



# Intro to JavaScript

WD-405

# Prerequisites

- HTML
- CSS
- Browser capabilities
- Programming background helps, not necessary

# JavaScript related courses

- **Programming**

- WD-405 JavaScript (for non-programmers)

- **Admin**

- WD-520 JavaScript powered web apps (Node.js, AngularJS...)

- **Web**

- WD-500 HTML5 and CSS3 for Web Designers
- WD-505 Intro to jQuery
- WD-510 Mobile Web App Design
- WD-515 Mobile Web App Development

# JavaScript related courses



- **Web**
  - WD-530 Angular with TypeScript



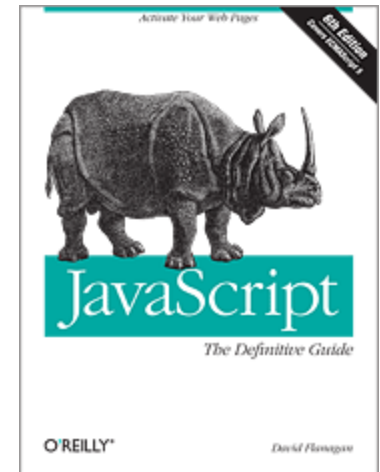
# JavaScript related courses

- Microsoft Official Courseware
  - MS-20480 Microsoft Programming in HTML5 with JavaScript and CSS3
  - MS-20481A Essentials of Developing Windows Store Apps Using HTML5 and JavaScript
  - MS- 20483 - C#

# Book for course



- **JavaScript: The Definitive Guide**, 6th Edition, David Flanagan, O'Reilly Media, Inc. May 3, 2011



# Basic books

- **Jump Start JavaScript**, Ara Pehlivanian; Don Nguyen, SitePoint, July 12, 2013
- **JavaScript Step by Step**, 3rd Edition Steve Suehring, Microsoft Press, June 6, 2013
- **JavaScript: The Good Parts**, Douglas Crockford, O'Reilly Media, Inc. May 8, 2008



# Repo



- <https://github.com/doughoff/core>
- <https://github.com/doughoff/WD-405>





THEN WE PROGRAM  
THE WEB SITE USING A  
FAST GUY IN TIGHTS  
AND A MOVIE ABOUT  
COFFEE.



www.dilbert.com scottadams@aol.com

CORRECT  
ME IF I'M  
WRONG.



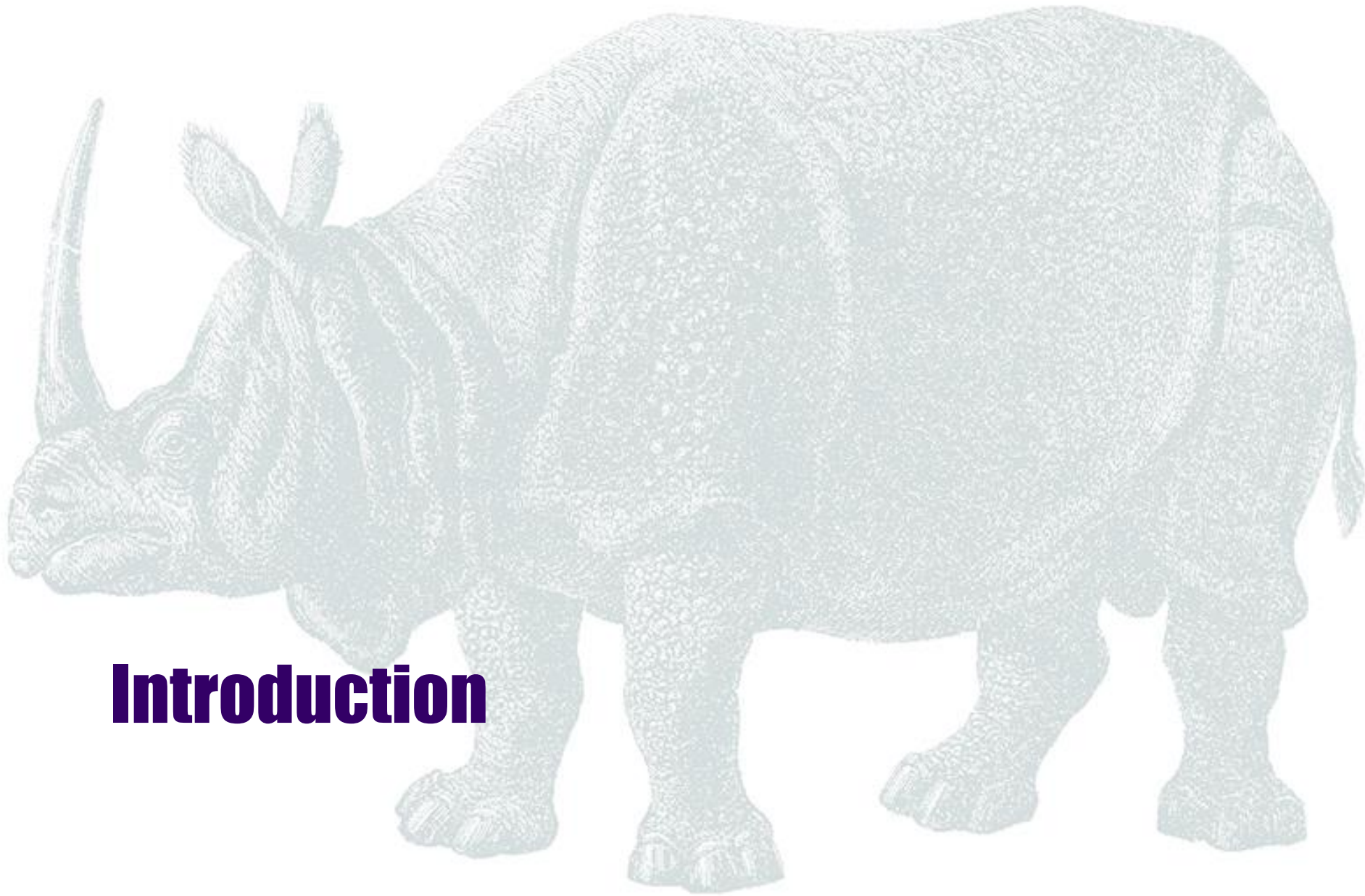
WE USE  
FLASH  
AND  
JAVA-  
SCRIPT



© 2001 Scott Adams, Inc./Dist. by UFS, Inc.

I SAID,  
"IF"!!!





# Introduction



# pre-ECMAScript 1995

- <http://www.ecmascript.org/>
- Mocha - created in 10 days in May 1995 by Brendan Eich
- Netscape's JavaScript (LiveScript) - Sep 1995 Navigator 2.0 beta
  - JavaScript - Dec 1995 Navigator 2.0b3
- Netscape and Sun Micro (Java) joint venture
  - renamed JavaScript
  - Microsoft shipped in 1996 as JScript
  - Netscape wanted standardization from ECMA Int.

# ECMAScript 1996 - 2011

- ECMAScript 1 - 1996-7,
- ECMAScript 2 - 1998
- ECMAScript 3 - 1999
  - regex, string, try/catch, function expressions
- ECMAScript 4 - 2000-08 abandoned
- ECMAScript 5, 5.1 - 2009/2011
  - Objects, function and array extensions, strict mode, foreach, JSON

# ECMAScript 6 (ES6)



- June 2015 complete
  - Object APIs, **classes**, modules, concise methods, arrow functions, rest and spread, property initializers, proxy/reflect, collections
  - Code-named "Harmony"
  - Not supported in IE11 or iOS Safari 8

# Popularity

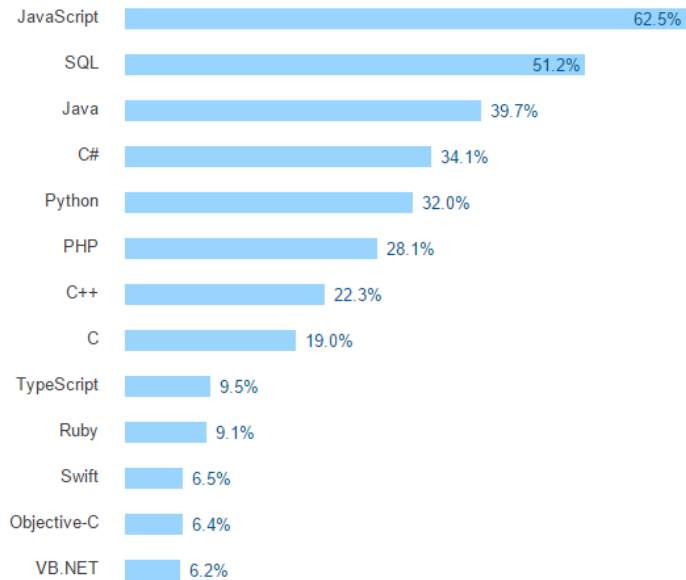


## Most Popular Technologies

### Programming Languages

% of This Category

% of All Respondents



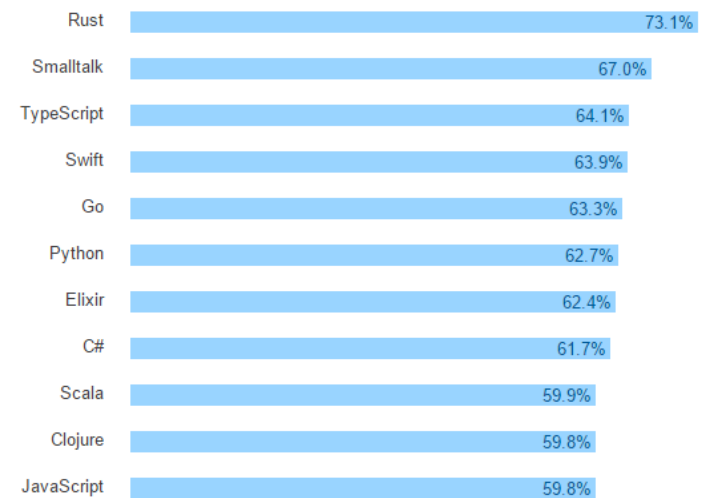
## Most Loved, Dreaded, and Wanted

### Most Loved, Dreaded, and Wanted Languages

Loved

Dreaded

Wanted



# Engines - v8

- Google developed their own JavaScript engine in 2008
  - not an interpreter, a compiler,
  - not bytecode driven (C#, Java) but native machine code
  - open sourced
  - Included in node.js
- <https://code.google.com/p/v8/>

# node.js



- <https://nodejs.org/en/>
- Three features in download
  - advanced JavaScript I/O library
  - npm – a package manager, downloads JS code
  - V8 engine – Chrome's JS execution environment, the same as is built in to their browser
- Used for development workflow
  - build tasks
  - development servers
  - scaffolding





# CDNJS



- Content delivery network for JavaScript
- <https://cdnjs.com/>

# Language support - Nashorn



- Java 8 includes a JavaScript parser

# Background

- Client side or server side
  - Executes in the browser or other tool with engine
- Object based
  - Works with data structures of objects
  - Does not require a class!
- Functional
  - Takes after Scheme more than C.
  - Helps code asynchronously.

# Development process choices

- 1. Run script code in the address bar of a browser.
  - Very old school
  - Used by hackers
    - `<a href='javascript: malicious code....'>Click me!</a>`
- `javascript: document.write('This text was written by JavaScript'); console.log('from the address bar');`

# Development process choices

- 2. Run the same kind of script in the Development Tools of a browser.
  - `document.write('This text was written by JavaScript');`
  - `console.log('from Dev Tools');`

# Exercise

- Try all of the methods to write and execute this code
  - `document.write('This text was written by JavaScript');`
  - `console.out('Program done.');`
- Questions:
  - Does the page you start on affect the outcome?
  - Do you have to be connected to the Internet?
  - Does the page you start on affect the source code?

# Development process choices

- 3. Add the script code to an HTML page.
  - Surround with script element
  - Always try to put your script element at the bottom of the body.
  - Save the file in a WD-405 folder
- Questions
  - Does it matter where the script is located in the html / head / body etc.?
  - How to you get the code to appear on a new line?

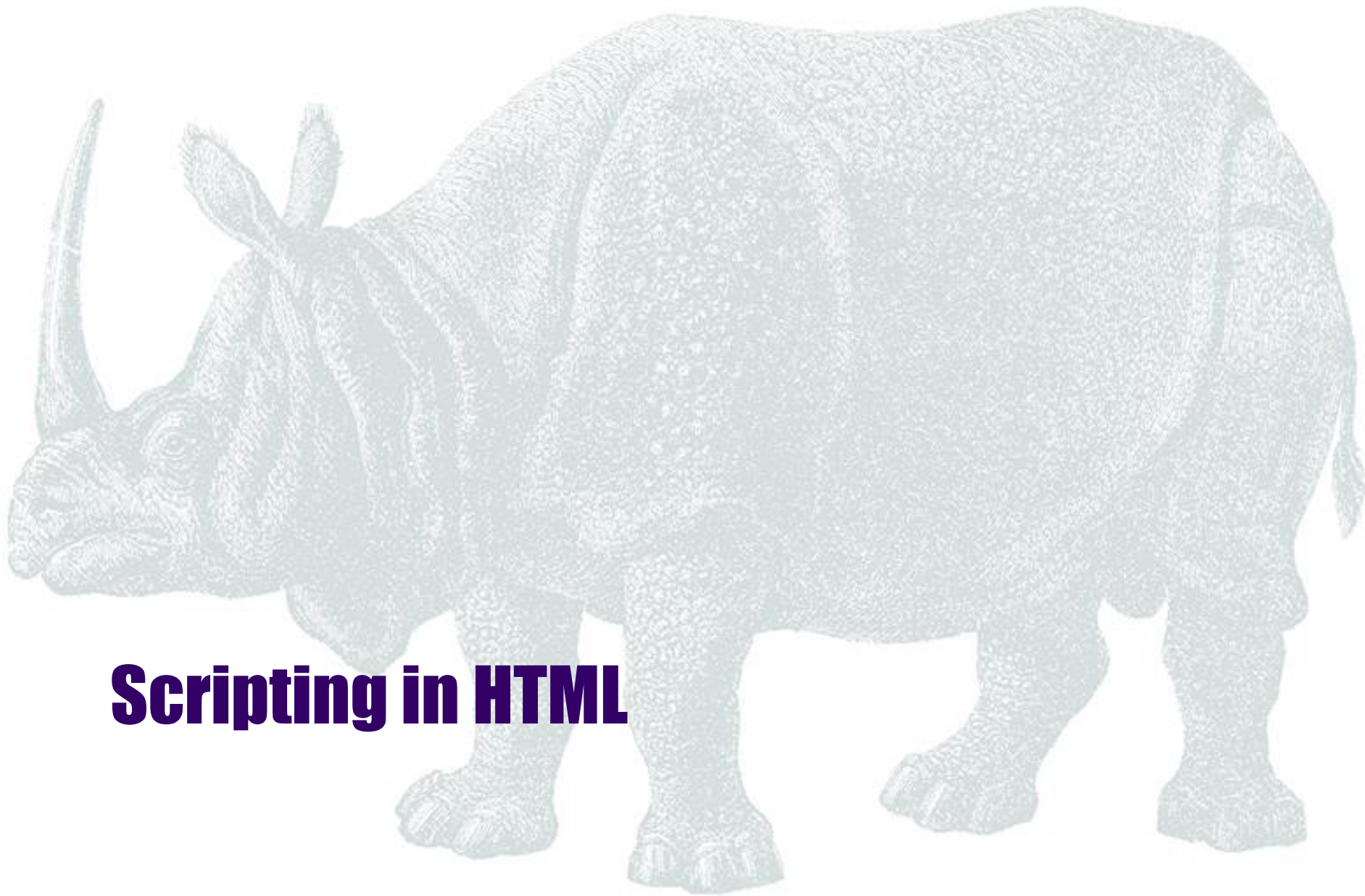
# Development process choices

- 4. Add the script code to an HTML page in an external file.
  - `<script src='write.js'></script>`
- 5. Run the script code in a shell (node.js)
  - Install node.js first
  - Will not process document object commands.



