

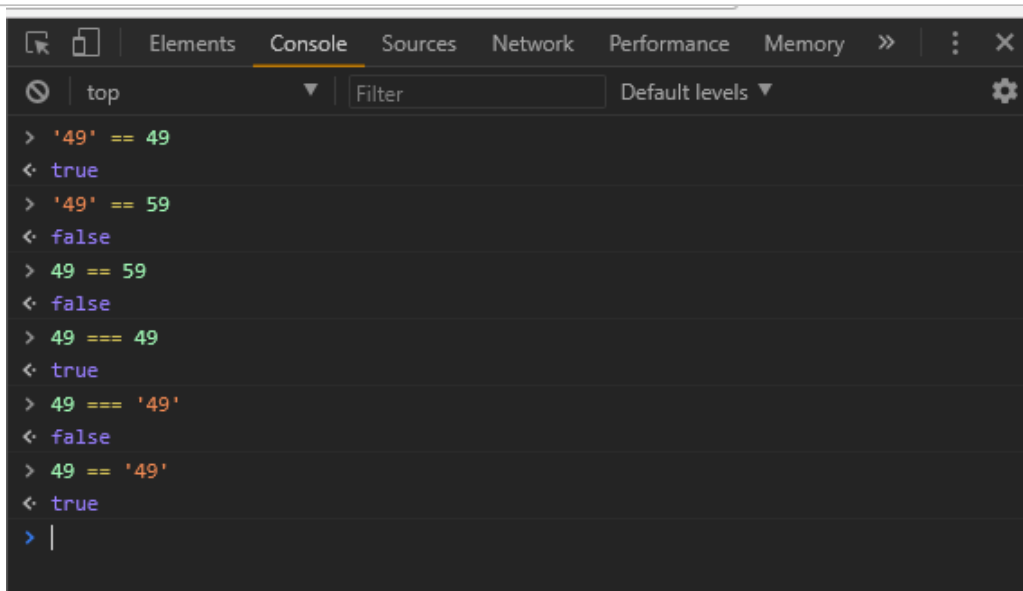
Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 2 - Wednesday

Dr Nick Hayward

JS Basics - examples of coercion - part I

Coerce strings and numbers...

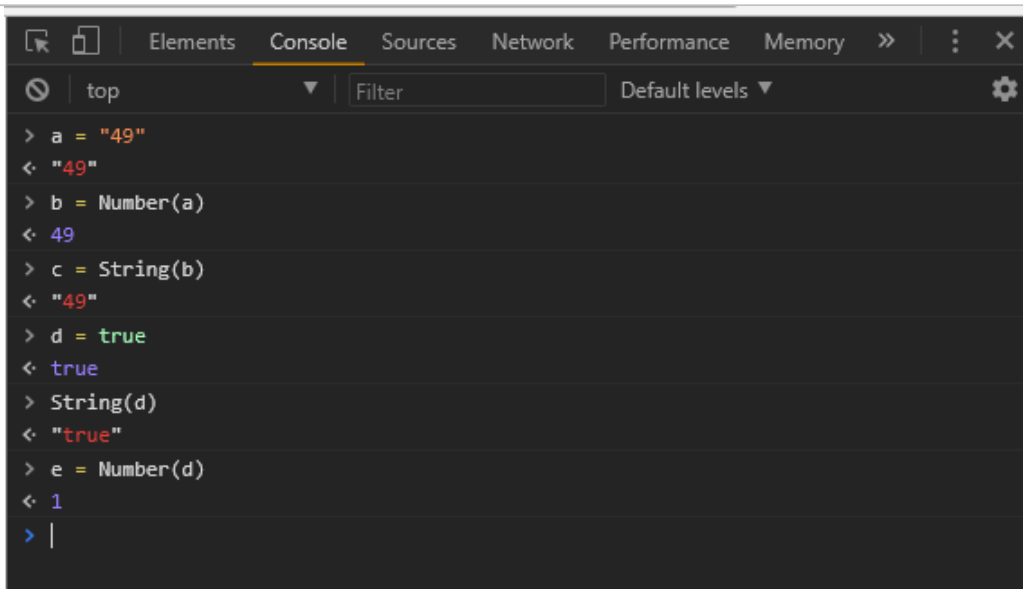
A screenshot of a web browser's developer console, specifically the 'Console' tab. The console shows a series of JavaScript commands and their results, demonstrating type coercion. The commands and results are: '> '49' == 49' returns '< true'; '> '49' == 59' returns '< false'; '> 49 == 59' returns '< false'; '> 49 === 49' returns '< true'; '> 49 === '49'' returns '< false'; and '> 49 == '49'' returns '< true'. The console interface includes a toolbar at the top with icons for opening the console, showing the Elements panel, and other developer tools. Below the toolbar, there's a 'top' dropdown, a 'Filter' input, and a 'Default levels' dropdown. The console output is displayed in a dark-themed area with syntax highlighting: strings are in orange, numbers in green, and booleans in blue.

```
> '49' == 49
< true
> '49' == 59
< false
> 49 == 59
< false
> 49 === 49
< true
> 49 === '49'
< false
> 49 == '49'
< true
> |
```

