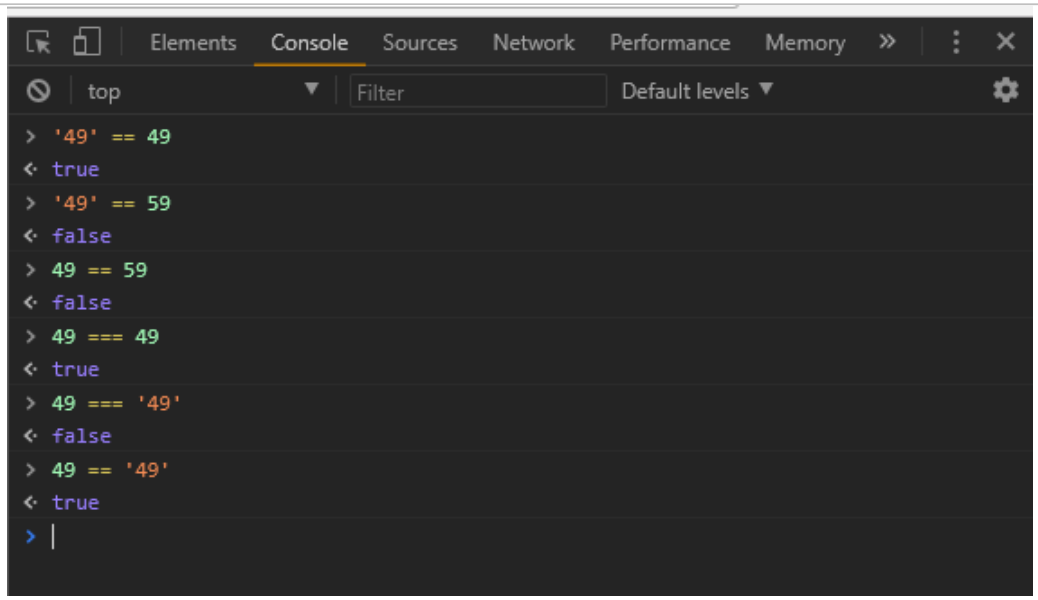
Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 2 - Wednesday

Dr Nick Hayward

JS Basics - examples of coercion - part I

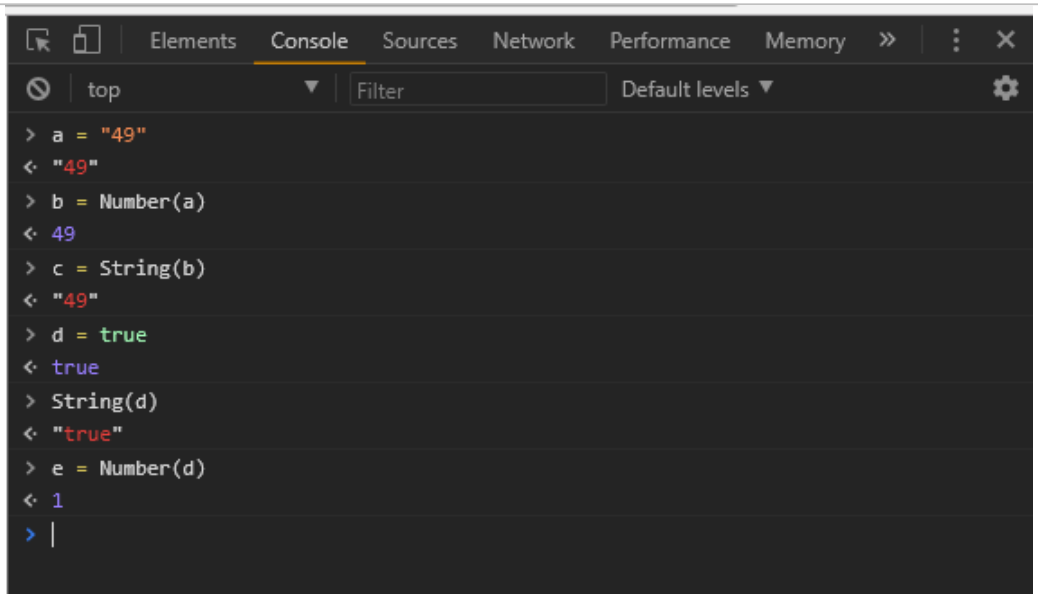
Coerce strings and numbers...



JavaScript - examples of type coercion

JS Basics - examples of coercion - part 2

Coerce strings, numbers, booleans...