# Exercise

- Create the same code you did before surrounded by the `<script>` element but instead, create a file for it and reference it with
- `<script src='externalFileName.js'></script>`
- Questions
  - Does it work when you make the script element the empty element shortcut `<script />`? In IE?
  - Is something missing in the script attributes?



# **Scripting in HTML**

# Script tags

- Internal on page
  - `<script type="text/javascript">`
  - `</script>`
- External script files
  - `<script type="text/javascript" src="scripts/file.js"></script>`
  - `<script type="text/javascript" src="http://cdnjs.cloudflare.com/ajax/libs/dojo/1.8.1/dojo.js"></script>`
- Using `<script>/**/ ... /*]]&gt;*/&lt;/script&gt;</code><ul><li>● Only necessary for embedded XHTML validation</li><li>● Trend is HTML5 not XHTML.</li></ul></li></ul></div>`

# <noscript>

- Use for output when browsers do not have JavaScript enabled.
  - Major pain
  - Use library instead.
- Better to use Modernizr
  - <http://modernizr.com/>
  - Support for CSS classes and browser features
  - JavaScript API

# "use strict"

- where
  - place as first line outside a function for checking on the external file
  - place inside function to check just that function
- what it checks
  - duplicate object keys
  - variables without var
  - duplicate arguments
  - freezes **arguments** in a function

# Output methods

- `alert( )`
- `document.write(...)`
  - will put contents on a browser page at that point in html
- `console.log(...)`
  - the best way
  - does not work unless DevTools is open in IE8/9!
    - eliminate output error by overriding the function
      - `if (!window.console) {`
      - `console.log = function(){};`
      - `}`

# Comments

- Single line from symbols
  - `//` short comment
- Multiple lines
  - `/*`
  - comments
  - `*/`



Containers to hold data and processes

# **Syntax & variables**



# Code blocks

- Any group of statements between { }
- Can be nested
  - {
    - { }
  - }
- Used to organize code as one unit

# Statements

- an executable set of keywords, operators, & literals
- Usually ends with a semicolon
  - recommended style
  - optional only if on two separate lines
  - common when interactive in browser
- Use as much white space as you want

# Variables

- A placeholder for a value
- No datatype
  - Different than Java, C#, etc.
  - No type is declared, inferred from value.

# Identifiers

- Use well named variable names
- case sensitive
- start with only A-Z, a-z, \$ or \_
- use digits in name only after starting character
- no spaces
- no keywords
- use camelCasing

# var

- var keyword
  - var x;
  - x = 5;
  - var x = 5;
- read backwards
  - 5 is assigned to the variable x.
- used for any type of data

# var

- declares a lexically scoped (not global or block) variable
  - `var x`
  - `x = 5;`
  - `var x = 5;`

# Block scope

- scope = access rights
- JavaScript does not use block scope like all C type languages (C#, Java, ...)
  - `int i = 5;`
    - `{ int k = i + 1; }`
  - `print(i + k);`
- k is limited to the code block it was defined in
- k will not be available now outside the code block

# Global declaration

- initialize without **var** declaration
- `x = 10;`
- variables are available from any function and after execution in environment
- not best practice
- debugging
  - useful for checking jQuery variable states



# Exercise

- Run in your browser
- ```
var i = 5;  
  { var k = 5; }  
  console.log(i + k);
```

# Statement types

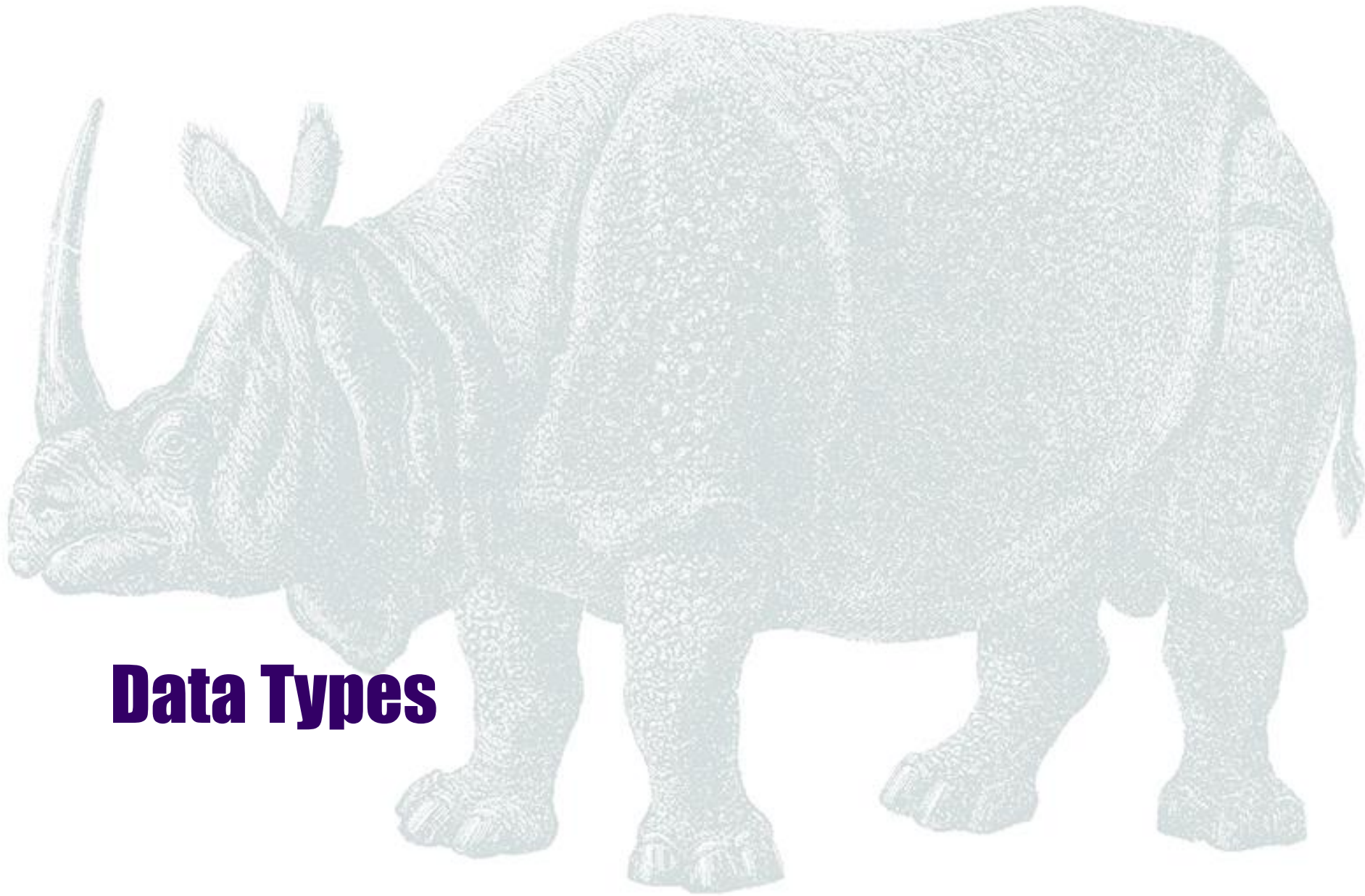
- Value assignment
  - direction is right to left
  - `x = 5;`
  - `x = y; (x ← y )`
- Execute a function
  - `console.log ("Hello, world!");`

# Variables get copied values

- `var value1 = 1;`
- `var value2 = value1;`
- `value1 = 1000;`
- `value2`

# Exercise

- In the browser console, execute this
  - `var x = 5;`
  - `var global = this;`
  - `global`
- Open the global object and see if you can find the value of `x`.
- Is it different in node?



# **Data Types**

# Data types

- Number (integers, floating point, scientific notation)
- String (text)
- Boolean (true, false)
  
- Undefined
- Null
- NaN (not a number)

# Null



## Shark Cordless Pet Perfect II Hand Vac (SV780)

★★★★☆ ▾ 2,033 customer reviews | 215 answered questions

Price: \$57.99 ✓Prime

In Stock.

Want it Monday, March 6? Order within 6 hrs 12 mins and choose Two-Day Shipping at checkout. [Details](#)

Ships from and sold by Amazon.com. Gift-wrap available.

Color: **Lavender**

- NULL
- Made in USA or Imported

## Shark Cordless Pet Perfect II Hand Vac (SV780)

★★★★☆ ▾ 2,125 customer reviews | 229 answered questions

Price: \$53.80 ✓Prime

In Stock.

Want it Monday, May 8? Order within 7 hrs 39 mins and choose Two-Day Shipping at checkout. [Details](#)

Ships from and sold by Amazon.com. Gift-wrap available.

- Nullify
- Made in USA or Imported
- Powerful cordless vacuum. Convenient cleaning for all surfaces.
- Twister Cyclonic Technology. Delivers consistent strong suction while cleaning.

# Data types

- Values
  - one simple value stored
  - text, numbers, true/false
- References
  - multiple values stored
  - like a spreadsheet row
- undefined / null
  - the use of a non-declared value
  - the absence of a value



# NaN

- a special value
- `Math.sqrt(-2)`
- `Math.log(-1)`
- `0/0`
- `parseFloat('foo')`
- `NaN === NaN`
- `isNaN(NaN)`
- But why are `0/1` and `1/0` not NaN?

|                       |     |
|-----------------------|-----|
| > Tech In-depth       | 2   |
| > AngularJS           | NaN |
| > Programming         | 6   |
| > Web Design/Programm | 181 |

# Strings (text)

- defined with ' ' or " "
- Single quotes are a little better to read than doubles, can be confusing when used with other languages.

# Quotes

- Either set of quotes, single or double, mean the same thing.
- Use mixed quotes so you don't have to use escaped quotes
  - 'He said "Yes!" '
  - "I'm in the P 'n' L District"

# Strings (text)

- Escaped characters
  - `"\t"` , `"One\ttwo\tthree"` – tab
  - `'\''` , `'It\'s here!'` – single quote
  - `"It's \"my life\""` – double quote
  - `"First line\nSecond line"` – new line
  - `"First line\n<br/>Second line"` – new line
  - `'ASCII A3 = \xA3'`
  - `'unicode 0xA3= \u00A3'`
    - <http://unicode.org/>
    - <http://www.amp-what.com/>
  - `'ES6 style Unicode code points = \u{A3}'`

# boolean

- `var isHappy = true;`
- `var isSad = false;`
- `var hasHappiness = 1;`
  - any non-zero number is a true value
- `var hasSadness= 0;`

# truthy / falsey

- a value is "truthy" when the value coerces (can be cast by JavaScript) to true when evaluated in a boolean context
  - lots of truthy values, not the same as `== true`
  - only 6 falsy values, `if(<valueBelow>)`
    - `false`
    - `0` (zero)
    - `""` (empty string)
    - `null`
    - `undefined`
    - `NaN`
  - test with - `<value> ? true result : false result`

# Variables in scripts

- Concatenation
  - **var stringVar = 'abc'**
  - stringVar + "text" + stringVar
  - stringVar + 123 + 456
  - 123 + 456 + stringVar
  - **var booleanVar = true;**
  - stringVar + booleanVar
  - stringVar + null
  - stringVar + undefined
  - null + undefined

# xkcd

MY NEW LANGUAGE IS GREAT, BUT IT  
HAS A FEW QUIRKS REGARDING TYPE:

```
[1] > 2 + "2"  
=> "4"  
[2] > "2" + []  
=> "[2]"  
[3] > (2/0)  
=> NaN  
[4] > (2/0)+2  
=> NAP  
[5] > "" + ""  
=> " "+"  
[6] > [1,2,3]+2  
=> FALSE  
[7] > [1,2,3]+4  
=> TRUE
```

```
[8] > 2/(2-(3/2+1/2))  
=> NaN.00000000000000013  
[9] > RANGE(" ")  
=> (' ', '!', ' ', ' ', '!', ' ')  
[10] > + 2  
=> 12  
[11] > 2+2  
=> DONE  
[14] > RANGE(1, 5)  
=> (1, 4, 3, 4, 5)  
[13] > FLOOR(10.5)  
=> |  
=> |  
=> |  
=> |__10.5__|
```

colors.rgb("blue") yields "#0000FF". colors.rgb("yellowish blue") yields NaN. colors.sort() yields "rainbow"



# Exercise

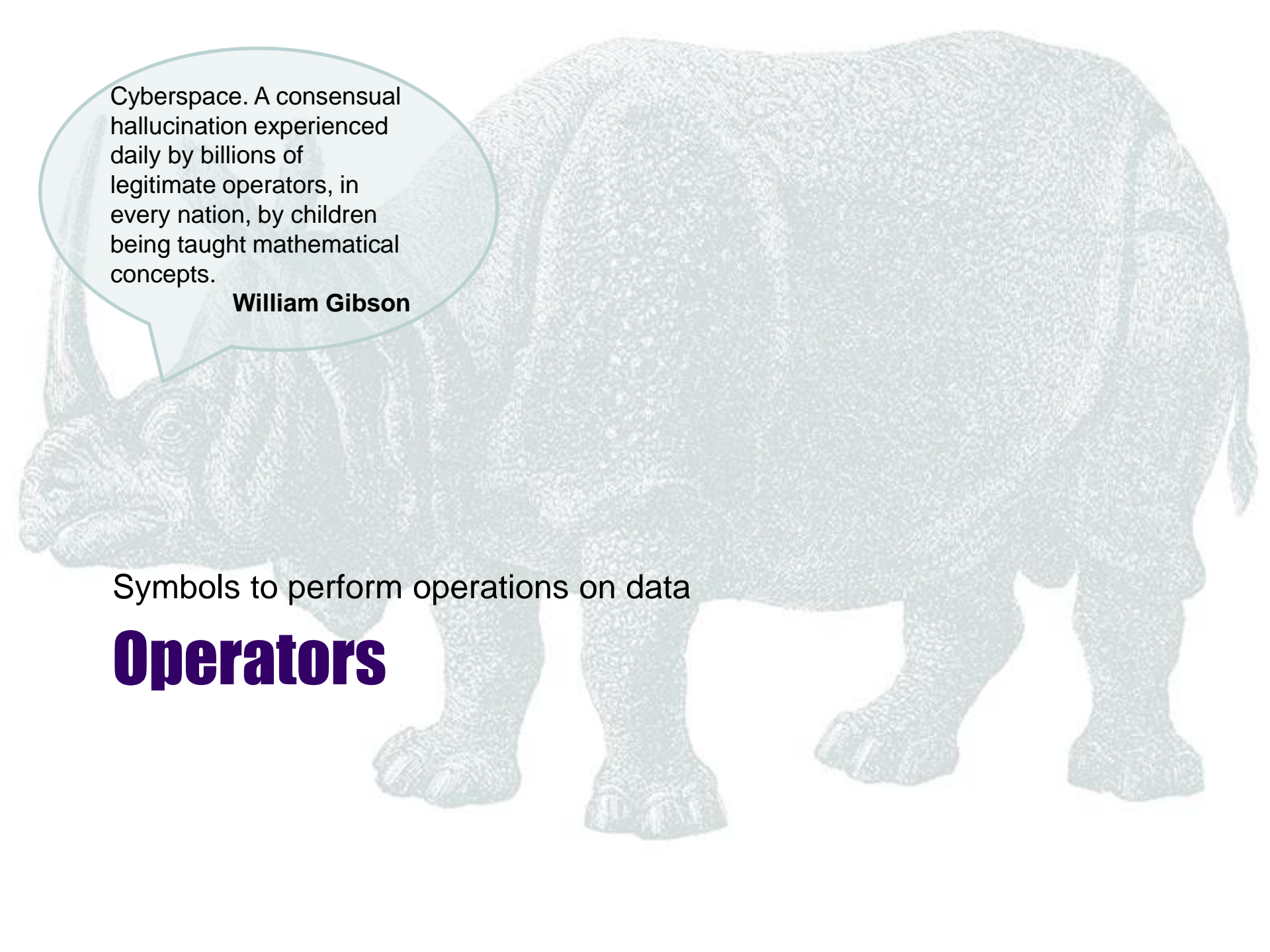
- `var dollar = 1.00`
- `var dime = dollar/10`
- `dime + dime + dime + dime + dime +  
dime + dime + dime + dime + dime`

# Exercise

- $1 + 2 = ?$
- $.1 + .2 = ?$
- what are the expected results of these?
- check in the browser console

# Exercise

- Evaluate these in the browser console
  - `1+1`
  - `2+'2'`
  - `1/0`
  - `-1/0`
  - `5/2`
  - `5/'two'`
  - `5/two`
  - `'two' + 'two'`
  - `1 + 2 > 3`
  - `'1' + 2 > 3`



Cyberspace. A consensual  
hallucination experienced  
daily by billions of  
legitimate operators, in  
every nation, by children  
being taught mathematical  
concepts.