JavaScript - examples of type coercion

JS Basics - examples of coercion - part 2

Coerce strings, numbers, booleans...

A screenshot of a web browser's developer console, specifically the 'Console' tab. The console shows a series of JavaScript commands and their outputs, demonstrating type coercion. The commands and outputs are: 1. Command: > a = "49"; Output: < "49" (string). 2. Command: > b = Number(a); Output: < 49 (number). 3. Command: > c = String(b); Output: < "49" (string). 4. Command: > d = true; Output: < true (boolean). 5. Command: > String(d); Output: < "true" (string). 6. Command: > e = Number(d); Output: < 1 (number). The console interface includes a search bar at the top with 'top' and 'Filter' text, and a 'Default levels' dropdown menu. The background is dark, and the text is light-colored for readability.

```
> a = "49"
< "49"
> b = Number(a)
< 49
> c = String(b)
< "49"
> d = true
< true
> String(d)
< "true"
> e = Number(d)
< 1
> |
```

JavaScript - examples of type coercion

JS Basics - variables - part I

- **symbolic** container for values and data
- applications use containers to keep track and update values
- use a **variable** as a container for such values and data
 - *allow values to vary over time*
- JS can emphasize types for values, does not enforce on the variable
 - **weak typing** or **dynamic typing**
 - *JS permits a variable to hold a value of any type*
- often a benefit of the language
- a quick way to maintain flexibility in design and development

JS Basics - variables - part 2

- declare a variable using the keyword `var`
- declaration does not include **type** information

```
var a = 49;  
//double var a value  
var a = a * 2;  
//coerce var a to string  
var a = String(a);  
//output string value to console  
console.log(a);
```

- `var` a maintains a running total of the value of a
- keeps record of changes, effectively **state** of the value
- **state** is keeping track of changes to any values in the application

JS Basics - variables - part 3

- use variables in JS to enable central, common references to our values and data
- better known in most languages simply as **constants**
- JS is similar
 - *creates a read-only reference to a value*
 - *value itself is not immutable, e.g. an object...*
 - *it's simply the identifier that cannot be reassigned*
 - *JS constants are also bound by scoping rules*
- allow us to define and declare a variable with a value
 - *not intended to change throughout the application*
- **constants** are often declared together
 - *uppercase is standard practice - although not a rule...*
- form a store for values abstracted for use throughout an app
- JS normally defines constants using uppercase letters,

```
var NAME = "Philae";
```

- ECMAScript 6, ES6, introduces additional variable keywords
 - e.g. *const*

```
const TEMPLE_NAME = "Philae";
```

- benefits of abstraction, ensuring value is not accidentally changed
 - *change rejected for a running app*
 - *in strict mode, app will fail with an error for any change*

JS Basics - comments

- JS permits comments in the code
- two different implementations

single line

```
//single line comment  
var a = 49;
```

multi-line

```
/* this comment has more to say...  
we'll need a second line */  
var b = "forty nine";
```

