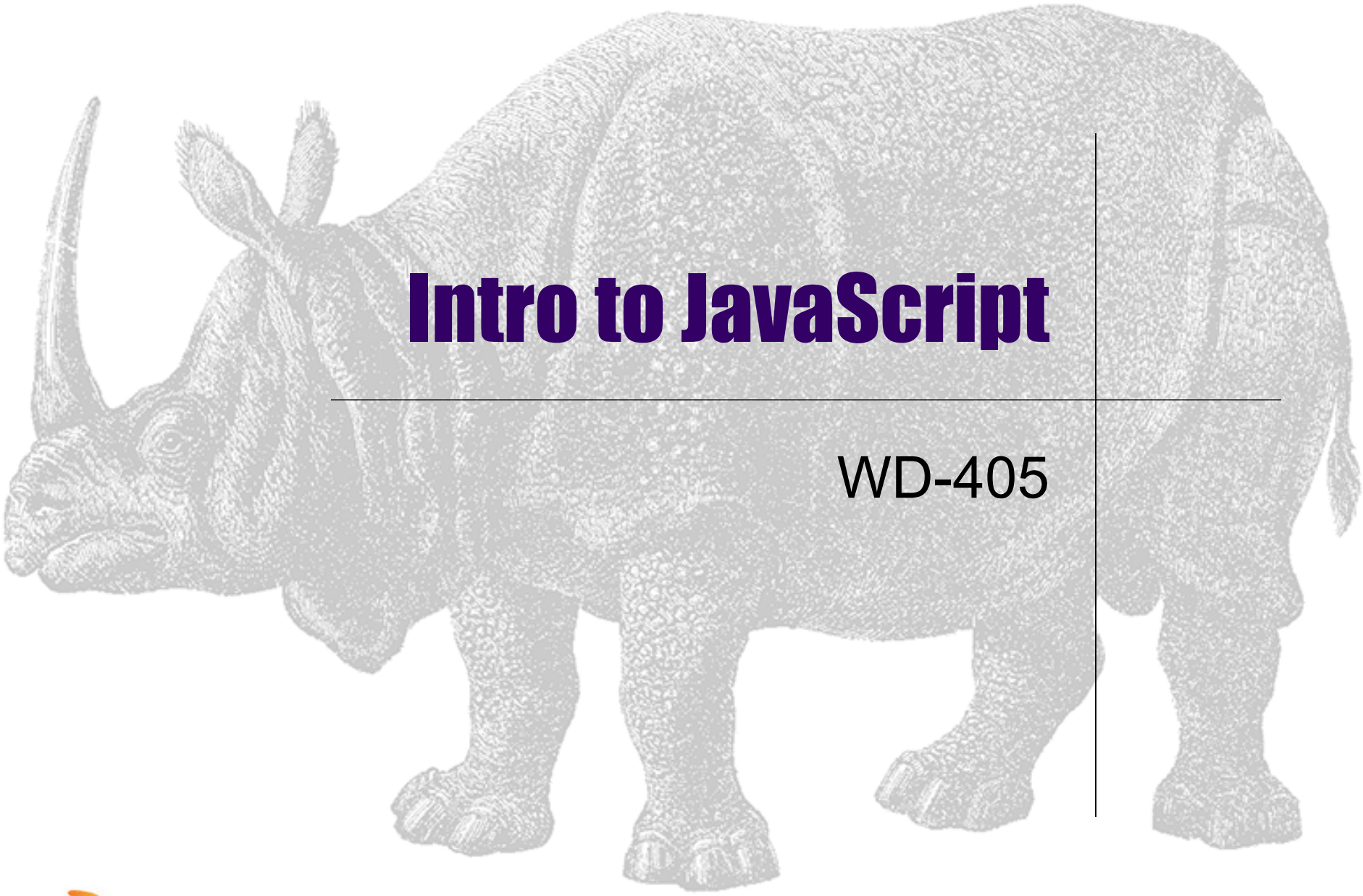


Intro to JavaScript

WD-405

CENTRIQ TRAINING

Prerequisites

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

JavaScript related courses

- **Programming**
  - WD-405 JavaScript (for non-programmers)
- **Web**
  - WD-500 HTML5 and CSS3 for Web Designers
  - WD-505 Intro to jQuery