**William Gibson**

Symbols to perform operations on data

# Operators

# Arithmetic

- **+, -, \*, /**
- **% modulus**
  - 5 % 3
  - 5 % 4
  - 5 % 9
- **++, --**
  - `X++ , ++X`
  - `var x = 0;`
  - `console.log(x++);`
  - `console.log(++x);`

# Relational

- `==` is equal to without comparing type?
- `!=` is not equal to?
- `>` , `<` , `>=` , `<=`

# Relational

- **=** means **is assigned the value of**
- **==** means **is it kinda equal to?**
  - loose equality, uses coercion on both operands
  - Douglas Crockford is against
  - generally avoid
- **===** means **is it equal to and of the same type?**
  - strict equality
- tables for **==**, **===**, **+**, **\***
  - <http://zero.milosz.ca/>

# Exercise

- `'0' == 0` (zeros)
- `'1' == 1`
- `'true' == true`
- `true == True`
- `true == 'true'`
- `true == 1`
- `true == 2`
- `false == 'false'`
- `'false' == 'False'`
- `(1.0 + 2.0) === 3.0`
- `(.10 + .20) === .30`
- `NaN === NaN`



# Logical



- Bit-wise
  - will execute always
  - **&** and, **|** or, **^** xor, **!** Not
  - mostly for graphics
  - will work for logical values, but don't

# Logical – short circuit

- `&&` = logical AND, the guard operator
- `||` = logical OR, the default operator
- one true in an OR expression makes it all true
  - `false || false || true`
- one false in an AND expression makes it all false
  - `true && true && false`
- evaluation stops after these two cases
  - `true && true && false && something && something`

## Exercise - || as default

- used for function args not passed in
- function nameIt(myName) {
  - var myName = myName || 'No name';
  - console.log(myName);
- }
- nameIt( );
  
- use || for default value
- use | with numbers for mask

# Converting values

- `var float = 1.11111;`
- `var int = float | 0;`
- `var rounded = (float + .5) | 0`
  
- `int = +'123';`
- `string = " + 123;`

## Exercise - && as guard

- guards against functions running without data
- `isTotallyTrue = true; empty = ''`
- `if (isTotallyTrue) { console.log('dude!'); } // boring`
- `isTotallyTrue && console.log('duuuude!'); // better`
- `!isTotallyTrue && console.log('oh no, duuuude!');`
- `empty.length && console.log(empty.search('a'))`

# Compound operators

- **`+=`, `-=`, `*=`, `/=`**
- **`x+=1`**
  - or `x = x + 1`
  - or `x++`
  - or `++x`
- `x+=10`
- `x*=.90`
- `var abc = 'abc'`
- `abc += 'def'`

# If-else replacement (ternary operator)

- (expression) **?** value if true **:** value if false
- `var items = 0;`
- `console.log(items + " item" +  
( (items===1) ? "" : "s" ) );` // handle plural
- Results
  - 0 items
  - 1 item
  - 2 items

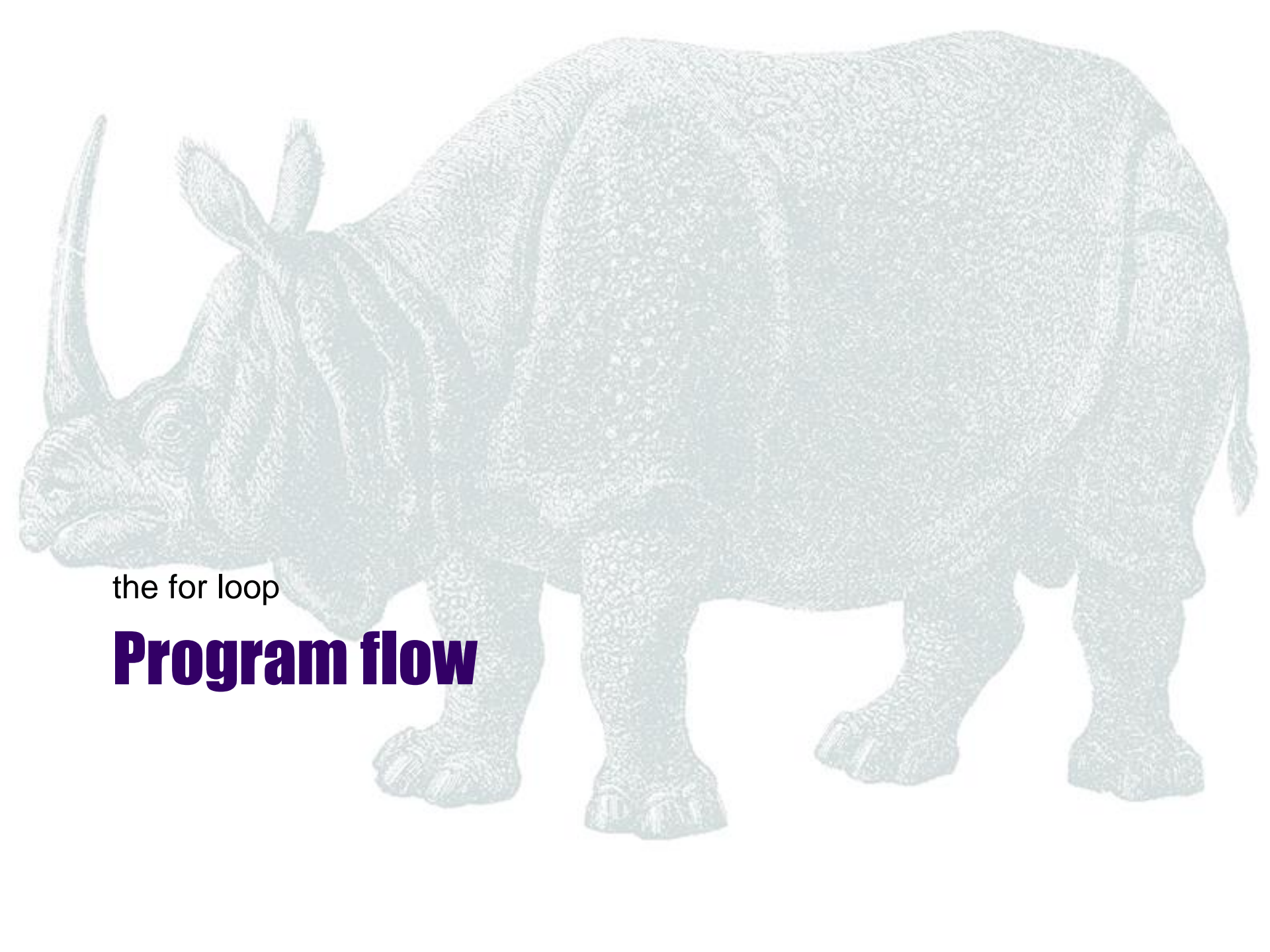
# typeof

- Type returns as a string
  - `typeof(5)`
  - `typeof(true)`
  - `typeof('text')`
  - `typeof(function( ) { } )`
- Useful more for packages/libraries



# Operator precedence

- Order in which operator is executed when other operators are present.
- Directional
- Don't memorize, use parentheses
  - $1 + 2 * 3 + 4$
  - $(1+2) * (3+4)$



the for loop

# **Program flow**

# Program flow types

- Line by line
  - Execute each line until end of script.
- Units
  - Execute reusable modules of code in another place as needed.
- Controller
  - Call units of code to do all the work.

# Program flow types

- Line by line
  - Execute each line until end
- Units
  - Execute reusable modules of code in another place as needed.
- Controller
  - Call units of code to do all the work.

# Program flow types

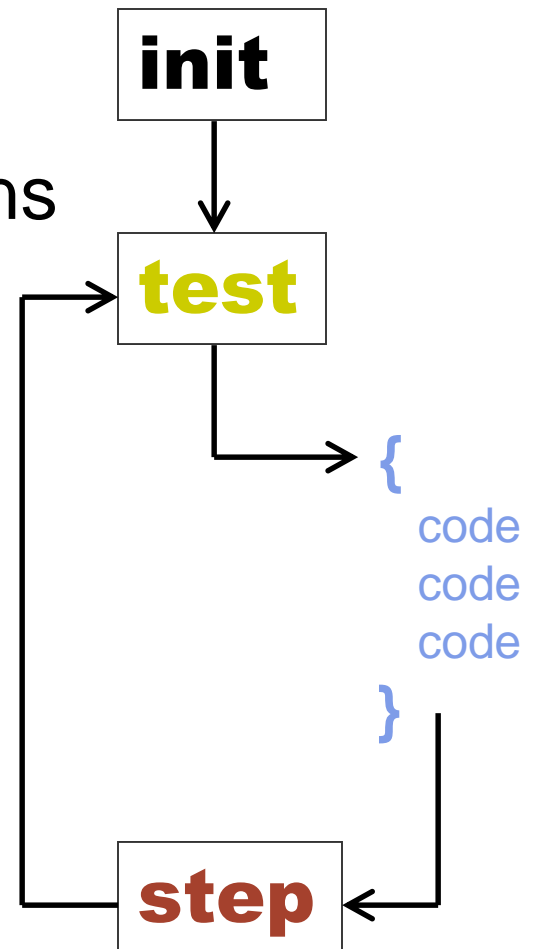
- Branching
  - Check the state of a rule and execute code based on results
- Iteration
  - Execute a unit of code multiple times all at once until a rule is met.

# Program flow types

- Asynchronous calls
  - Execute a unit of code anytime in the near future and get a reply whenever.
  - Two types
    - Distributed – AJAX
    - Non-distributed - Threading, HTML5 Web Workers

# for loop logic

- initialization before 1st iteration
- conditional testing
- code block or statement iterations
- "stepping" statements



# for syntax - init



```
for ( init ; test ; step ) {  
    code  
    code  
    code  
}
```



# for syntax - init

**for** ( **init** ; **test** ; **step** )

- local variables declared and initialized
- multiple variables separated by commas
- can be empty
- Examples:
  - `var i = 0`
  - `var i = 0, j = 1`

# for syntax - test

for ( init ; test ; step )

- result is true or false
- occurs before each iteration
- false result goes to end of code block
- Examples:
  - $i < 10$
  - $i < \text{array.length}$

# for syntax - step

for ( init ; test ; step )

- occurs at the end of each iteration
- Examples:
  - ++i
  - ++i, ++j
  - $j = i * 2$

# Loop keywords - control

- **break**
  - halt all iterations of loop; go to end of code block
- **continue**
  - like break but will start on the next iteration of the loop

# Exercise

- `for ( var i = 0; i < 10 ; i++ ) {`
- `console.log('i = ' + i);`
- `}`
  
- `for ( var i = 10; i > 0 ; i-- ) {`
- `console.log('i = ' + i);`
- `}`



# Exercise

- Print out a list of numbers that:
  - goes from 1 to 25
  - goes from 25 to 1
  - goes from 1 to 50 by 2s
  - goes from 1 to 10 by .5



**Debugging**

# Dev Tools



- Check web page for errors
- Watch network tab for bottlenecks



# console.\*



- Using the built-in development console
  - Safari & Chrome / IE8+ – Webkit Inspector / internal
  - Firefox – add Firebug
  - Opera - Dragonfly
- Standard console messages to output while running
  - **console.log( ),** console.info( ), console.debug( ), console.dir( ), or console.dirxml( )
  - **console.warn( )**
  - **console.error( )**



# console.log(a, b, c...)

- console.log( **arg**, arg, ...)
- console.log(a,b,c, window)
- 1 2 3 ▶ Window {external: Object, chrome: Object, document: document, google: Object, ...}

# console.table( )

- console.table(table or array [, array for headings])
  - output to table format
  - Chrome, Firebug
  - works with object of objects, not array of objects
- ```
var browsers = {  
  ● chrome: { name: "Chrome", engine: "WebKit" },  
  ● firefox: { name: "Firefox", engine: "Gecko" }  
  ● };  
  ●  
  ● console.table(browsers);
```

# debugger;

- Include
  - debugger;
- in your code to stop execution at that point.
- more... <http://fixingthesejquery.com>
- <https://developers.google.com/web/tools/chrome-devtools/>

# Tips

- Set a breakpoint at problem and try `console.log($("#yourSelector").length)`
- Add code before a problem of `console.log($("#yourSelector").length)`
- Add a Watch expression of `$("#yourSelector").length`

# Use Chrome devtools to edit

- Create a workspace
  - drag a local directory onto the source panel
- Map a file
  - with the file open, right click and “Map to file system resource”. Save is now a context option. Includes CSS.



# Chrome Dev Tools

- Node 6.3 (May 2016)
- Debug
  - `node --inspect index.js`
  - `node --inspect --debug-brk index.js` (stop on 1<sup>st</sup> line)
- URL is returned to paste into Chrome
  - right-click and select Mark, click and shift-click first to last column on lines, then just right-click.
  - paste into browser, remove the space added between the lines

