JavaScript
Functions & Objects-PT1
COGS3
Review of expressions

A JavaScript statement:

```javascript
scoops = scoops - 1;
```

Variable ➔ Assignment ➔ Expression
You can write expressions that result in numbers...

**Numeric expressions**

- \((9 / 5) \times \text{tempC} + 32\)
- \(x - 1\)
- \(\text{Math.random()} \times 10\)
- \(2.123 + 3.2\)

...and you can write expressions that result in strings.

**Other expressions**

- `function () { ... }`
- `document.getElementById("pink")`
- `new Array(10)`

You can write expressions that result in the boolean values true or false (these are, obviously, boolean expressions).

**Boolean expressions**

- \(2 > 3\)
- `startTime > now`
- `tempF < 75`
- `pet = "Duck"`
- `level == 4`

...and you can write expressions that result in strings.

**String expressions**

- "super" + "cali" + youKnowTheRest
- "March" + "21" + "st"
- `P.innerHTML`
- `phoneNumber.substring(0, 3)`
What does JavaScript evaluate the following statements to?

numORString1 = "3" + "4"
numORString2 = "3" * "4"

Everything seems to work well if I add numbers to numbers or strings to strings, but what if I add a number to a string? Or an integer to a floating point number?
\[3 + 4 = \boxed{7}\]

"3" + 4 = \boxed{34}

"3" + "4" = \boxed{34}

3 \times 4 = \boxed{12}

"3" \times 4 = \boxed{12}

"3" \times "4" = \boxed{12}
if (scoops == 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 2) {
    alert("Going once!");
} else if (scoops == 1) {
    alert("Going twice!");
} else if (scoops == 0) {
    alert("Gone!");
} else {
    alert("Still lots of ice cream left, come and get it.");
}
if (scoops == 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 2) {
    alert("Going once!");
} else if (scoops == 1) {
    alert("Going twice!");
} else if (scoops == 0) {
    alert("Gone!");
} else {
    alert("Still lots of ice cream left, come and get it.");
}
<!doctype html>
<html lang="en">
<head>
<title>Icecream Scoops</title>
<meta charset="utf-8">
</head>
<script>
var scoops = 10;
while (scoops >= 0) {
  if (scoops == 3) {
    alert("Ice cream is running low!");
  } else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!";
  } else if (scoops == 2) {
    alert("Going once!");
  } else if (scoops == 1) {
    alert("Going twice!");
  } else if (scoops == 0) {
    alert("Gone!");
  } else {
    alert("Still lots of ice cream left, come and get it.");
  }
  scoops = scoops - 1;
}
alert("life without ice cream isn’t the same");
</script>
</head>
<body>
<h1>Counting ice cream scoops</h1>
</body>
</html>
var word1 = "a";
var word2 = "nam";
var word3 = "nal p";
var word4 = "lan a c";
var word5 = "a man a p";

var phrase = "";

for (var i = 0; _____; _____) {
    if (i === 0) {
        phrase = ____________;
    }
    else if (i === 1) {
        phrase = ____________ + word4;
    }
    ________ (i === 2) {
        ____________ = phrase + word1 + word3;
    }
    ________ (_____ ) {
        phrase = phrase + ________ + word2 + word1;
    }
}
alert(phrase);
Most Common

- putting the script element in the head → executes JS first

- You can always reference your JS using a link → put the URL of the file in the src attribute of the opening script tag

- You can put your code directly in the body too! → the JS is parsed at the body time. with your code in HTML

Place `<script>` elements in the `<head>` of your HTML to have them executed before the page loads

You can type your code right into your web page, or reference a separate JavaScript file using the src attribute of the script tag.

Or you can place your code (or a reference to your code) in the body. This code gets executed as the body is loaded.

Most of the time code is added to the head of the page. There are some slight performance advantages to adding your code at the end of body, but only if you really need to super-optimize your page's performance.
1. When you load a page into the browser, the browser parses the HTML and creates an internal model of your document, that contains all the elements of your HTML markup.

2. Your JavaScript can interact with the DOM to get access to the elements and the content in them. JavaScript can also use the DOM to create or remove elements.