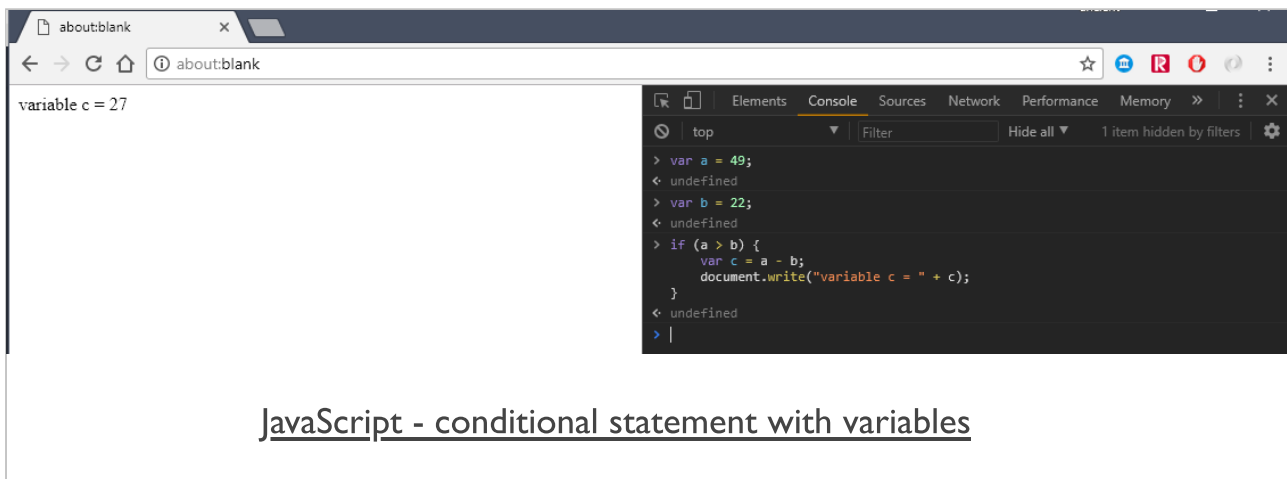
JS Basics - logic - blocks & conditionals - part I

- simple act of grouping contiguous and related code statements together
 - *known as **blocks***
- block defined by wrapping statements together
 - *within a pair of curly braces, { }*
- **blocks** commonly attached to other forms of control statement

```
if (a > b) {  
  ...do something useful...  
}
```

- conditional statements require a decision to be made
- JS includes many different ways we can express **conditionals**
- most common example is the `if` statement
 - *if this given condition is true, do the following...*
 - *if statement requires an expression between the parentheses*
 - *evaluates as either true or false*

JS Basics - logic - conditional statement



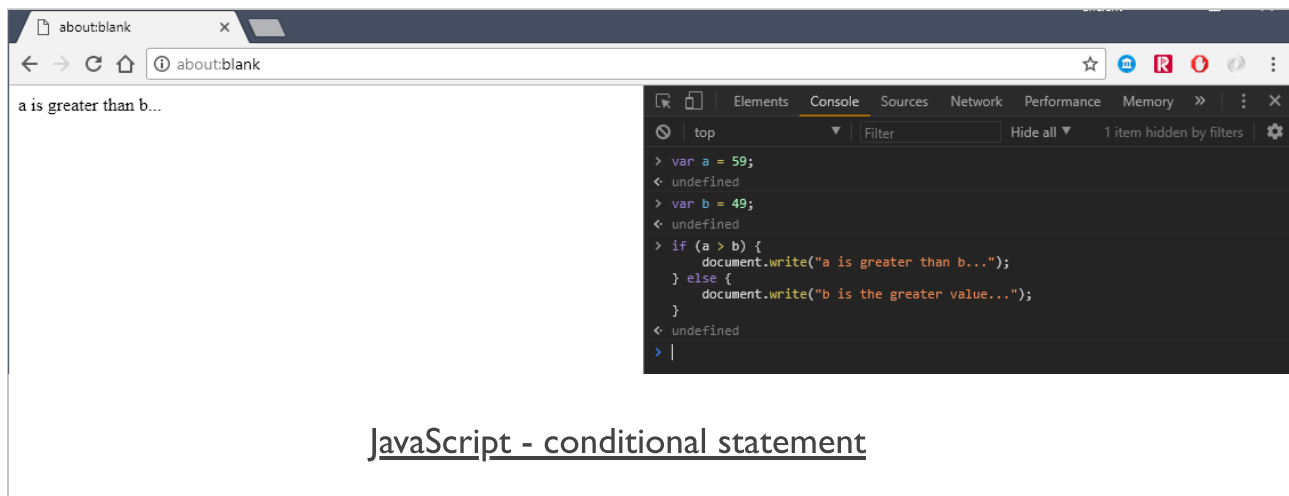
JS Basics - logic - blocks & conditionals - part 2