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";
```

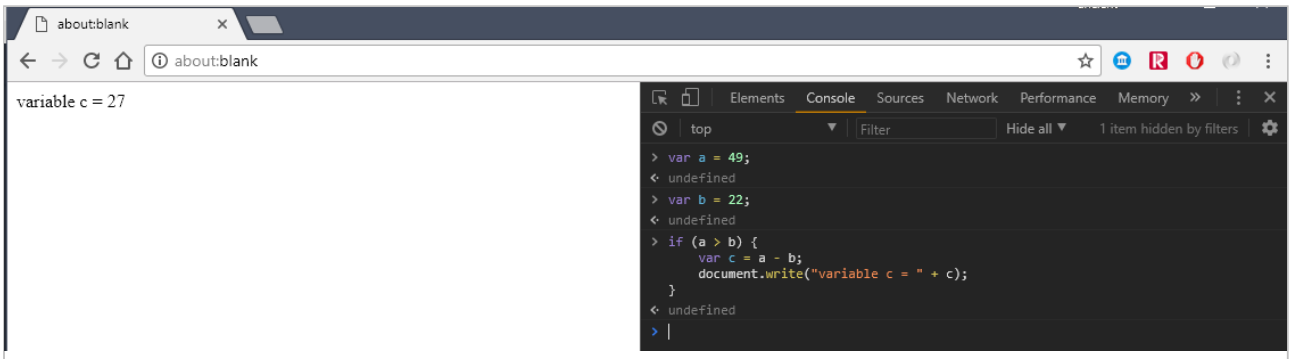
JS Basics - logic - blocks & conditionals - part I

- 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



JavaScript - conditional statement with variables

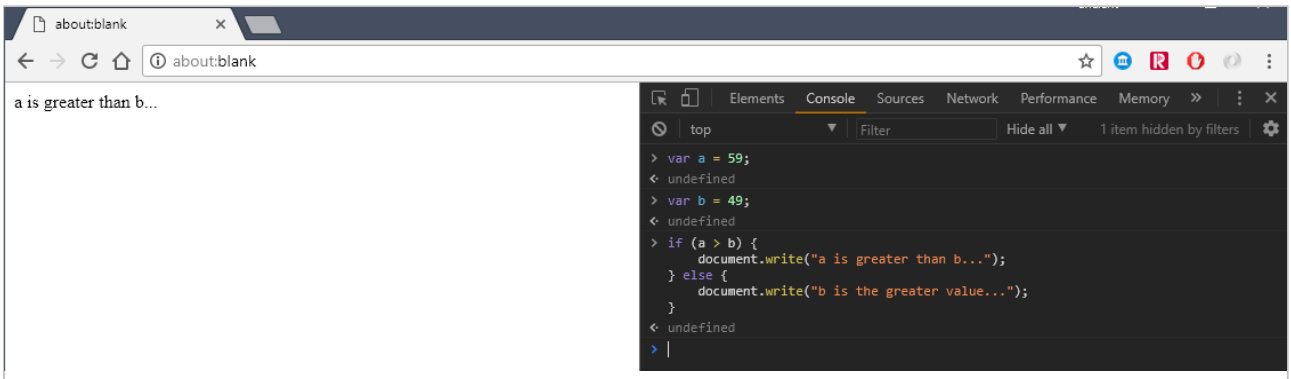
JS Basics - logic - blocks & conditionals - part 2

- 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



JavaScript - 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`

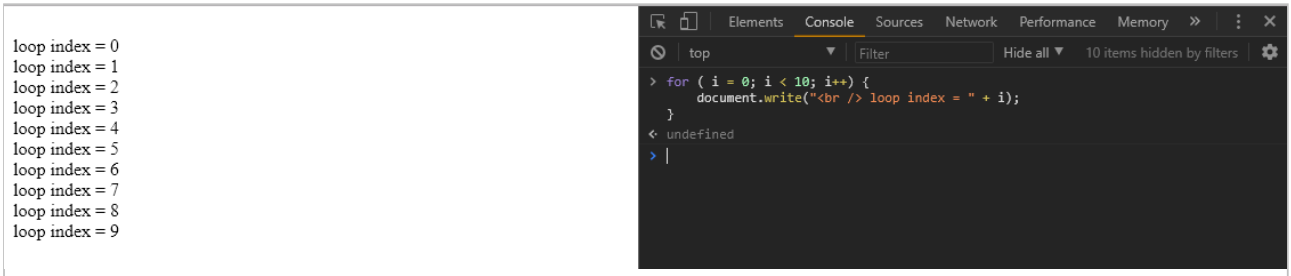
- `for` loop has three clauses, including
  - *initialisation clause*
  - *conditional test clause*
  - *update clause*

```
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`

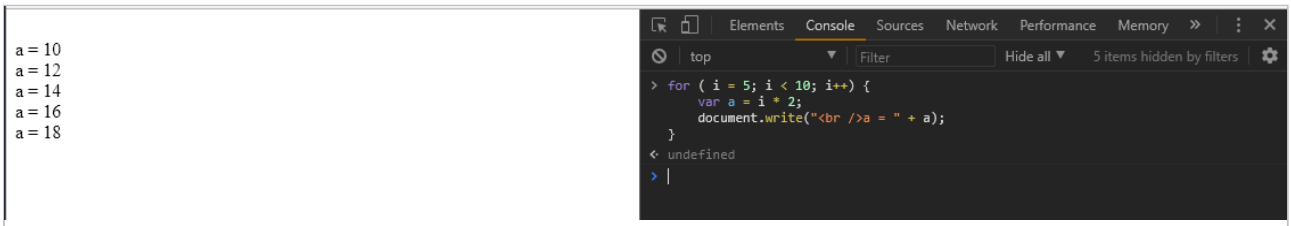
Loop through a defined index from 0...