3. When JavaScript modifies the DOM, the browser updates the page dynamically, so you see new content on your page.
1. Start by creating a document node at the top.

```
<document />
```

2. Next, take the top level element of your HTML page, in our case the `<html>` element, call it the current element and add it as a child of the document.

```
<document>
  <html />
</document>
```

3. For each element nested in the current element, add that element as a child of the current element in the DOM.

```
<document>
  <html>
    <head>
      <title>My blog</title>
      <meta charset="utf-8" />
      <script src="blog.js" /></script>
    </head>
    <body>
      <h1>My blog</h1>
      <div id="entry1">
        <h2>Great day bird watching</h2>
        <p>Today I saw three ducks!
           I named them
           Huey, Louie, and Dewey.</p>
        <p>I took a couple of photos...</p>
      </div>
    </body>
  </html>
</document>
```
The DOM includes the content of the page as well as the elements. (We don’t always show all the text content when we draw the DOM, but it’s there.)
DRAW THE DOM FOR THIS

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Movies</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1">Plan 9 from Outer Space</h2>
    Playing at 3:00pm, 7:00pm.
    <span>
      Special showing tonight at <em>midnight</em>!
    </span>
    <h2 id="movie2">Forbidden Planet</h2>
    Playing at 5:00pm, 9:00pm.
  </body>
</html>
```
JS & HTML

→ communicate via DOM

JS can get access to DOM via: `getElementById`

`getElementById`

`camelCase`
document.getElementById("greenplanet");

getElementsBy("greenplanet") returns the paragraph element corresponding to "greenplanet"...

Here we’re asking the document to get us an element by finding the element that matches the given id.

Remember the document represents the entire page in your browser and contains the complete DOM, so we can ask it to do things like find an element with a specific id.

doncument.getElementById("greenplanet");

Here we’re asking the document to get us an element by finding the element that matches the given id.

document.getElementById("greenplanet");

Here we’re asking the document to get us an element by finding the element that matches the given id.

window.onload = init;

function init() {
  var planet = document.getElementById("greenplanet");
  planet.innerHTML = "Red Alert: hit by phaser fire!";
}

<h1>Green Planet</h1>
<p id="greenplanet">All is well</p>

<h1>Red Planet</h1>
<p id="redplanet">Nothing to report</p>

<h1>Blue Planet</h1>
<p id="blueplanet">All systems A-OK</p>
Here's a DOM with a secret message hidden in it. Evaluate the code below to reveal the secret!

document.getElementById("e7")
document.getElementById("e8")
document.getElementById("e16")
document.getElementById("e9")
document.getElementById("e18")
document.getElementById("e13")
document.getElementById("e12")
document.getElementById("e2")

```

```

What is the secret message???

Write the element each line of code selects, as well as the content of the element to reveal the secret message!
Green Planet
All is well

Red Planet
Nothing to report

Blue Planet
All systems A-OK
getElementById gets access to an element – use **innerHTML** to change the content of that element.
We're assigning the element to a variable named `planet`.

```javascript
var planet = document.getElementById("greenplanet");
```

Here's our call to `getElementById`, which seeks out the "greenplanet" element and returns it.

And in our code we can now just use the variable `planet` to refer to our element.

```javascript
planet.innerHTML = "Red Alert: hit by phaser fire!";
```

We can use the `innerHTML` property of our `planet` element to change the content of the element.

We change the content of the `greenplanet` element to our new text... which results in the DOM (and your page) being updated with the new text.
```javascript
var planet = document.getElementById("greenplanet");

planet.innerHTML = "Red Alert: hit by phaser fire!";
```

We change the content of the greenplanet element to our new text... which results in the DOM (and your page) being updated with the new text.

Any changes to the DOM are reflected in the browser’s rendering of the page, so you’ll see the paragraph change to contain the new content!
• Your DOM is created when the page is fully loaded.
• If the JS is executed before the DOM is created, then it cannot change anything.
• Need to tell the browser: “run my code after you’ve fully loaded in the page and created the DOM”
First, create a function named `init` and put your existing code in the function.

Notice that your code goes between an opening `{` and a closing `}`.

Here, we’re setting the value of the `window.onload` property to the function name.

This says when the page is fully loaded, execute the code that is in `init`. 

```html
<doctype html>
<html lang="en">
    <head>
        <title>Planets</title>
        <meta charset="utf-8">
    </head>
    <script>
        function init() {
            var planet = document.getElementById("greenplanet");
            planet.innerHTML = "Red Alert: hit by phaser fire!");
        }
        window.onload = init;
    </script>
    <body>
        <h1>Green Planet</h1>
        <p id="greenplanet">All is well</p>
        <h1>Red Planet</h1>
        <p id="redplanet">Nothing to report</p>
        <h1>Blue Planet</h1>
        <p id="blueplanet">All systems A-OK</p>
    </body>
</html>
```