- additional option if this expression returns false
 - using an **else** clause

```
if (a > b) {  
  console.log("a is greater than b...");  
} else {  
  console.log("no, b is greater...");  
}
```

- for an `if` statement, JS expects a `boolean`
- JS defines a list of values that it considers *false*
 - e.g. `0`...
- any value **not** on this list of *false* values will be considered true
 - coerced to true when defined as a *boolean*
- conditionals in JS also exist in another form
 - the *switch* statement
 - more to come...

JS Basics - logic - conditional statement



JS Basics - logic - loops

- loops allow repetition of sets of actions until a condition fails
- repetition continues whilst the requested condition holds
- loops take many different forms and follow this basic behaviour
- a loop includes the *test condition* as well as a *block*
 - *normally within curly braces*
 - *block executes, an iteration of the loop has occurred*
 - *four kinds of loop by default in JS,*
 - `for`
 - `for/in`
 - `while`
 - `do/while`

JS Basics - logic - loops - for

- for loop has three clauses, including
 - *initialisation clause*
 - *conditional test clause*
 - *update clause*

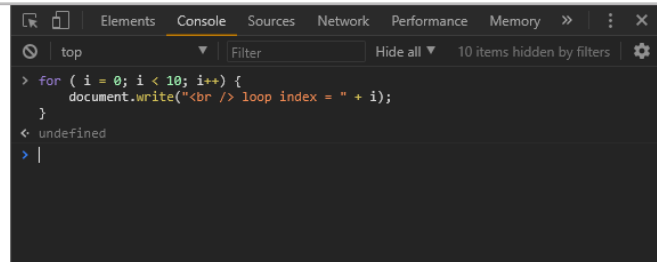
```
for (statement1; statement2; statement3) {  
    ...code block...  
}
```

- statement1 = executes before loop starts
 - *statement2 = condition for running the loop*
 - *statement3 = executes after each iteration of the loop*

JS Basics - logic - loops - for

Loop through a defined index from 0...

```
loop index = 0  
loop index = 1  
loop index = 2  
loop index = 3  
loop index = 4  
loop index = 5  
loop index = 6  
loop index = 7  
loop index = 8  
loop index = 9
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the output of a JavaScript for loop: 'loop index = 0' through 'loop index = 9'. The code in the console is:

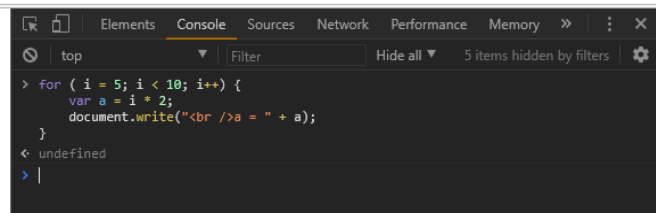
```
> for ( i = 0; i < 10; i++) {  
    document.write("<br /> loop index = " + i);  
}  
< undefined  
> |
```

JavaScript - for loop

JS Basics - logic - loops - for

Create a custom index and multiply per loop iteration...