JavaScript related courses

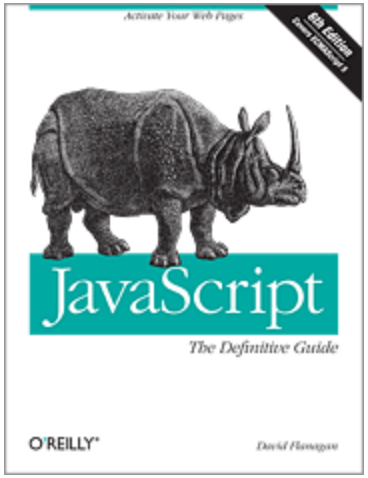
- **Web**
  - WD-530 Angular with TypeScript

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

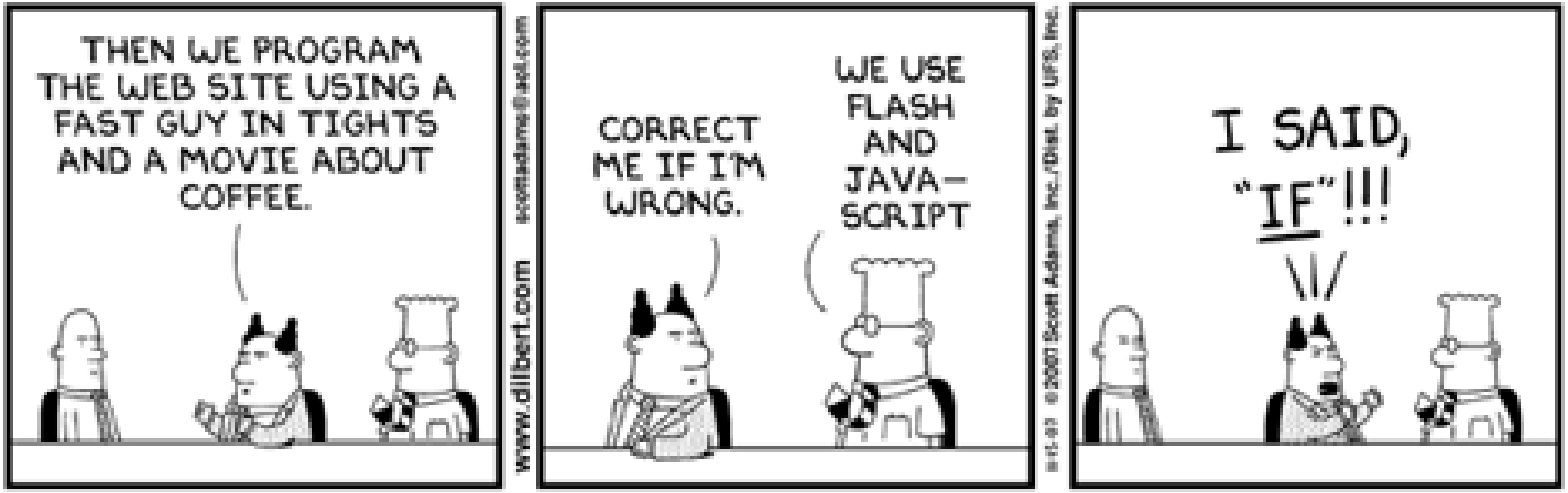
- **Jump Start JavaScript,** Ara Pehlivanian; Don Nguyen, SitePoint, July 12, 2013
- **JavaScript Step by Step**, 3rd Edition Steve Suehring, Microsoft Press, June 6, 2013
- **JavaScript: The Good Parts**, Douglas Crockford, O'Reilly Media, Inc. May 8, 2008

Videos

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

Repo

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



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)