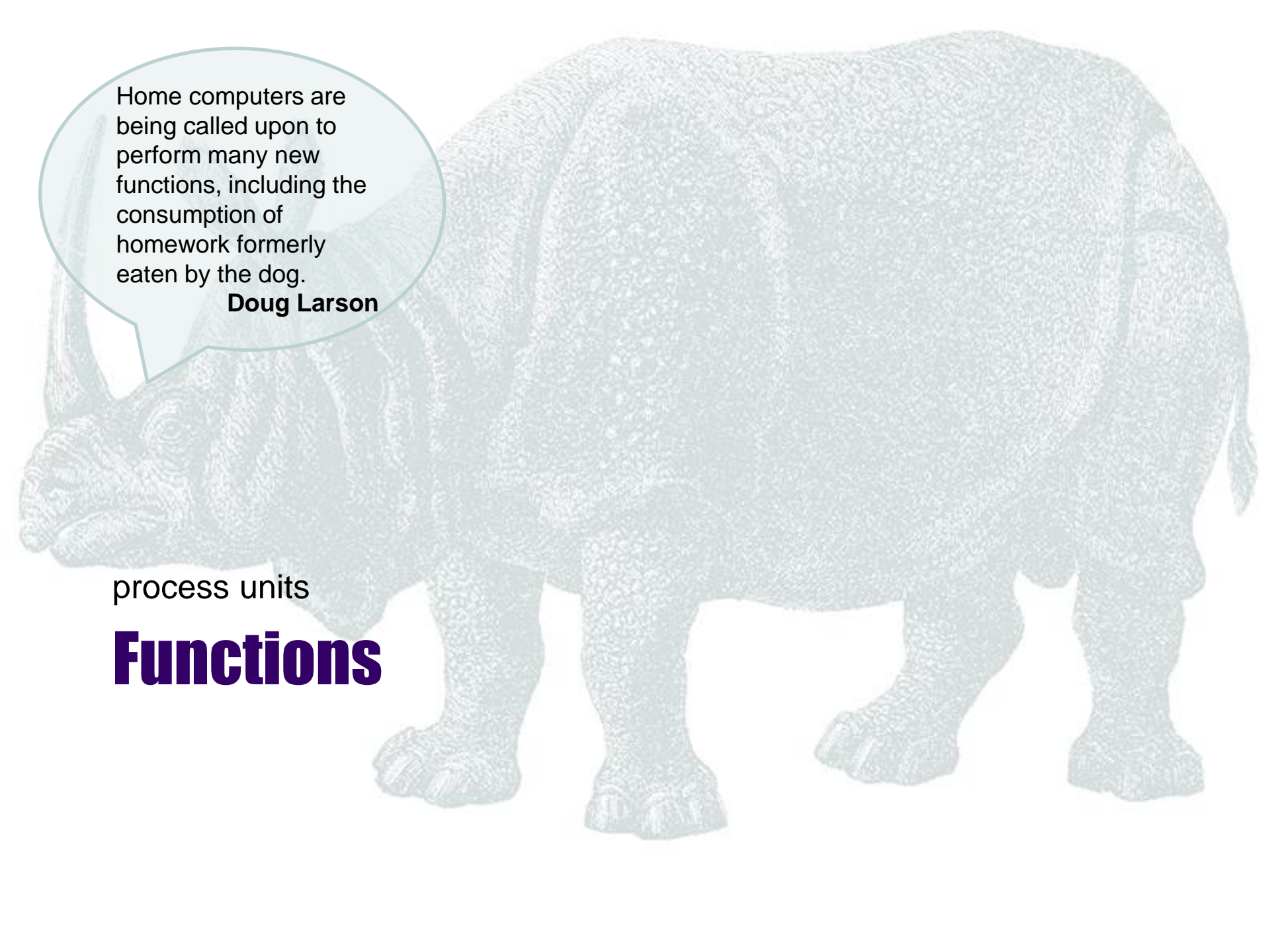
```
C:\Users\Doug\Desktop\TypeScript 2>node --inspect --debug-brk HelloWorld.js
Debugger listening on port 9229.
Warning: This is an experimental feature and could change at any time.
To start debugging, open the following URL in Chrome:
    chrome-devtools://devtools/remote/serve_file/@521e5b7e2b7cc66b4006a8a54cb9c4
e57494a5ef/inspector.html?experiments=true&v8only=true&ws=localhost:9229/node
```

# Use Chrome devtools for node debugging



- Experimental feature
- <https://nodejs.org/api/debugger.html>
- `node --inspect --debug-brk typescript/HelloWorld.js`





Home computers are  
being called upon to  
perform many new  
functions, including the  
consumption of  
homework formerly  
eaten by the dog.

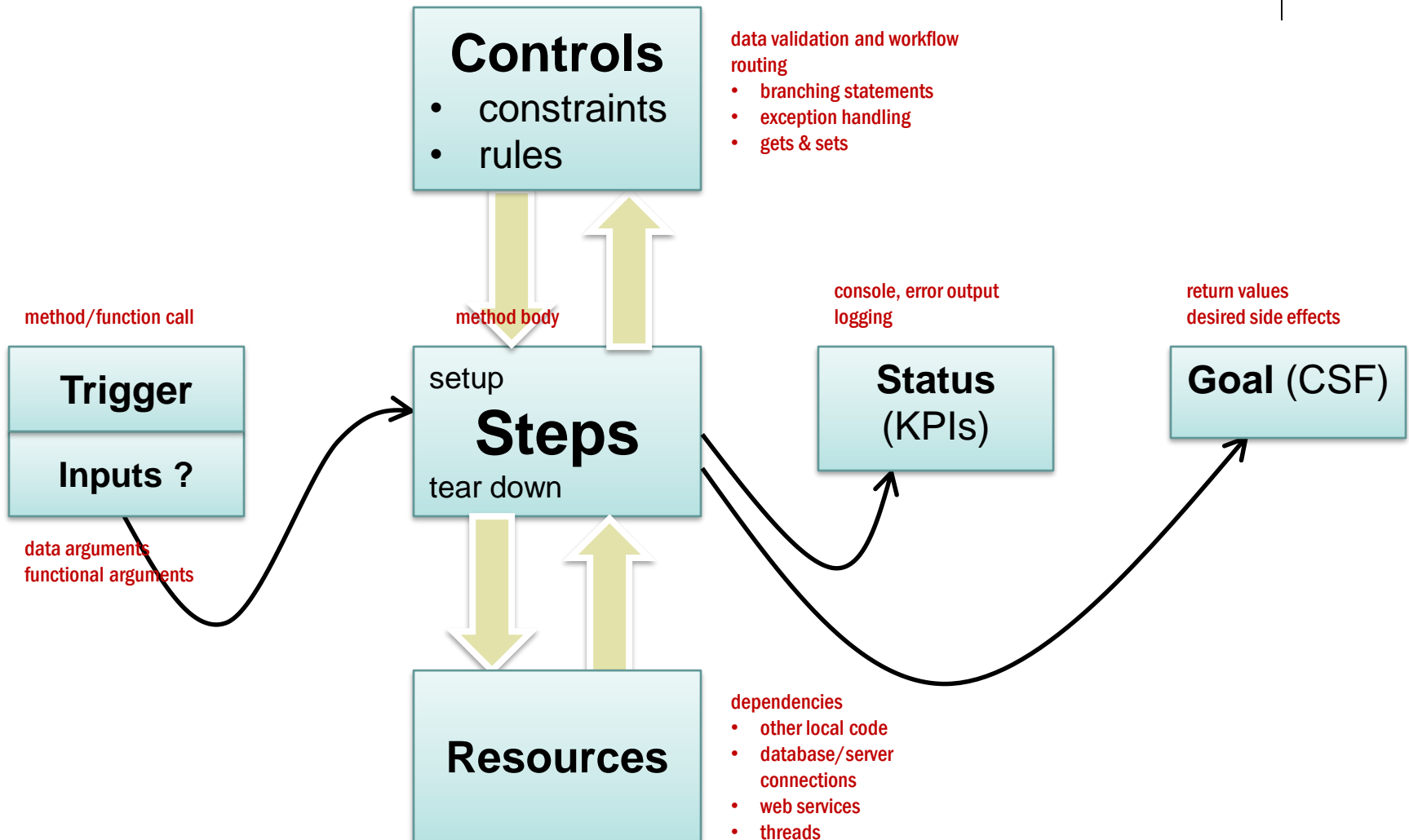
**Doug Larson**

process units

**Functions**

# A process/function model

the process parts in computer language



# Basics

- Groups multiple statements for
  - reuse
  - understanding the purpose
  - calling from the html
- Does not execute unless it is called to execute
- Blocks of code precede the main script

# Syntax

- Declaration
  - **function** doSomething( ) { }
  - **function** doSomething( parameterVar ) { }
  - **function** doSomething( parameterVar, parameterVar ) { }
- { code block for statements; }
- White space is not important so structure how you like.

# Naming

- Use variable naming rules.
- Be descriptive
- Try to start function name with a strong verb.
  - calculateTotalOfLineItems( )
  - printOrderForm( )

# Using parameters

- Creating a variable to receive data when function is called (told to execute)
- Do not use the var keyword
  - **function** doSomething( howManyTimes ) { }
- Scope (when value is available to be used)
  - variable is dead after code block completes
- Arguments are optional to parameters

# Calling functions

- Functions with no return
  - `doSomething( );`
  - `doSomething(5);`
- Function code executes and then script continues from the call.

# Calling functions

- Declaring functions with a return
  - **function** doSomething( ) { return 5; }
  - return is last statement in code block.
- Function call becomes the value returned
  - var number = 3 + 4 + 5 ;
  - var number = 3 + 4 + doSomething( ) ;



# Exercise

- `function doubleMe(aNumber) { return aNumber*2;}`
- `doubleMe; // shows the function`
- `doubleMe();`
- `doubleMe(3);`
- `doubleMe(3,4);`
- `var f1 = doubleMe;`
- `f1(3);`

# Pre-defined functions

- Functions are pre-defined for browser objects
- The object precedes the function call with a dot
- the document object
  - represents the html page
  - `document.write("text");`
- the window object
  - represents the browser
  - `window.alert("text")`



# Exercise

- `function hey(you) { alert('hey! ' + you); console.log('hey ' + you); }`
  - `hey('Doug');`
- `function beAlert(aPerson) { hey(aPerson); console.log('beAlert ' + aPerson); }`
  - `beAlert('Doug');`
- `function makeStuffHappenTo(who) { hey(who); beAlert(who); console.log('makeStuffHappenTo ' + who); }`
  - `makeStuffHappenTo('Doug');`



# Exercise

- `<!-- lasagna.html -->`
- `<html>`
- `<head>`
- `<script src='lasagna.js'>`
- `</script>`
- `</head>`
- `<body>`
- `This is a web page.`
- `</body>`
- `<script>`
- `lasagna = makeLasagna();`
- `// show me the lasagna`
- `</script>`
- `</html>`
- `// lasagna.js`
- `function makeLasagna() {`
- `// do stuff here to make it`
- `...`
- `return someLasagna;`
- `}`



# Exercise

- // Multiple references to functions
- function doFunction1( ) {
  - return "function1";
- }
  
- var f2 = doFunction1;
- f2( );

# Function scope

- Variables declared with var inside a function are only accessible to that function.
- Functions use code blocks but it's about the function not the code block.

# Variable hoisting

- Hoisting = moving all declarations to the top of the current scope (function scope)
- Initializations (getting a value) are not hoisted
- `var x, y; // hoisted placement in code`
- `var sum;`
- `sum = (x + y) | 0 ;`
- `console.log('before inits: ' + sum);`
- `var x = 1;`
- `var y = 2;`
- `// run twice to use inits`

# Function hoisting

- using a function before it's declared
  - does not work with assigning anonymous function to variable
- `iveBeenHoisted( );` // call before declaration
- `iCantBeHoisted( );` // won't run
- `function iveBeenHoisted( ){`
  - `console.log("Hey, I'm here! ");`
- `}`
- `var iCantBeHoisted = function ( ) {console.log('?');}`





# Exercises

- Set up web page to load JavaScript from a functions.js file.
- Load the page and execute the functions in the console of the browser that are the tests at the bottom of the JavaScript.
- doAllFunctions(string1, string2) is the refactored version of all the functions to a new function using the default operator ||

# Math



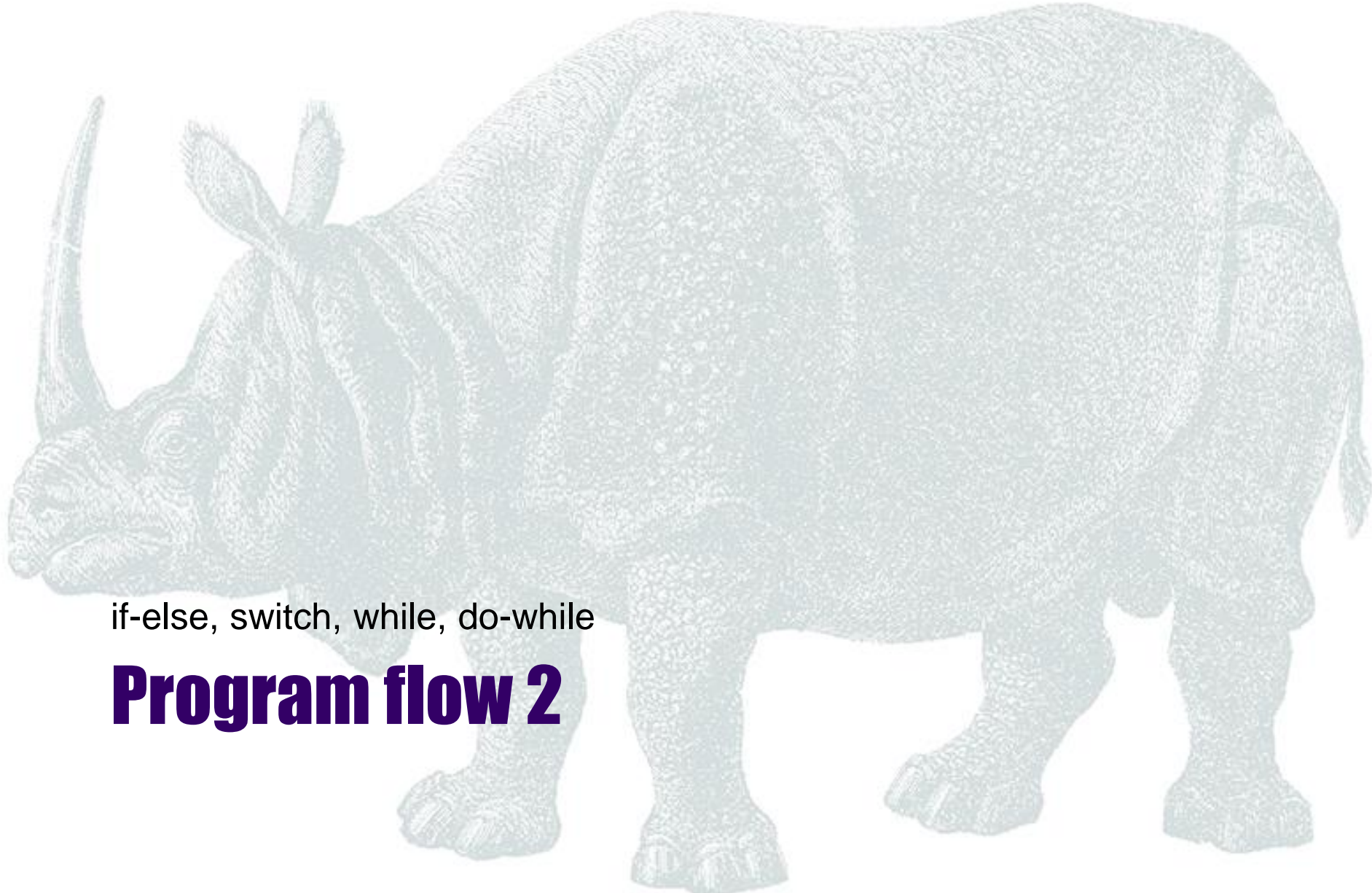
- Math contains many utility functions
  - `Math.sqrt(4)`
  - `Math.random()`
  - `Math.pow(2,8)`
  - `Math.round(2.4)`
  - `Math.abs(-2.7)`
- Math contains several reference values
  - `Math.E`
  - `Math.PI`

# Exercises

- Write and call a function that will
  - Calculate the area of a circle ( $\pi * r^2$ )
  - Calculate the volume of a sphere ( $\frac{4}{3} * \pi * r^3$ )
  - Use a radius variable
- Useful Math stuff
  - `Math.pow(2,8)`
  - `Math.PI`

# Using a function as an argument

- `function doFunction(f) { console.log(f()); }`
- `doFunction(calcAreaOfCircle);`



if-else, switch, while, do-while

## **Program flow 2**

# Conditional basics

- if (logical expression or value coerced to boolean)
  - { then do something because it's true }
- else
  - { do something because it's false }
- Put code for each section in a { code block }
- Don't use if (x = 3) { ... }
  - always true!

# Conditionals

- **if** (this is true)  
    execute this statement;  
    but this statement will always run;
- **if** (this is true) {  
    execute this entire code block;  
}
- **if** (this is true)  
    { execute this block; }  
  **else**  
    { execute this block; }

# Nesting ifs

- `if (part one) {`
  - `if (part two) {`
    - do something only if both are true
- `}`
- `if ( part one && part two) {`
  - do the same thing since both are true }
- Try to avoid nesting.



