Introduction to JavaScript

Instructor:	Sukhbir Tatla
Email:	sukhbir.tatla@sheridancollege.ca
Course:	Web Programming (SYST10199)

Typecasting

- ***Typecasting is the act of converting one data type to another.***
  - Implicit Typecasting
  - Explicit Typecasting

Implicit Typecasting

- ***Typecasting is the act of converting one data type to another.***
- Implicit typecasting is when the compiler or interpreter does the conversion automatically for you.
- In Boolean expressions, (i.e. the expressions that evaluate to `true` or `false` to control `if` statements and loops), JavaScript uses implicit typecasting to compare.

🛠️ Do It Yourself!

- Try the following in a JavaScript console:

```
var x = 50;
var y = "50";
y == x;
```

- This comparison is `true`, even though `y` is a `string` and `x` is a `number`.

Implicit Typecasting

- Because of this *type-blindness*, there is a special Boolean operator `===` that returns `true` **if two values are equal and also of the same type.**
- This is known as being ***exactly equal*** or ***identical***.
- Similarly, `!==` is a ***not exactly equal to***.

Do It Yourself!

- Try these statements (with the same `x` and `y` variables from the last DIY example.

```
var x = 50;
var y = "50";
```

- Try to predict the result before you hit enter.

```
y === x;
y === "50";
y !== x;
y !== "50";
```

- Explain to yourself what is happening in each case, and why each statement is either `true` or `false`.

Explicit Typecasting

- Sometimes, you really want to treat a string as a number.
- For example, the prompt function always returns a string, which works for most things, but suppose we are asking the user to enter a number and then we want to add 1 to it.

🛠 Do It Yourself!

- Try this in a `<script>` element or a console window.
- What is going wrong?

```
var y = prompt("Enter a number");
y = y + 1;
alert(y);
```

Explicit Typecasting

- The previous example doesn't do what we expect because `y` is a string.
- So, JavaScript treats + as concatenation (much like Java).
- In this case, we need to use explicit typecasting to force the type we want.
- JavaScript has built in `parseInt` and `parseFloat` functions.
- Note that `I` and `F` are uppercase in `parseInt` and `parseFloat`.

🛠️ Do It Yourself!

- Here's the fix for the last example. Try it to verify it works.

```
var y = prompt("Enter a number");
y = parseInt(y) + 1;
alert(y);
```

Not a Number

- But what if the user typed something that can't be interpreted as a number?
- In that case, the `prompt` will return the special value *NaN (Not a Number)* which can be detected with the Boolean function `isNaN()`.

- The `isNaN()` function determines whether a value is an illegal number (Not-a-Number).
- This function returns `true` if the value is NaN, and `false` if not.

🛠 Do It Yourself!

- Try this out in a `<script>` element or in a JavaScript console.

```
var num = prompt("Enter a number");
if (isNaN(num))
    alert("You entered an invalid number");
else
    alert("You entered a valid number");
```

🛠 Do It Yourself!

- Try this out in a `<script>` element or in a JavaScript console. Can you explain what each line is doing?

```
var n;
do
{
    n = prompt("Enter a number");
} while(isNaN(n));

n = parseInt(n) + 1;
alert(n);
```

Arrays

Arrays in JavaScript

- Like most languages, JavaScript contains support for arrays.
- In many cases the code for processing arrays will look very familiar.
- Things are only slightly different in JavaScript because of the consequences of the language being dynamically-typed.

🛠 Do It Yourself!

- Use your Java knowledge to understand and describe the effect of this JavaScript code:

```
var a = [3, 4, 5];
for(var i = 0; i < a.length; i++)
    a[i] = a[i] * 2;
```

- Take a minute to think about what this code does, then type it into a browser console window to run it, and then type **a** and press enter to examine the contents of the array. Did you get it right?

Arrays in JavaScript

- The following pieces of code show two different ways of creating a generic empty array.

```
var a = [];
var a = new Array();
```

- You can also create an array with some initial contents, like this:

```
var initializedArray = [43, "hello", -2.5, true];
```

- Notice that this array contains values of three different types (Number, String, and Boolean).

🛠 Do It Yourself!

- You never have to declare the size of an array in advance because you can arbitrarily extend its length after you create it.
- Type the following into a browser console.

```
var a = [];
a.length;
```

- Press enter to see the output.
- Now, try the following statements:

```
a[0] = -23;
a[1] = 45;
a.length;
```

- This is an example of extending the length of an initially empty array.

🛠️ Do It Yourself!

- It is also possible to create an array with **holes** in it.
- Complete the previous DIY, and then in the same console window, type the following:

```
a[9] = 0;
```

- You just created an array with 7 holes in it from indices 2 to 8.
- How long is the array now?
- Type `a.length` to find out, then type `a` and hit enter to find out what the contents of the array looks like.
- The holes in the array have been filled with the special global property `undefined`.
- So if you are ever not sure whether an array index has been given a value, you can test it in an `if` statement.
- Try this:

```
if (a[3] === undefined)
      alert("index 3 is undefined");
```

Array Differences in Java and JavaScript

- JavaScript arrays are a lot like Java arrays, except:
    1. You don’t need to specify the type or size of the array.
    2. You can mix data types in the same array.
    3. You can increase the size of the array after it has been created.
    4. The array indices you fill with values don’t have to be contiguous.

🛠 Do It Yourself!

1. Create a page that prompts the user for 10 numbers and then computes the average value of the numbers entered and reports in an alert box how many of the values they entered were above average.
2. Create a page with a script that defines an array and puts the String values "Hello", "World", and "!" into random array locations between 1 and 10. Then prompt the user for three integers and check to see if they have found the array locations where the three String values are stored. Give them a score out of 3 depending on how many they found and show them the values they uncovered using alert boxes.