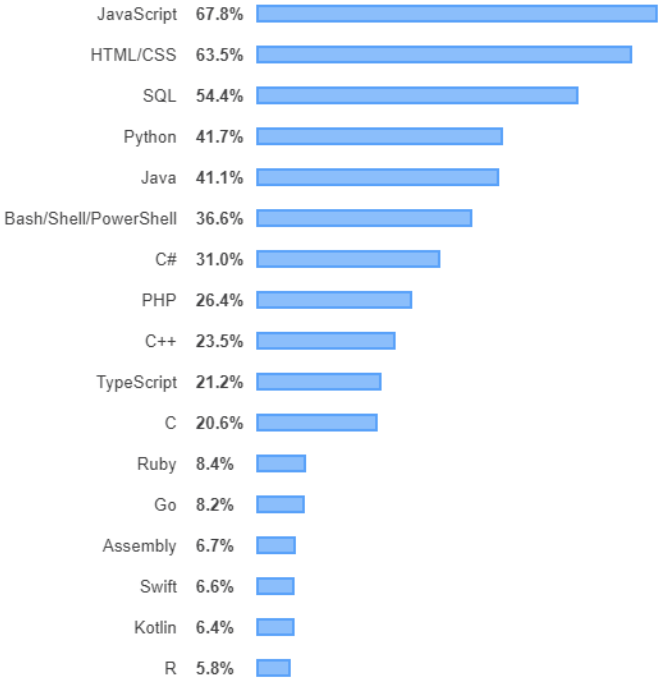
- June 2015 complete
  - Object APIs, **classes**, modules, concise methods, arrow functions, rest and spread, property initializers, proxy/reflect, collections
  - Code-named "Harmony"
  - Not completely supported in IE11 or iOS Safari 8

Popularity

Most Popular Technologies

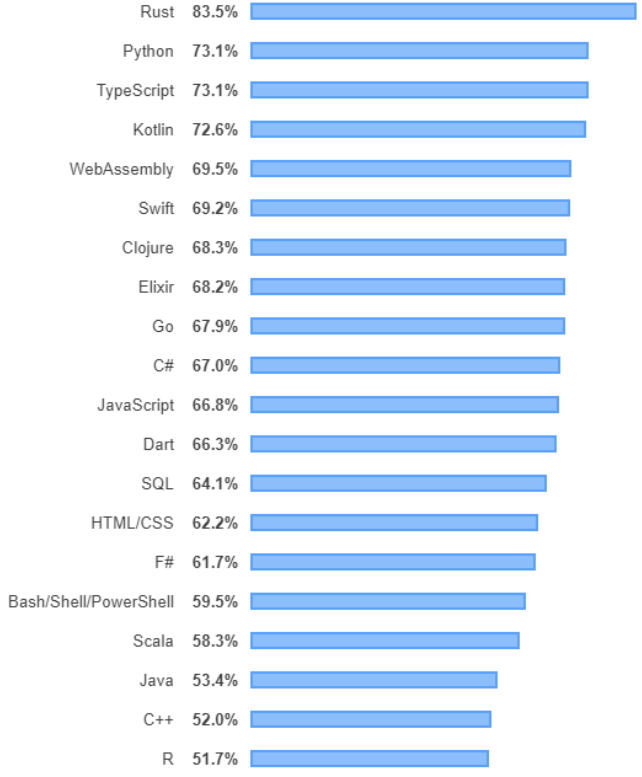
Programming, Scripting, and Markup Languages

All Respondents | Professional Developers



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

CDNs

- CDNJS - https://cdnjs.com/
- jsDelivr - https://www.jsdelivr.com/

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.
- Asynchronous

Development process choices

- 1. Run script code in the address bar of a browser.
  - Very old school
  - Used by hackers
    - Click me!
- javascript: document.write('This text was written by JavaScript'); console.log('from the address bar');
  - Chrome will not paste in the javascript: part

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.log('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.
- 6. Run the external file with node.js

Development process choices

- Use Visual Studio Code with plug-in Code Runner

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>`

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="scripts/file.js“ /> ??
  - <script type="text/javascript" src="http://cdnjs.cloudflare.com/ajax/libs/dojo/1.8.1/dojo.js"></script>
- Using <script>/*<![CDATA[*/ … /*]]>*/</script>
  - Only necessary for embedded XHTML validation
  - Trend is HTML5 not XHTML.