# Checking values with switch

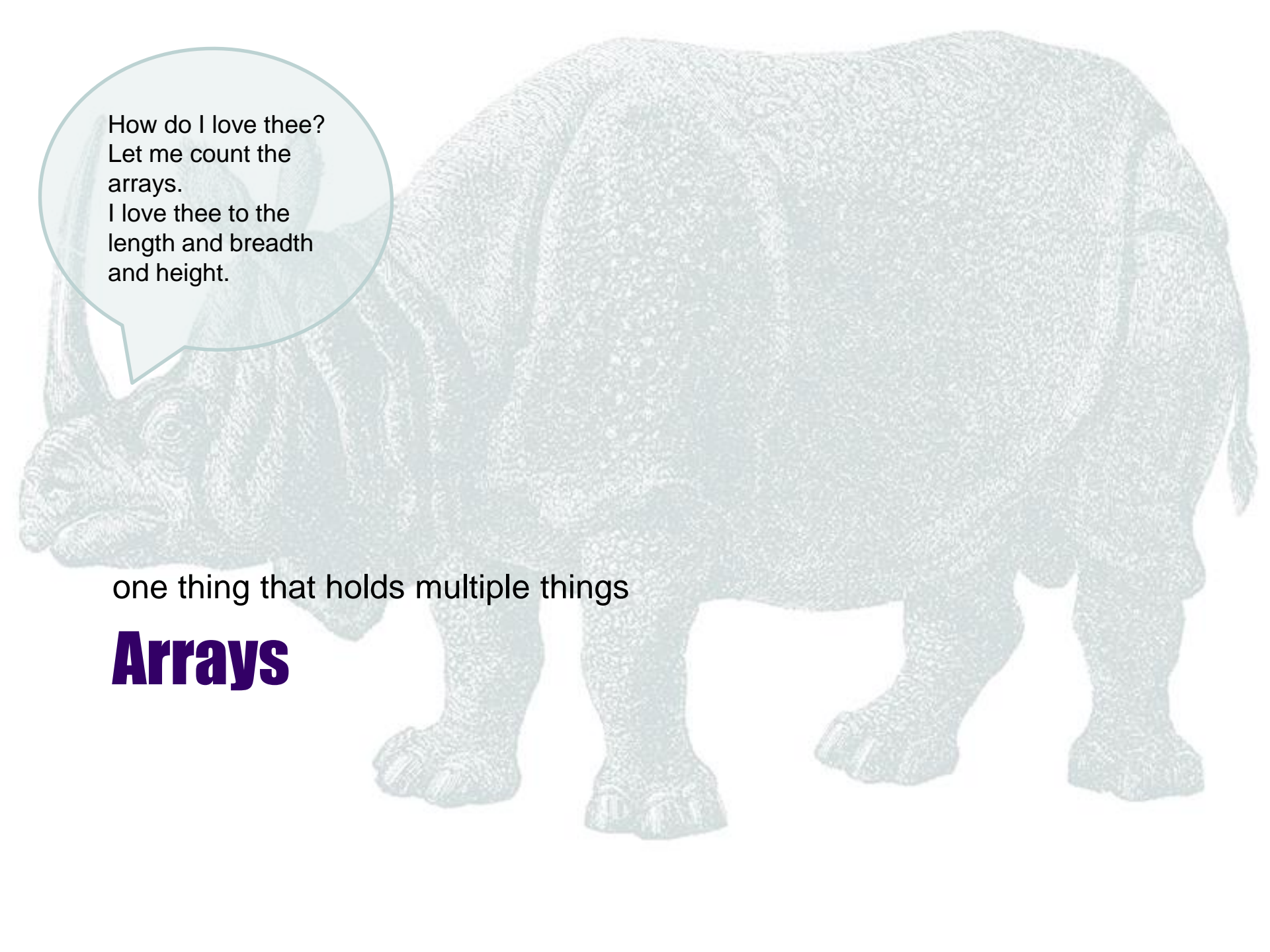
- Use to test variable for one of a set of values.
- Allows for "none of the above" as an option. (default)
- When a match is found in a case, execution continues
- Execution of code only stops with a **break**;
- Break jumps out of the switch block.
- `switch (thingToTest) {`
  - **case** 'a': // do stuff
  - `break;`
  - **case** 'b': // do stuff
  - `break;`
  - **default:** ...

# while loop

- **init** a variable
- while (**test** is true) {
  - do code;
  - **step** the variable;
- }
  
- `var i = 0;`
- `while (i <= 10) {`
  - `console.log(i);`
  - `i++;`
- `}`

# do while loop

- **init** a variable
- do {
  - do code at least once no matter what;
  - **step** the variable;
- } while (**test** is true)



How do I love thee?  
Let me count the  
arrays.  
I love thee to the  
length and breadth  
and height.

one thing that holds multiple things

# Arrays

# Creating + initializing

- An array is a data structure to hold many values **of many types**.
- Using array initializer - **best**
  - `var anArray = [1, ' 2 ', true];`
  - `var emptyArray = [ ];`
  - `var twoArrays = [anArray, emptyArray];`

# Accessing and length

- By integer indices with zero based numbering
  - `arrayOfValues[0]` - first element
  - `arrayOfValues['0']` - also valid
- `.length` - how many indices between the first and the last
- `.length = #` - will change length to # indices (lossy)
- check for element: boolean
  - 0 in `arrayOfValues`
  - 1 in `arrayOfValues`, etc.

# Initializing

- Arrays can be added to!
- Try this
  - `var emptyArray = [ ]`
    - or: `var emptyArray = new Array();`
  - `emptyArray[0]`
  - `emptyArray[0] = 1;`
  - `emptyArray[1] = 2;`
  - `emptyArray`
  - `1 in emptyArray`
  - `2 in emptyArray`



# Sparseness

- A sparse array
  - `var almostEmptyArray = [1] // store a number in index 0`
  - `almostEmptyArray[100000] = 1;`
  - `almostEmptyArray.length`
- Another one
  - `var bunchesUndefined = [1, , , , , , , , , , , , ];`
  - `bunchesUndefined.length`



# for loop

- `var index;`
- `var a = ["a", "b", "c"];`
- `for (index = 0; index < a.length; ++index) {  
 console.log(index, a[index]);  
}`

# Indexes & properties

- Arrays can be initialized with either **indexes**
  - `helloArray[0] = 'hello'`
  - `helloArray[1] = 'hi'`
- Or **properties** (associative)
  - `helloArray['cajun'] = "how y'all are?"`
- Pick one style but not both - confusing

# for-in

- Arrays
  - loops through indexes **and** properties of arrays
  - skips undefined values - useful for sparse arrays with checks
- Objects
  - loops through the *enumerable properties of an object*
  - ```
for(item in window) {  
    console.log(item, window[item])  
}
```

# for-in

- `cars =`  
`[,,,,,,,,,'Prius',,,,,,,,,,,,,,'Saab',,,,,,,,,,,,,,,,,,,,,,'Ford'];`
- `cars['vendor'] = 'Hoff Used Cars';`
- `for(carKey in cars){`  
    `console.log(carKey, "=", cars[carKey]);`  
    `}`
- `cars // view result in browser`

# Exercise

- `var eggCarton = new Array(12);`
- `var results = ""`
- `eggCarton[0] = "a medium white egg";`
- `eggCarton[17] = "the last egg";`
- `eggCarton // show in browser`
- `for (key in eggCarton) {`
- `results += "(" + eggCarton[key] + ") ";`
- `}`
- `console.log(results);`

# Exercise

- `var arrayOfCars=['BMW','Volvo','Saab','Ford'];`
- `var i=0;`
- `while (arrayOfCars[i])`
  - `{`
    - `document.write(arrayOfCars[i] + "<br>");`
    - `i++;`
  - `}`
- Rerun with for-in
  - `for (var car in arrayOfCars) {`
    - `document.write(arrayOfCars[car] + "<br>");}`

# Array indexes & properties

- Positive integers and types coercable (changed by JS) to positive integers (strings, floating point numbers) become indexes.
- Anything else is a property
- Try it
  - `var lettersWithMetadata = ['a', 'b', 'c'];`
  - `lettersWithMetadata['charset'] = 'Unicode';`
  - `lettersWithMetadata`
  - `lettersWithMetadata['charset']`
  - `lettersWithMetadata['0']`
  - `lettersWithMetadata[1.000]`

# Array indexes & properties

- `var two = 2`
- `lettersWithMetadata[two]`
- `lettersWithMetadata.charset`
- Alternative dot notation syntax
  - `someArray.keyUsingName`
  - not as flexible but easier to type



# Property (associative) arrays

- An array based on properties rather than numbers.
  - Also known as a map, dictionary, or key-value pair
- `poolBalls['yellow'] = 1;`
- Think of it as storage like an object field or metadata to an array.



# Exercise

- `poolBalls = [ ];`
- `poolBalls['yellow'] = 1; poolBalls['blue'] = 2;`
- `poolBalls['red'] = 3; poolBalls['white cue ball'] = 0;`
- `var results = "";`
- `for (key in poolBalls){`
- `results += key + ' = ' + poolBalls[key] + "; " ;`
- `}`
- `console.log(results)`
- `poolBalls = ['pool cue', 'rack', 'extender' ];`
- `console.log(poolBalls.length)`
- `// rerun for loop above`
- `console.log(results)`

array **properties**

key= yellow

value = 1

show all key/values

array **index**

key = 0

value = pool cue

length only for

index

# The arguments array

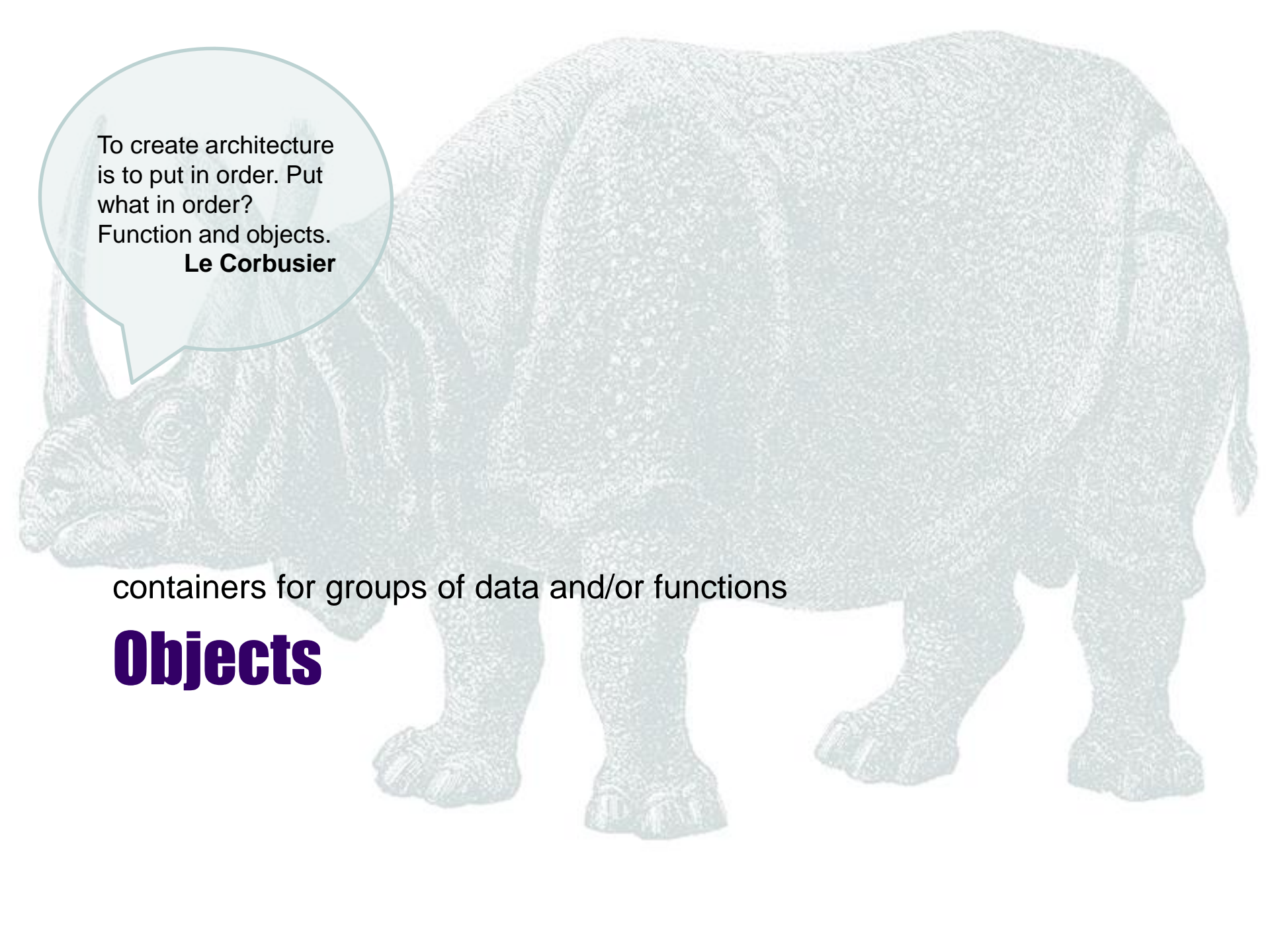
- arguments = data passed to a function
- parameters = variables used to name the arguments
- the keyword **arguments** is an array used in a function
  - `function foo() { console.log(arguments); }`
  - `foo(1,2,3)`

# Exercise

- `function args(a,b,c){`
- `for (argKey in arguments) {`
- `console.log(argKey, '=', arguments[argKey]);`
- `}`
- `};`
- `args(1,2,3,4,5);`

# Array of functions

- Try it
  - `function a(){console.log('a');}`
  - `function b(){console.log('b');}`
  - `var functions = [a,b];`
  - `a( )`
  - `functions[1]( )`



To create architecture  
is to put in order. Put  
what in order?  
Function and objects.  
**Le Corbusier**

containers for groups of data and/or functions

**Objects**

# Intro



- similar to an array but a different datatype

# A relational data structure

- A relational database
  - is defined by a **table** (the schema)
    - divided into **fields**
      - assigned a data type
  - has data created in **rows**
    - is uniquely defined by a primary key
    - can tie to other rows by storing a foreign key
  - sometimes has a **stored procedure** that operates on a table of data



# An object data structure

- A object
  - is defined by a **class** (the schema, prototype)
    - divided into **fields** or properties
      - assigned a data type
  - has data created in **objects**
    - is uniquely defined internally
    - can tie to other objects by storing a reference
  - sometimes has a **method/function** that operates on a class of data

# Relational vs. OO

- Program entities = database = spreadsheet
- Class = table schema = Excel tab
- Object = row of data
  - Instance = object
  - Instantiate = create

# Objects

- Data structures are like a database record
  - object = row
  - field, property = column of table
  - all objects of same type = table of data
  - methods/functions = ~stored procedures
- Accessing values
  - object.field
- Calling functions defined for that type of data
  - object.function()

# Creating an object - object initializers

- Simpler than the constructor
- `objectName = {propertyKey:value, pk:v, ...}`
- `var myBear = {color:"brown", state:"Alaska"};`
- `myBear.color`
- Array syntax is clumsier
  - `var myBearArray = [ ];`
  - `myBearArray['color'] = 'brown';`
  - `myBearArray['state'] = 'Alaska';`

# Creating an object - constructors

- Looks like a function
  - `function Bear(firstArg, secondArg) {`
    - `this.color = firstArg;`
    - `this.weight = secondArg;`
    - `this.state = "Alaska";`
  - `}`
- no return value
- Arguments passed to parameters are assigned to on-the-fly properties in the code block

# Creating an object - constructors

- `var myBear = new Bear("black", 550)`
  - uses pre-existing fields
- `var yourBear = myBear;`
  - two references to the same object
- `myBear = null;`
  - this reference does not point to a bear anymore.