JavaScript - `for` loop

JS Basics - logic - loops - `for`

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



JavaScript - `for` loop with multiplication

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

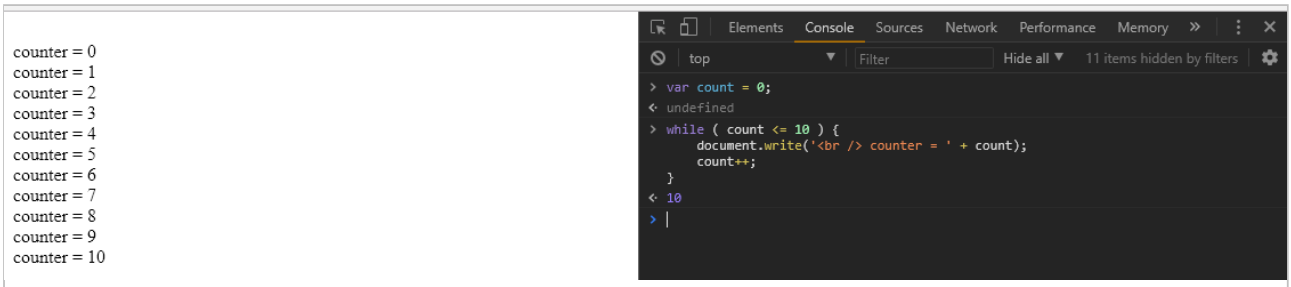
- `while` and `do...while` loops
- basic difference between these loops, `while` and `do...while`
  - *conditional tested is before the first iteration (`while` loop)*
  - *after the first iteration (`do...while`) loop*
- if the condition is initially false
  - *a `while` loop will never run*
  - *a `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...

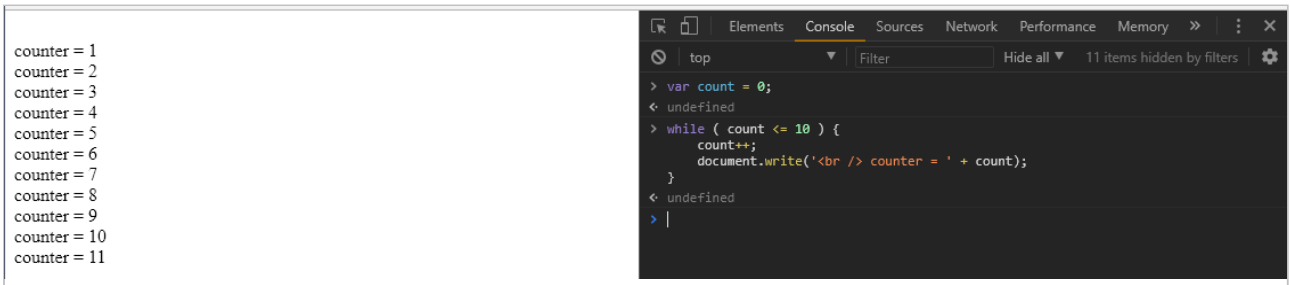
```
while (condition is true) {
    ...code block...
}
```



JavaScript - `while` loop

JS Basics - logic - loops - `while`

`while` loop with counter increment before output...

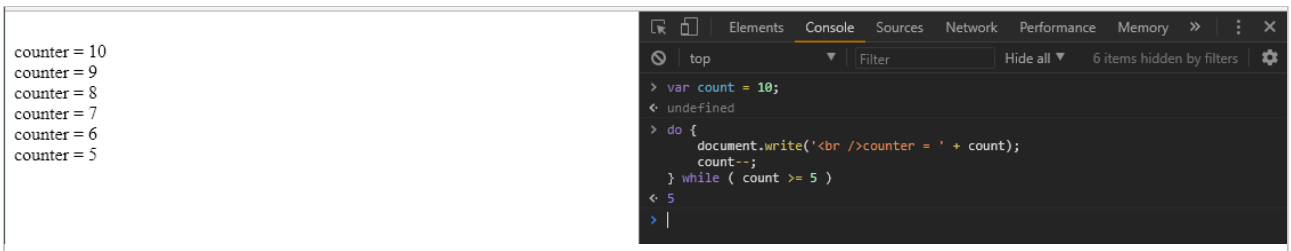


JavaScript - `while` loop

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

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

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



JavaScript - `do/while` loop