```
a = 10  
a = 12  
a = 14  
a = 16  
a = 18
```



```
top  
Filter  
Hide all 5 items hidden by filters  
> for ( i = 5; i < 10; i++) {  
  var a = i * 2;  
  document.write("<br />a = " + a);  
}  
← undefined  
> |
```

JavaScript - for loop with multiplication

JS Basics - logic - loops - while & do/while

- `while` and `do...while` loops
- basic difference between these loops, `while` and `do...while`
 - *conditional tested is before the first iteration (`while` loop)*
 - *after the first iteration (`do...while`) loop*
- if the condition is initially false
 - *a `while` loop will never run*
 - *a `do...while` will run through for the first time*
 - *other specialised forms of loop in JavaScript*
 - e.g. `for/in...`

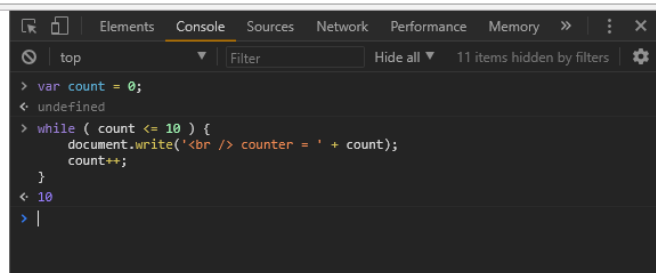
n.b. programming languages, and CS in general, start counting at 0. i.e. an index of values...

JS Basics - logic - loops - while

while loop continues to execute whilst condition remains true...

```
while (condition is true) {  
    ...code block...  
}
```

```
counter = 0  
counter = 1  
counter = 2  
counter = 3  
counter = 4  
counter = 5  
counter = 6  
counter = 7  
counter = 8  
counter = 9  
counter = 10
```

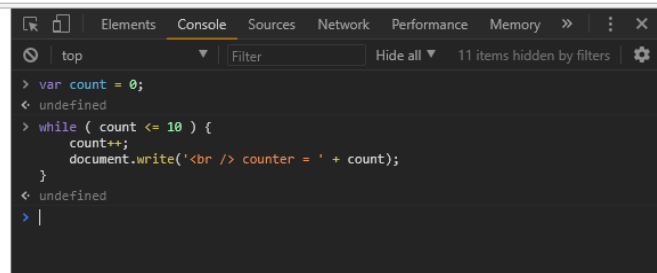
A screenshot of a web browser's developer console. The console shows the execution of a JavaScript while loop. The code entered is: `> var count = 0;`, `< undefined`, `> while (count <= 10) {`, `document.write('
 counter = ' + count);`, `count++;`, `}`, `< 10`, and `> |`. The console also shows a message 'Hide all 11 items hidden by filters' and a gear icon for settings.

JavaScript - while loop

JS Basics - logic - loops - while

while loop with counter increment before output...

```
counter = 1  
counter = 2  
counter = 3  
counter = 4  
counter = 5  
counter = 6  
counter = 7  
counter = 8  
counter = 9  
counter = 10  
counter = 11
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The code being executed is as follows:

```
> var count = 0;  
< undefined  
> while ( count <= 10 ) {  
    count++;  
    document.write('<br /> counter = ' + count);  
}  
< undefined  
> |
```

The console output shows the counter incrementing from 1 to 11, with each value on a new line. The text '11 items hidden by filters' is visible at the top right of the console panel.

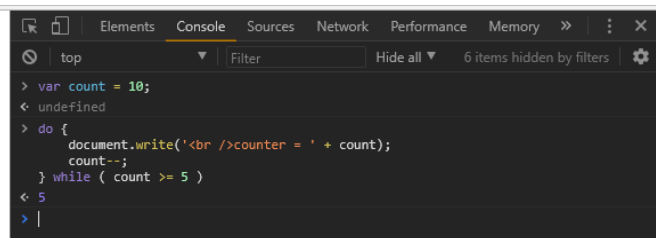
JavaScript - while loop

JS Basics - logic - loops - do/while

do/while loop executes do first, and then checks while condition...

```
do {  
    ...code block...  
} while (condition is true)
```

```
counter = 10  
counter = 9  
counter = 8  
counter = 7  
counter = 6  
counter = 5
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The code being executed is as follows:

```
> var count = 10;  
< undefined  
> do {  
    document.write('<br />counter = ' + count);  
    count--;  
} while ( count >= 5 )  
< 5  
> |
```

The output in the console shows the value of 'count' decreasing from 10 to 5, with each iteration of the loop writing a line to the document. The console also indicates that 6 items are hidden by filters.

JavaScript - do/while loop