# Using the properties

- `myBear.color`
- `myBear.color = "brown"`
- `var yourBear = myBear;`
  - `yourBear` points to the same object that `myBear` points to
- `yourBear.color = "black"`
- `myBear.color` ?
  - is now "black"

# Adding functions

- Create the function
  - `function growl( ) { console.log('Grrrrrr'); }`
- Add the method
  - `myBear.makeSound = growl;`
- Call the method with the object
  - `myBear.makeSound( );`
- Create and add together
  - `myBear.growl = function( ){console.log ('Grrrrrr'); }`
  - `myBear.growl( )`



# Composition

- Add a friend to your bear
  - `var polarBear = {color:"white", country:"Canada"};`
  - `myBear.friend = polarBear;`
  - `myBear.friend.name`
  - `myBear.friend.growl = growl;`
  - `myBear.friend.growl( )`

# for-in

- also cycles through the properties of an object
- uses array syntax of [ ] for value
  - myBear = {color:"brown", country:"Canada"};
  - var results = "";
  - **for** (var key **in** myBear) {
  - results += (key + '=' + myBear[key] + "\n") ;
  - }

# Object or array syntax

- Object.syntax uses property names only
- array["syntax"] uses property names as values
  - and allows use of variables to represent property names
  - var prop = "syntax";
  - array[prop]

# Treating objects like an array

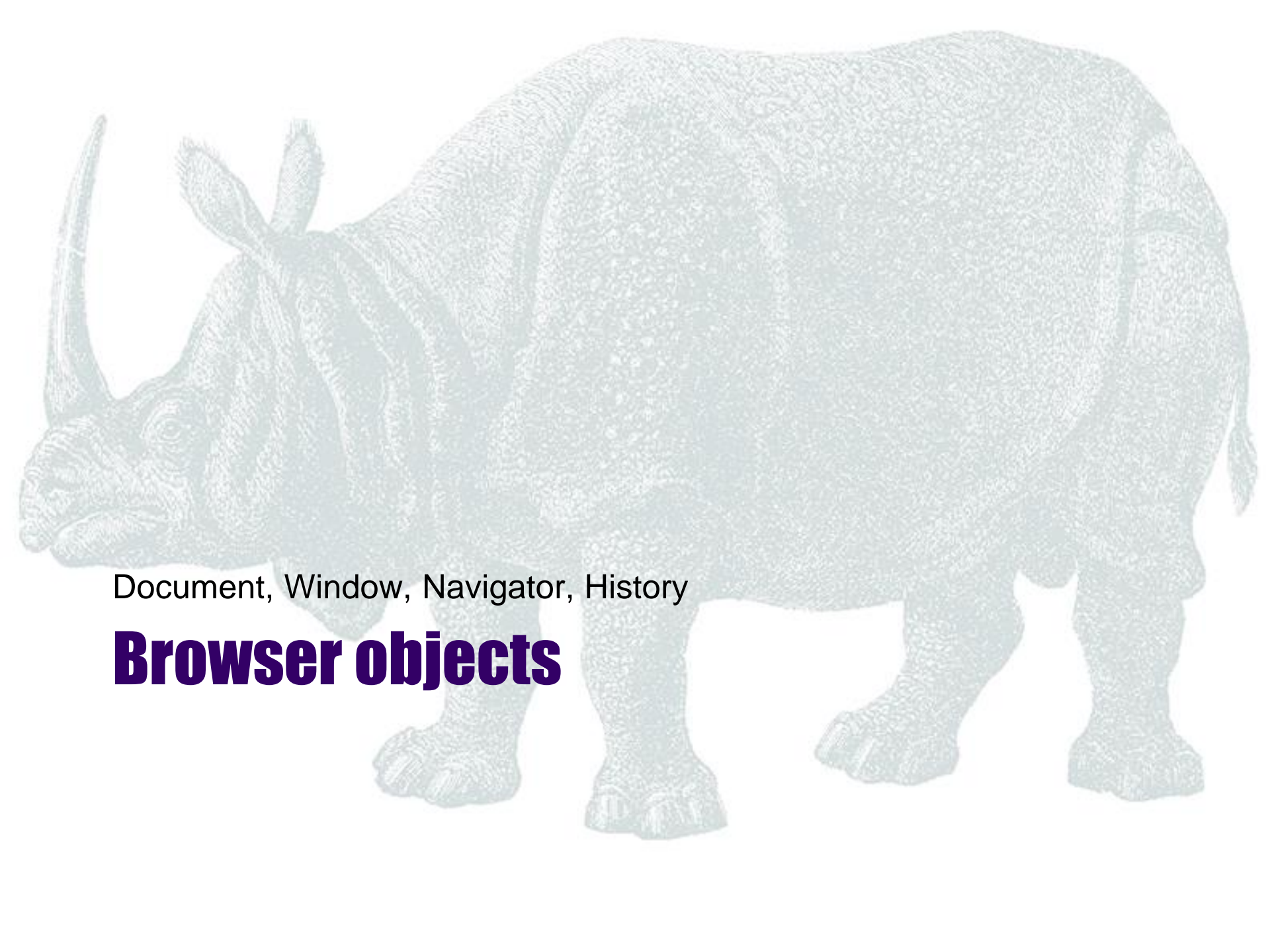
- `var cars = { 1:'BMW', 2:'Volvo', 3:'Saab', 4:'Ford' }`
- `var i=1;`
- `while (cars[i])`
  - `{`
    - `console.log(cars[i] + "\n");`
    - `i++;`
  - `}`
- Must use only numeric keys
- Better to use for-in

# Context sensitive function

- `function listAll(){`
  - `for(key in this){`  
`console.log(key, '=', this[key]);`  
`}`
- `}`
- this refers to the object it's associated with
- `polarBear.listAll = listAll`

# Exercise

- Create a **Dog** object with a few fields
- Create a **Person** object with a few fields
- Composition
  - Add an owner field to the **Dog** object and set the Person to it
  - Add a pet/dog field to the **Person** object and set the Dog to it
- Expand the object in the browser



Document, Window, Navigator, History

# **Browser objects**

# the DOM



- Document Object Model
- The in-memory XML structure of the html parsed into objects (nodes) so JavaScript can talk to it.
- DOM Levels 1, 2, 3 (deprecated)
  - DOM Living Standard
    - <http://dom.spec.whatwg.org>





# Navigator properties

- navigator.appCodeName
- navigator.appName
- navigator.appVersion
- navigator.cookieEnabled
- navigator.language
- navigator.userAgent

# Exercise

- // print the keys and values of the Navigator object
- ```
for (var key in navigator ) {  
    console.log(key, ' = ', navigator[key],  
    '\n' +  typeof(navigator[key]) );  
}
```

# History properties

- `history.length`
- `history.back()`
- `history.forward()`
- `history.go(#)`
- `history.length = 0` // does not clear history, only in plugins

# Exercise

- print the keys and values of the History object
- print just the last URL of the history

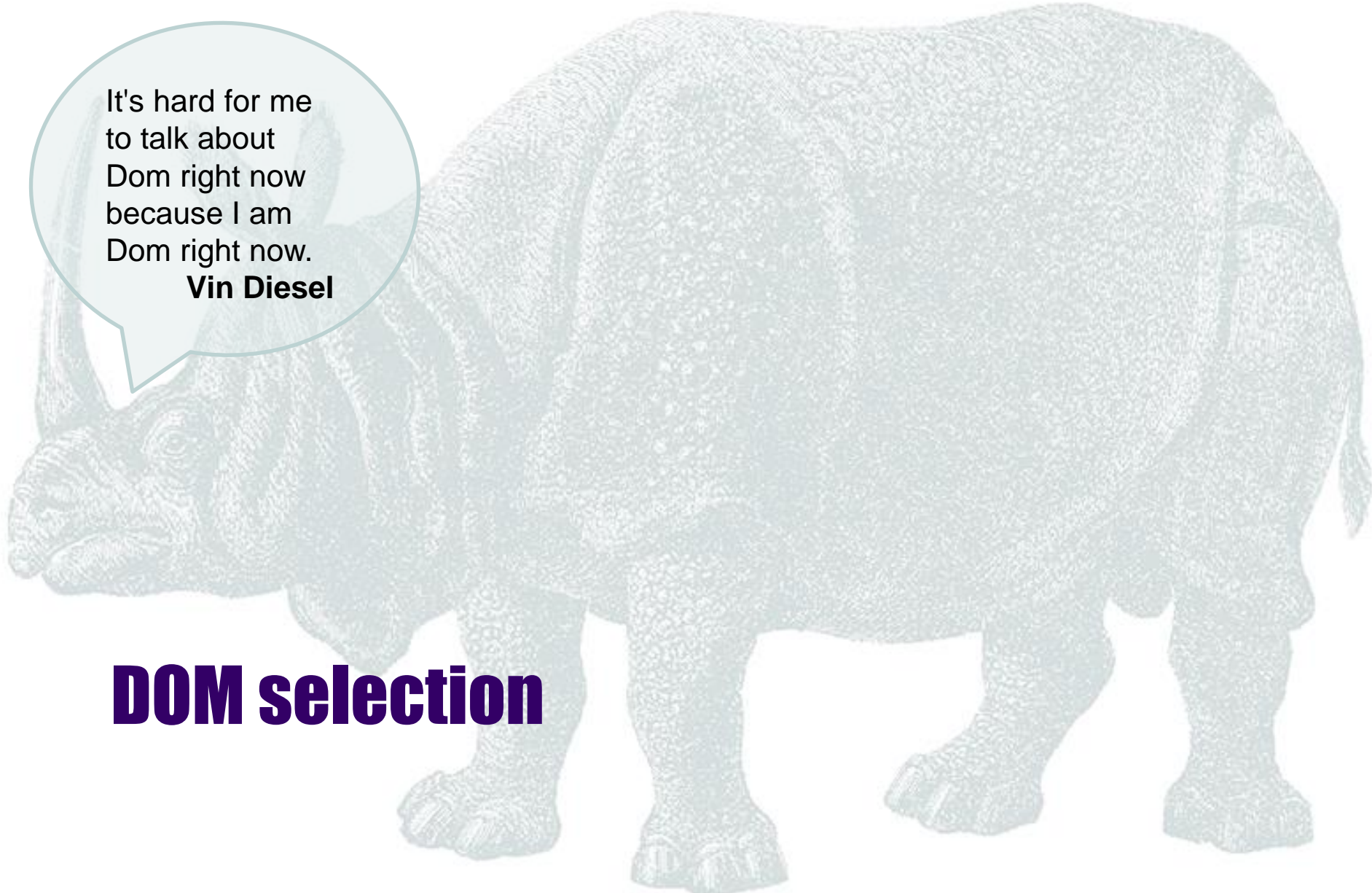
# window



- The top level, global namespace
- Represents the frame of the document ("the chrome")
- Contains
  - location
  - screen info
    - innerHeight
    - innerWidth
  - status bar (message at bottom)

# Request / redirect

- *window.location* = someURL
  - load the URL into the browser
    - Use protocol (http...)
  - also *window.location.href* = someURL
- a **redirect** is a client-side request after a response
- a **forward** is a server-side action returning a different page



It's hard for me  
to talk about  
Dom right now  
because I am  
Dom right now.

**Vin Diesel**

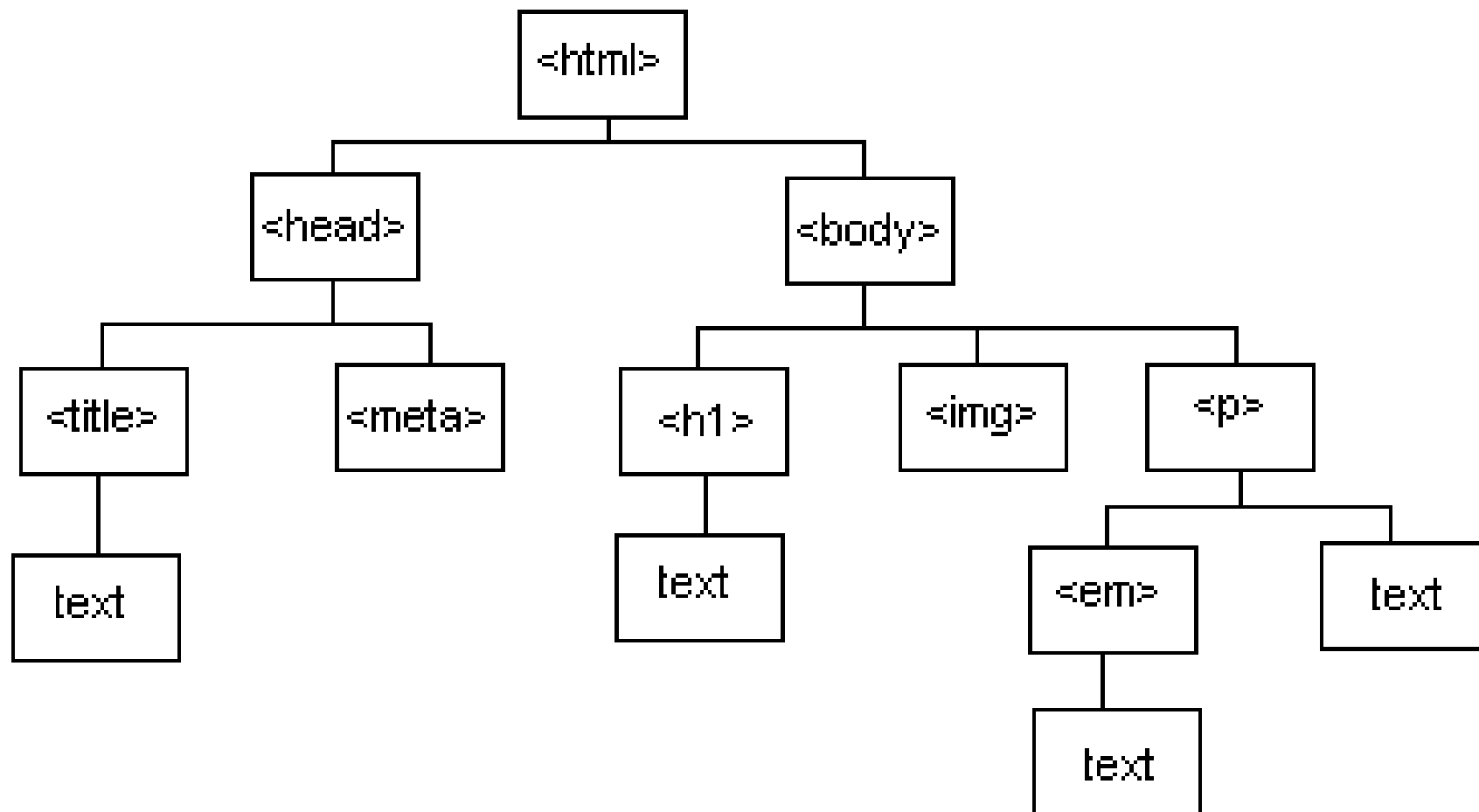
**DOM selection**

# Nodes

- Documents are made up of nodes
- Nodes have parent / child relationships
  - Element → attribute, text, other elements
- Functions
  - node.nodeName
  - node.parentNode
  - node.childNodes







# A DOM tree of nodes



# Node types

- Element
- Attribute
- Text
- and all the others...

| Name  | Value |
|---|-------|
| ELEMENT_NODE  | 1     |
| ATTRIBUTE_NODE         | 2     |
| TEXT_NODE   | 3     |
| CDATA_SECTION_NODE     | 4     |
| ENTITY_REFERENCE_NODE  | 5     |
| ENTITY_NODE            | 6     |
| PROCESSING_INSTRUCTION_NODE   | 7     |
| COMMENT_NODE  | 8     |
| DOCUMENT_NODE   | 9     |
| DOCUMENT_TYPE_NODE  | 10    |
| DOCUMENT_FRAGMENT_NODE  | 11    |
| NOTATION_NODE        | 12    |

# document properties

- some deprecated, some non-standard
  - bgColor
  - body, title
  - referrer
  - cookie
  - URL, domain
  - lastModified
- shortcuts
  - links, images, forms, stylesheets

## Exercise: document shortcut properties

- How many **links** are on the cnn.com page?
- How many **images**?
- How many **forms**?
- How many **stylesheets** do they use?
- Extra credit
  - How many frames are contained in the window?

