, we're using a library called jQuery which allows us to accomplish the same thing but a little more elegantly. Even though it looks more complicated here, for bigger features it simplifies code quite a bit.

- Let's look at `form-0.html`[5]:

```
<!DOCTYPE html>

<html>
    <head>
        <title>form-0</title>
    </head>
    <body>
        <form action="register.php" method="get">
            Email: <input name="email" type="text"/>
            <br/>
            Password: <input name="password" type="password"/>
            <br/>
            Password (again): <input name="confirmation" type="password"/>
            <br/>
            I agree to the terms and conditions: <input name="agreement"
 type="checkbox"/>
            <br/><br/>
            <input type="submit" value="Register"/>
        </form>
    </body>
</html>
```

This might be how we implement a sign-up form, but it allows us to put anything or nothing at all in each field.

- `form-1.html`[6], on the other hand, allows us to **validate** each field, or check that the value in each field matches our expectations.
- We could (and should) check that the fields are valid when the form is sent back to our server, but since the JavaScript code runs in the users' browsers, it runs a lot faster than sending a request to the server and waiting for a response. But we should still

[5] http://cdn.cs50.net/2016/mba/classes/8/src8/form-0.html.src
[6] http://cdn.cs50.net/2016/mba/classes/8/src8/form-1.html.src

check on our servers too because some users don't enable JavaScript. And libraries exist such that we might program these checks once, and have our code generate the validation code for both our server and in JavaScript.

- We take a look at this example[7] to see what we might be able to do. Remember that in computers, images are made up of pixels, each of which are composed of varying values of each of three colors, red, green, and blue.
- For the Iron Image, we first notice that we have something called a `SimpleImage` object. And we aren't told this on the page directly, but we can run functions on that object:

```
getRed(x, y)
getGreen(x, y)
getBlue(x, y)
```

  # Each of these functions get the value of that color at the `x` and `y` coordinates of the `SimpleImage` object.

- Likewise, we can call these functions:

```
setRed(x, y, value)
setGreen(x, y, value)
setBlue(x, y, value)
```

  # These functions set the value of that color at the `x` and `y` coordinates to `value`.

- So first, we want to set all the blue and green values to `0`:

```
var im = new SimpleImage("iron-puzzle.png");
for (x = 0; x < im.getWidth(); x++) {
  for (y = 0; y < im.getHeight(); y++) {
    im.setBlue(x, y, 0);
    im.setGreen(x, y, 0);
  }
}
print(im);
```

[7] http://nifty.stanford.edu/2011/parlante-image-puzzle/

  # The `for` loops seem to go over all the `x` values from `0` to the width of the image, and for each of those `x` values, go over all the `y` values from `0` to the height of the image, so we are going over each pixel one row at a time.
  # So for each of those pixels, we set the blue and green values to `0`.

- Now we want to multiply each red value by 10 and set it back:

```
var im = new SimpleImage("iron-puzzle.png");
for (x = 0; x < im.getWidth(); x++) {
  for (y = 0; y < im.getHeight(); y++) {
    im.setBlue(x, y, 0);
    im.setGreen(x, y, 0);
    var value = im.getRed(x, y);
    im.setRed(x, y, value * 10);
  }
}
print(im);
```

  # First, we get the red value and save it to a variable called `value`, and then we set 10 times that to the red value of the image.

- We could make this more elegant by using the value without saving it to a variable:

```
var im = new SimpleImage("iron-puzzle.png");
for (x = 0; x < im.getWidth(); x++) {
  for (y = 0; y < im.getHeight(); y++) {
    im.setBlue(x, y, 0);
    im.setGreen(x, y, 0);
    im.setRed(x, y, im.getRed(x, y) * 10);
  }
}
print(im);
```

- We solve the second puzzle similarly:

```
var im = new SimpleImage("copper-puzzle.png");
for (x = 0; x < im.getWidth(); x++) {
  for (y = 0; y < im.getHeight(); y++) {
    im.setRed(x, y, 0);
    im.setBlue(x, y, im.getBlue(x, y) * 20);
    im.setGreen(x, y, im.getGreen(x, y) * 20);
  }
```

```
}
print(im);
```

- Now we'll learn to use an API for Google Maps with this tutorial[8].
- In our C9 workspace, we'll make a new file called `map.html`, start our webserver by running `apache50 start .` in our Terminal, and visit the URL that our workspace has, with `/map.html` at the end.
- Now we copy and paste the Hello, World code from the tutorial into our `map.html`. But we need something called an **API key**, which allows us to access the service. (We signed up for one in advance for the class, `AIzaSyAmk7cP6WPlqLXgIP4mlQKg7RTDVxhKm50`, and we want to paste it into the code where it reads `YOUR_API_KEY`.)
- The default location is in Australia, so to relocate that, we'd look up our latitude and longitude and put it on line 24: `center: {lat: -34.397, lng: 150.644},`.
- And to change the map from graphical to satellite, we could read the documentation[9] and follow the instructions in setting the map type.
- So we've just scratched the surface here, but there are lots of other services provided to us by APIs, both free and paid, that we might utilize.
- Next time we'll talk about the technology stack, or what technologies we might use from back-end to front-end, and finish our time together with a discussion on mobile app programming.

[8] https://developers.google.com/maps/documentation/javascript/tutorial
[9] https://developers.google.com/maps/documentation/javascript/maptypes