# Selecting nodes

- **getElementById ('tb\_name')**
  - will find
    - `<input id='tb_name' type=text></input>`
  - also
    - `window.tb_name`
  - jQuery: `$('#tb_name')`
- **getElementsByName('cb\_removeItem')**
  - will find
    - `<input name= cb_removeItem type=radio value='1'>`
    - `<input name= cb_removeItem type=radio value='2'>`
  - jQuery: `$("[name='cb_removeItem']")`

# Selecting nodes

- **getElementsByTagName("div")**
  - will find
    - `<div id='main'>...</div>`
    - `<div id='footer'>...</div>`
  - jQuery: `$('#div')`
- **\* is a universal selector**
  - `getElementsByTagName(" * ");`
- **also in Element class so you can do this:**
  - `var firstDiv = getElementsByTagName("div")[0];`
  - `var imgsInDiv = firstDiv.getElementsByTagName('img')`

# Selecting nodes

- **getElementsByClassName("subhead")**
  - will find
    - `<p class='subhead'>...</p>`
    - `<span class='subhead'>...</span>`
  - jQuery: `$('.subhead')`

# Traversal

- Nodes are XML based and have useless information on comments and text nodes
  - parentNode, childNodes, firstChild, lastChild, nextSibling, previousSibling
  - nodeType, nodeValue, nodeName
- Element type node traversal is newer and much easier and only works on elements
  - children, firstElementChild, lastElementChild, nextElementSibling, previousElementSibling, childElementCount
  - IE9+



# Node properties

- text, textContent
- link
- vLink
- aLink
- bgColor
- background
- title
- lang
- translate
- dir
- dataset
- hidden
- tabIndex
- accessKey
- draggable
- spellcheck
- contentEditable
- isContentEditable
- offsetParent
- offsetTop
- offsetLeft
- offsetWidth
- offsetHeight
- style
- innerText
- outerText
- webkitdropzone

# querySelector( )

- Uses CSS syntax
  - .aClassName
  - #anIdName
  - [anAttributeName]
  - [anAttributeName=aValue]
- document.**querySelector**('right');
- document.querySelector('.banner');
  - selects first one not all
- CSS2: IE7+  
CSS3: IE9+

# querySelectorAll( )

- `document.querySelectorAll('div');`
- `document.querySelectorAll('.banner');`
- `document.querySelectorAll('[data-ng-controller]')`

# Console shortcuts – jQuery style

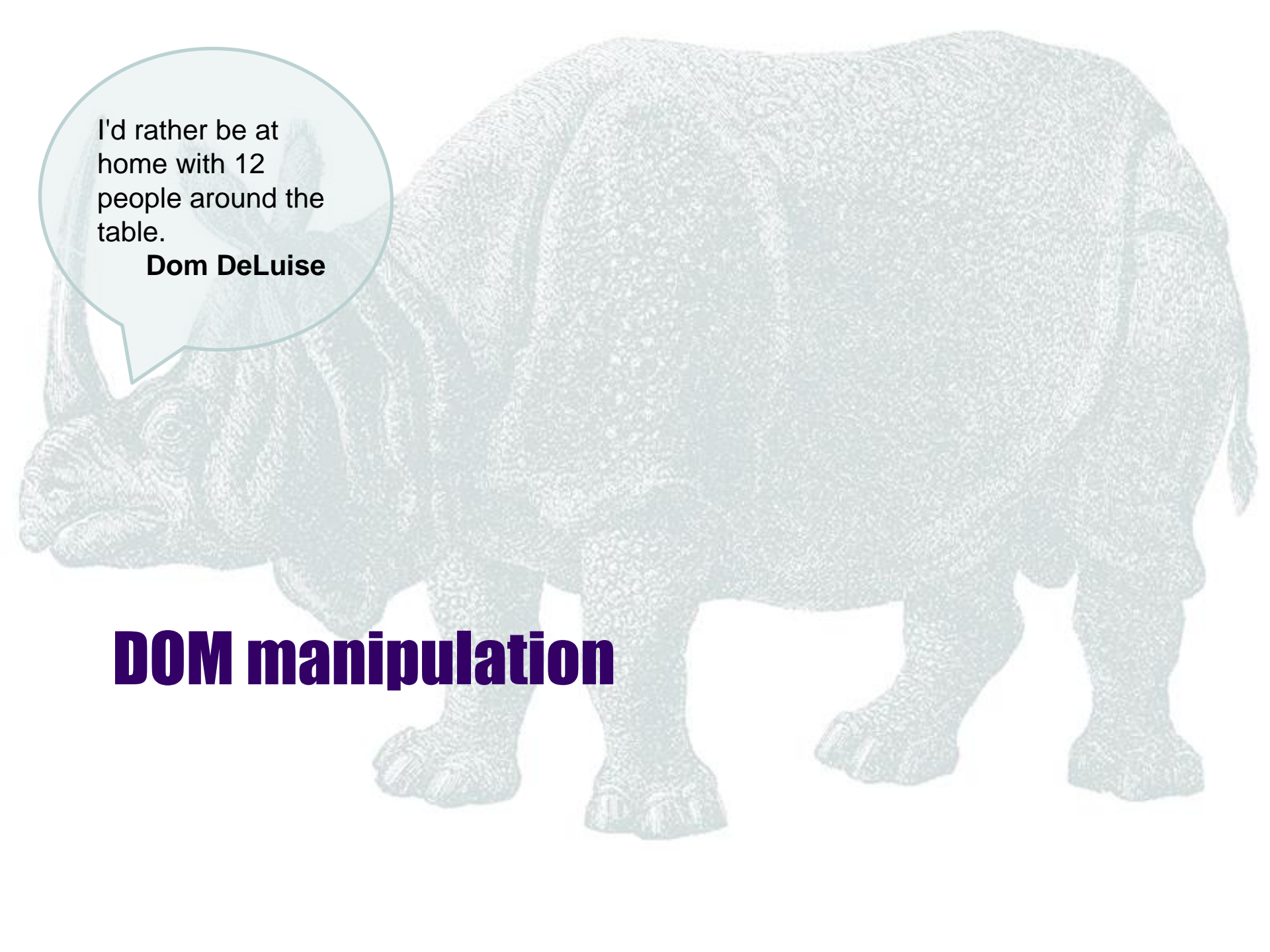
- Chrome, IE11, Firefox 38
  - `$ = document.querySelector`
  - `$$ = document.querySelectorAll`
- `$('#right');`
- `$$('div');`
- `$$('.banner');`

# Other console shortcuts

- `$_` - the result of the last expression
- `$0` - the currently selected DOM node in the elements panel

# Exercise

- on the Centriq.com home page try:
  - `document.getElementsByClassName('menu-item')`
  - `document.querySelector('.menu-item')`
  - `document.querySelectorAll('.menu-item')`
  - `document.querySelectorAll('.menu-item a')`
  - `document.querySelector('#gf_5')`
  - `document.querySelectorAll('h1, h2, h3')`
  - `document.querySelectorAll('input[value][type="checkbox"]:not([value=""]));`



I'd rather be at  
home with 12  
people around the  
table.

**Dom DeLuise**

**DOM manipulation**

# Intro



- DOM manipulation
  - doing any kind of changes with JavaScript to the document such as hiding, showing, animation, altering data.



# Attributes - HTML

- Most HTML attributes are JS properties
  - `image = document.getElementsByTagName('img')[0];`
  - `imageSrc = image.src;`
  - `form = document.forms[0]`
  - `formAction = form.action`
  - `formStyle = form.style // .font, .width, .height, etc.`
- exceptions
  - `class`
    - `form.className`
  - `data prefixed (data-role)`
    - `dataset.role`

# Exercise

- Go to your corporate site
- Select an image
- Change the src attribute to a new image found on Google Images or another site.

# Attributes - HTML

- **element**.attributeName = 'value'
  - **element**.style.propertyName
    - div.style.background = 'darkblue'
    - div.style.backgroundColor = 'darkblue'
- **element**['attributeName'] = 'value'

# Exercise: change style

- Navigate to a page with images.
- Hide all the images with:
  - `for (var i = 0 ; i < document.images.length; i++) {`
  - `document.images[i].style.display = 'none';`
  - `}`
- Put the images back
  - Set display value to empty string or 'block'

# Attributes – class – IE10+

- Add a class
  - `myDiv.classList.add('myCssClass');`
- Remove a class
  - `myDiv.classList.remove('myCssClass');`
- Toggle a class
  - `myDiv.classList.toggle('myCssClass');`

# Attributes - non HTML

- Attributes in HTML, not properties in JS, require special methods
  - `getAttribute( name )`
  - `setAttribute( name, value )`
  - `hasAttribute( name )`
  - `removeAttribute( name )`



# Exercise

- Navigate to a page with images.
- Border all the divs with:
  - `var divs = document.querySelectorAll('div');`
  - `for (var eachKey in divs) {`  
    `divs[eachKey]['style'] = 'border: 5px fuchsia dotted';`  
    `}`

# getComputedStyle( )

- `window.getComputedStyle(element)`
- `window.getComputedStyle(element).propCamelCase`
- `window.getComputedStyle(element).["prop-skewer-case"]`



# Exercise

- `var body = document.querySelector('body');`
- `var currentOpacity =  
parseFloat(window.getComputedStyle(body)["opacity"], 10);`
- `var newOpacity = currentOpacity * .60;`
- `body.style.opacity = newOpacity;`

# Content – get/set

- `.innerHTML` - returns all content
  - `.innerHTML = 'html code'`
- `.textContent` - returns just text
  - `.textContent = 'text'`
  - IE - `innerText`

# Exercise



- Show all the text on a web page

# Delete

- process
  - find the immediate parent (not ancestor)
  - remove the immediate child
- `var body = $('body')`
- `document.querySelector('html').removeChild(body)`
- `badNode.parentNode.removeChild(badNode)`

# Create / clone

- Create a node
  - `document.createElement('tag name')`
- Clone a node
  - `var dupNode = node.cloneNode(deep: Boolean);`
    - `deep` = clone all child nodes & event handlers
  - `var div2 = div1.cloneNode(true);`
  - `div2.id = div1.id + '_clone';`

# Insert

- `nodeVar.appendChild(node)`
- `parentNodeVar.insertBefore(newNode, refNode)`
- `nodeVar.insertAdjacentHTML(position, node)`
  - positions =
    - beforebegin
      - afterbegin
      - beforeend
    - `<div>`
    - `</div>`
    - afterend

# Move an element

- Use selection then an insert command.



# Forms

- get/set values of text box input
  - `document.querySelector('#q').value;`
  - `document.querySelector('#q').value = 'JavaScript';`
- radio buttons
  - `document.querySelector('#radio2').checked`
  - `document.querySelector('#radio2').checked = true;`
- drop downs
  - `document.querySelector('#select-choice option').selected`
  - `document.querySelector('#select-choice option[value="Choice 2"]').selected = true;`



# Forms

- **textareas**
  - `document.querySelector('#textarea').value;`
  - `document.querySelector('#textarea').value = 'This is a text area.';`
- **checkboxes**
  - `document.querySelector('#checkbox').checked;`
  - `document.querySelector('#checkbox').checked = true;`
- See an example at: <forms/form-input-output.html>

# Useful value properties

- **anInputElement.value**
  - the contents of a text box `<input type="text" />`
  - does not work with checkboxes
- **anElement.innerHTML**
  - contents of an element
  - add raw HTML instead of creating/cloning elements

# Packages - forms

- Stretchy – autosizing
  - <http://leaverou.github.io/stretchy/>
- Autocomplete.js
  - <http://www.jacklmoore.com/autocomplete/>
  - textareas only

Input with placeholder:

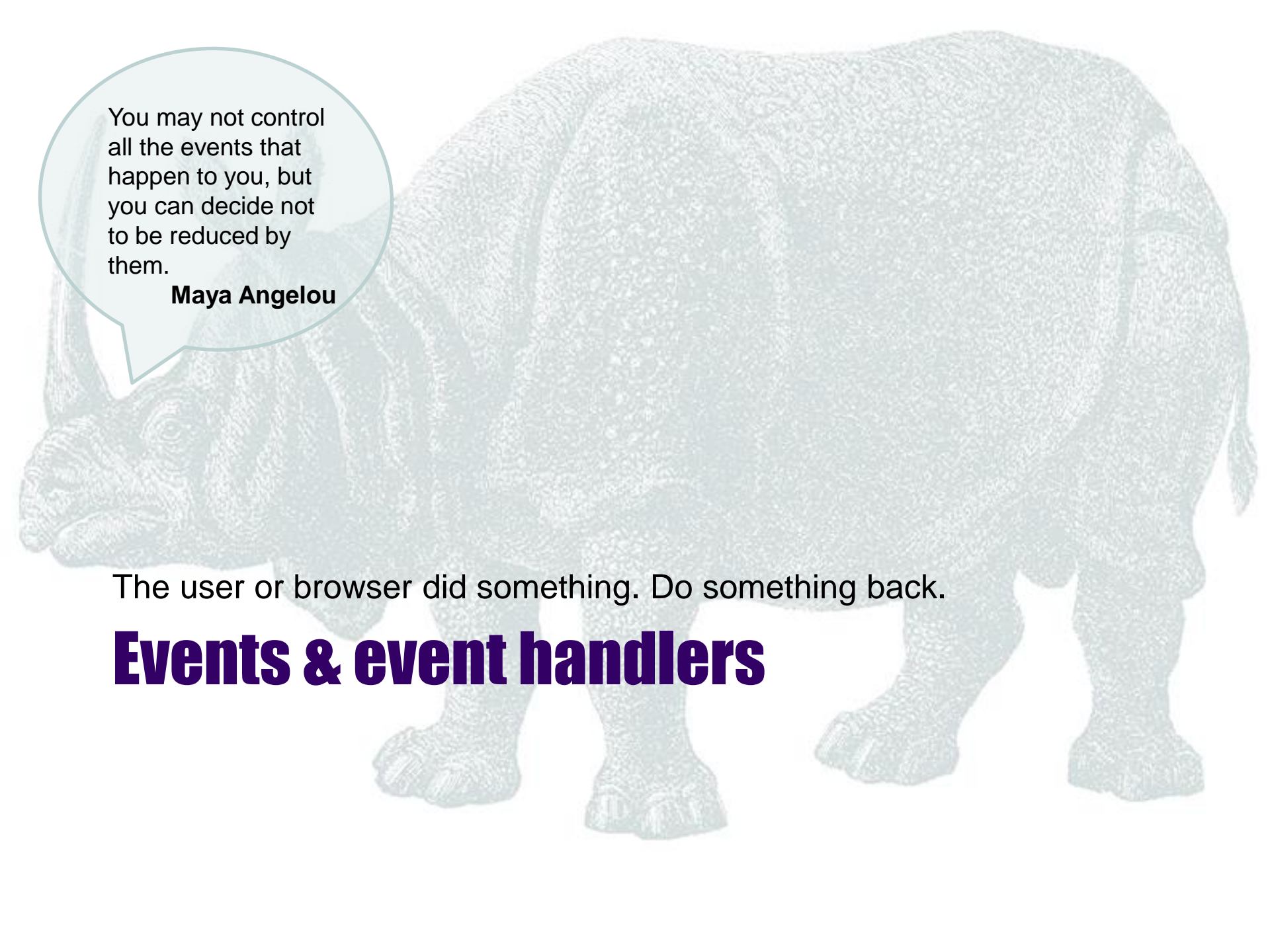
Input with initial value:

Select:

Input with placeholder:

Input with initial value:

Select:



You may not control  
all the events that  
happen to you, but  
you can decide not  
to be reduced by  
them.

**Maya Angelou**

The user or browser did something. Do something back.

## **Events & event handlers**

# Intro



- Browser detected interactions with the GUI that cause it to execute predefined functions if available.
- **Event listener/handler** – your code that gets called when an event is triggered.
- All html elements have attributes for predefined events that can be triggered.
- old style
  - `<input type="button" onclick="doSomething();" ...`

# Node events



- onblur
- onerror
- onfocus
- onload
- onresize
- onscroll
- onbeforeunload
- onhashchange
- onlanguagechange
- onmessage
- onoffline
- ononline
- onpagehide
- onpageshow
- onpopstate
- onstorage
- onunload
- onrejectionhandled
- onunhandledrejection
- onabort
- oncancel
- oncanplay
- oncanplaythrough
- onchange
- onclick
- onclose
- oncontextmenu
- oncuechange
- ondblclick
- ondrag
- ondragend
- ondragenter
- ondragleave
- ondragover
- ondragstart
- ondrop
- ondurationchange
- onemptied
- onended
- oninput
- oninvalid
- onkeydown
- onkeypress
- onkeyup
- onloadeddata
- onloadedmetadata
- onloadstart
- onmousedown
- onmouseenter
- onmouseleave
- onmousemove
- onmouseout
- onmouseover
- onmouseup
- onmousewheel
- onpause
- onplay
- onplaying
- onprogress
- onratechange
- onreset
- onseeked
- onseeking
- onselect
- onshow
- onstalled
- onsubmit
- onsuspend
- ontimeupdate
- ontoggle
- onvolumechange
- onwaiting

# Binding/registering event handlers

- Give your element an ID
  - `<input id="btn_submit" ...`
- Find the element in the html document
  - `var submitButton = document.getElementById("btn_submit");`
  - `var submitButton = $('#btn_submit');`
- Create the event handler
  - `function doWhenSubmitted() { }`
- Register the handler with the element event
  - `submitButton.onclick = doWhenSubmitted;`

# Register event handlers

- Combine them:
  - `submitButton.onclick = function ( ) {`
    - `// code`
  - `};`
- **or**
  - `document.getElementById("btn_submit").onclick = doWhenSubmitted ;`



# Exercise

- Find an element on a page
- Write a function for an event handler
  - `window.location.href = 'new location';`
    - or just `window.location = ...`
- Bind and execute

# Event type overview

- blur / focus
  - validation, put cursor in form field on page load
- click / dblclick
- mousedown / mouseup
- mouseover / mouseout
  - image swaps
- scroll
- use as function names
  - click( ), blur( ), focus( ), etc.

# Exercise

- Add an event handler to Google's logo or doodle to tell you when you move the mouse over it.

# Events in the handler

- JavaScript will pass an event object to your event handler as an argument if you want
- Declare your function with one arg
  - `function doThisOnClick(e) { console.log(e); }`
- Properties
  - type
  - srcElement, srcElement.id
  - x,y,pageX, pageY, screenX, screenY, offsetX, offsetY
  - which (keyboard key)

# Events in the handler – IE8

- `function doOnSomeEvent(e){`
- `e = e || window.event; // IE8`
- `// rest of code...`
- `}`

# Show event target in Chrome

- `e.target`
  - the source of the event, the clicked element, the focused element, the blurred element...
- `console.log(e.target)`
  - highlights element on page when hovering
- `console.dir(e.target)`
  - shows property tree in console for browsing

# Exercise

- create an event handler for a document click event anywhere on the html
  - log the x and y coordinates
  - log the element that was clicked on
- Add the function to the mouseover event
- Click on a button to redirect to Google

# **addEventListener() – IE9+**

- W3C
- `elementVar.onEvent = function`
- `elementVar.addEventListener('event', function, bubble?)`
  - 'event' - one of the event names e.g. click
  - function - the function name or an anonymous function
  - bubble? - a boolean that allows multiple event handlers to 'hear' the event, false to allow bubbling, true to consume event
- `removeEventListener(same as above)`



# Exercise

- Find a search box
- Bind a keypress event handler to show any key (e.which)

# attachEvent( )

- same as addEventListener() but will always bubble
- elementVar.attachEvent('event', function)
- detachEvent('event', function)

# Default action / bubbling / propagation



- e is the event object passed into the event handler
- e.stopPropagation( )
  - <a> will bubble/propagate the click event to its parent
- e.preventDefault ( )
  - <a> will request a page.
- Disable all links on a page, show the one clicked.
  - ```
function disableClicks(e) { e.preventDefault();  
  console.log(e.target || e.srcElement); }
```
  - ```
for(eachLink in document.links) {  
  document.links[eachLink].onclick = disableClicks; }
```

# return false;

- combines
  - e.stopPropagation( )
  - e.preventDefault ( )
- traditional way to do it
- not best practice, but very common
- return false; // no prop, no default

# void(0)

- the void unary operator, requires argument
- the smallest script possible that evaluates as undefined
- used for `<a href="javascript: dosomething(); void(0)">Print</a>` to return to the same page
- otherwise returns a page of the evaluated expression

## Exercise - basic

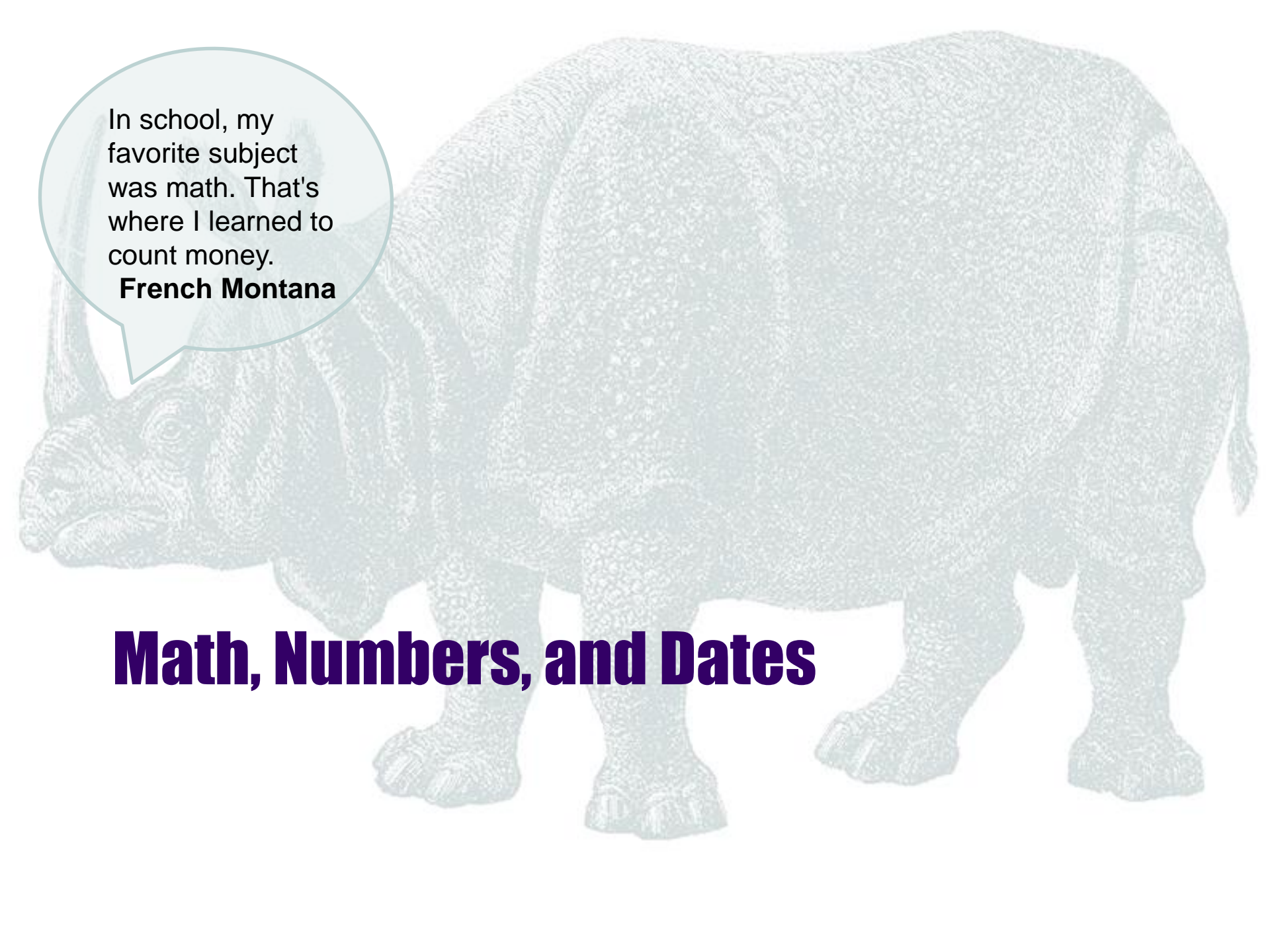
- Use the text on any web page and
  - get the id of a block of text
  - clear the text
  - add new text
  - append text
- What's the difference between `innerHTML()` and `innerText()`?

- [events/append-onclick.html](#)

## Exercise



- Append any element's text to the bottom of the document.



In school, my  
favorite subject  
was math. That's  
where I learned to  
count money.

**French Montana**

# **Math, Numbers, and Dates**



# Randomizing

- `function randomBetween(min, max) {`
  - `min = min || 0;`
  - `max = max || 100;`
  - `return (Math.floor(Math.random() * (max-min) ) + min );`
- `}`
- `randomBetween(0,5);`
- `randomBetween(3,5);`
- `randomBetween(6,5);`

# Random link

- `var urls = new Array("pagereource.com", "javascriptcity.com", "mydemos.com", "yahoo.com", "google.com");`
- `var randomUrl =  
 urls[randomBetween(0,urls.length)];`
- `var link = "<a href='http://www.'+ randomUrl +  
 '>' + randomUrl + '</a>';`

# Number



- To test numbers for validity:
  - `Number.MAX_VALUE`, `Number.MIN_VALUE`
  - `Number.NaN`
  - `Number.NEGATIVE_INFINITY`,  
`Number.POSITIVE_INFINITY`

# Number

- To convert numbers
  - Rounding – `Number.toFixed()`, `Number.toPrecision()`
  - To text – `Number.toExponential()`, `Number.toString()`, `Number.toSource()`

# Strings to numbers

- used to convert text to numbers to use in a calculation
- `parseInt('6') + 3`
- `parseFloat('5.6') / 2`

# Date

- Based on Java's implementation
- `new Date()` is the current time/date
- Try it
  - `new Date()`
  - `new Date().toString()`
  - `new Date().toISOString()`
  - `new Date().toLocaleString()`
  - `new Date().toLocaleDateString()`
  - `new Date().toLocaleTimeString()`

# Packages


- Date-fns
  - <https://date-fns.org/>
- DateJS
  - <http://www.datejs.com/>



# Exercise

- `var startOfToday = dateFns.startOfToday();`
- `var thisYear = dateFns.getYear(new Date());`
- `var nextChristmas = new Date(thisYear, 11, 25);`
- `var timeUntilChristmas =  
dateFns.distanceInWordsToNow(nextChristmas)  
;`





“I excel at pulling  
strings!” said  
Arachne. “I’m a  
spider!”  
- Rick Riordan

**Strings**

# Intro

- String objects: `var s = new String("string");`
- String literals: `var s = "string";`
- Duplicate values of string literals are `==` to each other.
  - values are being compared.
- Duplicate values contained in different string objects are NOT `==` to each other.
  - objects are being compared.

# Manipulation

- `charAt()` / `charCodeAt()` - Returns character / Unicode at position #
- `concat()` - Joins two or more strings
- `fromCharCode()` - Converts Unicode values to characters
- `indexOf()` / `lastIndexOf` - Returns the position of the first/last found occurrence of a specified value
- `match()` / `search()` – see regular expressions
  - <http://regexlib.com>

# Manipulation

- `replace()` – regular expression driven
- `slice()` / `substr()` or `substring()` - Extracts a part / well defined part of a string
- `split()` / `join()` – Splits/combines a string into an array of substrings
- `toLowerCase()` / `toUpperCase()`
  - Converts to lowercase/uppercase letters
- `valueOf()` - Returns the numeric value of a String object

# Libraries

- StringJS
  - <http://stringjs.com/>





**jQuery**

# Motives for jQuery

- create standard cross-browser JavaScript API
- simple selector syntax
- AJAX was simpler - promises

# Setup

- Use a content delivery network
  - Use <https://cdnjs.com/>
  - Use jQuery.com, google.com, etc.
- Load at top of page
- Follow JavaScript rules of where to put code
  - DOM manipulation at bottom
  - DOM manipulation at top with defer or other delay
  - Other functions optionally at top



# jQuery and \$

- Use jQuery when it's necessary to understand the difference between jQuery and anything else using a \$ variable.
  - \$('selector')
  - jQuery('selector')
  - document.querySelectorAll('selector')

# the jQuery datatype

- an array-like object
  - looks like an array
  - uses all array functions
- all jQuery functions return a jQuery object type
- all jQuery functions require an argument / target object of type jQuery

# Creation / selection

- `var $newParagraph = $('<p/>')`
- `var $allMyPChildren = $('body p')`
- `var $boldness = $('.bold')`
- `var $theOneAndOnly = $('#unique')`

# Conversion

- Common JavaScript objects
  - \$(document)
  - \$(navigator)
  - \$(this) or \$(self)
  - \$(window)
  - \$(chrome)
  - \$(history)



**Libraries**

# General Utilities

- Underscore - <http://underscorejs.org/> 4kb
  - 60+ functions
- Lo-Dash - <http://lodash.com/>

# UX



- Autocomplete
  - <http://bevacqua.github.io/horsey/>

# Communciation

- Node
- Socket IO - <http://socket.io/>
- AJAX
  - IE11 does not use promises - <http://caniuse.com/#search=promise>
  - Axios - <https://github.com/mzabriskie/axios>
  - SuperAgent - <https://visionmedia.github.io/superagent/>
  - SuperAgent-promise - <https://github.com/lightsofapollo/superagent-promise>



# Charting

- Google Chart API
  - <https://developers.google.com/chart/>
- \*Chartist - SVG
  - <http://gionkunz.github.io/chartist-js/>
- \*D3 - complex SVG
  - Britecharts - <http://eventbrite.github.io/britecharts/>
- Chart.js - canvas
  - <http://www.chartjs.org/>

# Charting

- [libraries/chartist.html](#)
- [libraries/chartjs.html](#)



# SVG generation

- Used mostly for advanced charts
- d3.js - best tool for advanced charts
  - <http://d3js.org/>
  - <http://vimeo.com/45558674>
- d3 addons
  - Cubism - time series visualization
    - <http://square.github.com/cubism/>

# Diagramming



- GoJS - interactive diagrams
  - <http://www.gojs.net/>

# Mapping

- Google Maps API
  - <https://developers.google.com/maps/documentation/javascript/>
- Leaflet
  - mobile-friendly interactive maps, Tech. Radar Assess
  - <http://leafletjs.com/>
- OpenStreetMap
  - <http://leafletjs.com/>

# Color



- Color Thief - extract main colors from image  
<http://lokeshdhakar.com/projects/color-thief/>

# UX



- Carousel

- [http://codecanyon.net/item/responsive-bootstrap-carousel/full\\_screen\\_preview/13112740](http://codecanyon.net/item/responsive-bootstrap-carousel/full_screen_preview/13112740) \$12

# Animation



- Greensock TweenMax / TimelineMax
  - <http://greensock.com/products/>



# Misc

- Natural language dates
  - Date for humans - <http://matthewmueller.github.io/date/>
- Tree structures
  - <http://jnuno.com/tree-model-js>
- Directory
  - <http://www.jsdb.io/> - some
- <canvas>
  - Fabric.js - <http://fabricjs.com/>

# Misc

- Drop.js - absolute positioning
  - <http://github.hubspot.com/tether/>
- Email
  - <http://emailjs.org/>
- Excel functions
  - <http://www.stoic.com/formula>
- Search, sort, filter
  - <http://listjs.com/>
- Google Closure Tools
  - <https://developers.google.com/closure/>

# Misc



- Text editor
  - Quill - <http://quilljs.com/>
- Clipboard copy
  - <http://zenorocha.github.io/clipboard.js/>

# Videos

- Douglas Crockford: The JavaScript Programming Language
  - Aug 25, 2011
  - <http://youtu.be/v2ifWcnQs6M>