<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"

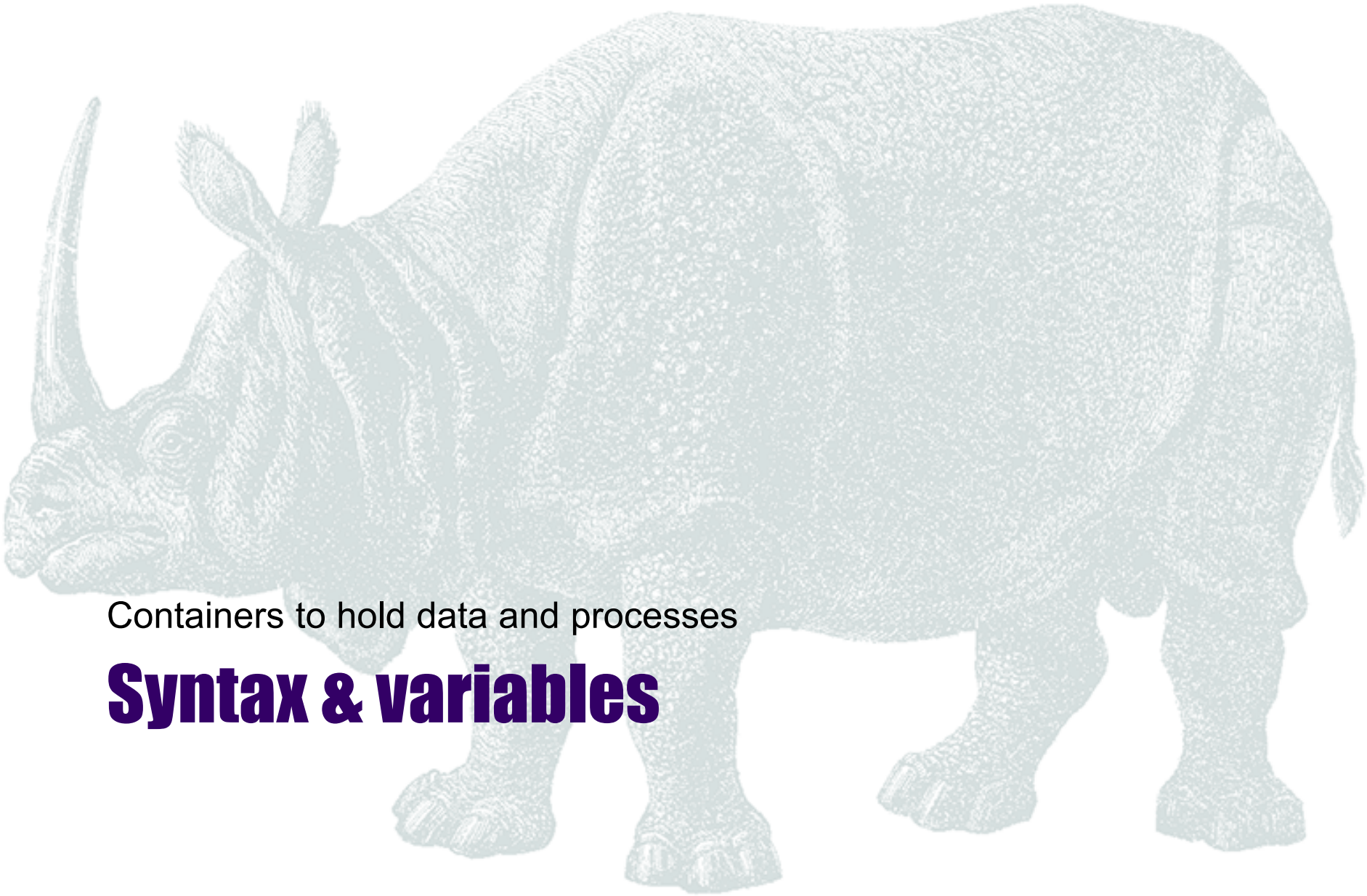
- where
  - place as first line outside a function for checking on the external file
  - place inside function to check just that function
- what it checks
  - duplicate object keys
  - variables without var
  - duplicate arguments
  - freezes `arguments` in a function

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
  - */
- VS Code – Control + /
  - Comment / uncomment hot key



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 (function) 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;
    - { let 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

let and const – ES6

- **let** and **const** have block scope
  - value of a variable that is not initialized is undefined
- variable declared with **const** cannot be reassigned
  - no new reference for whole
  - parts can be mutable

Global declaration

- initalize 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);

- Refresh and run this
- **let** i = 5;
  { **let** k = 5; }
  console.log(i + k);

Statement types

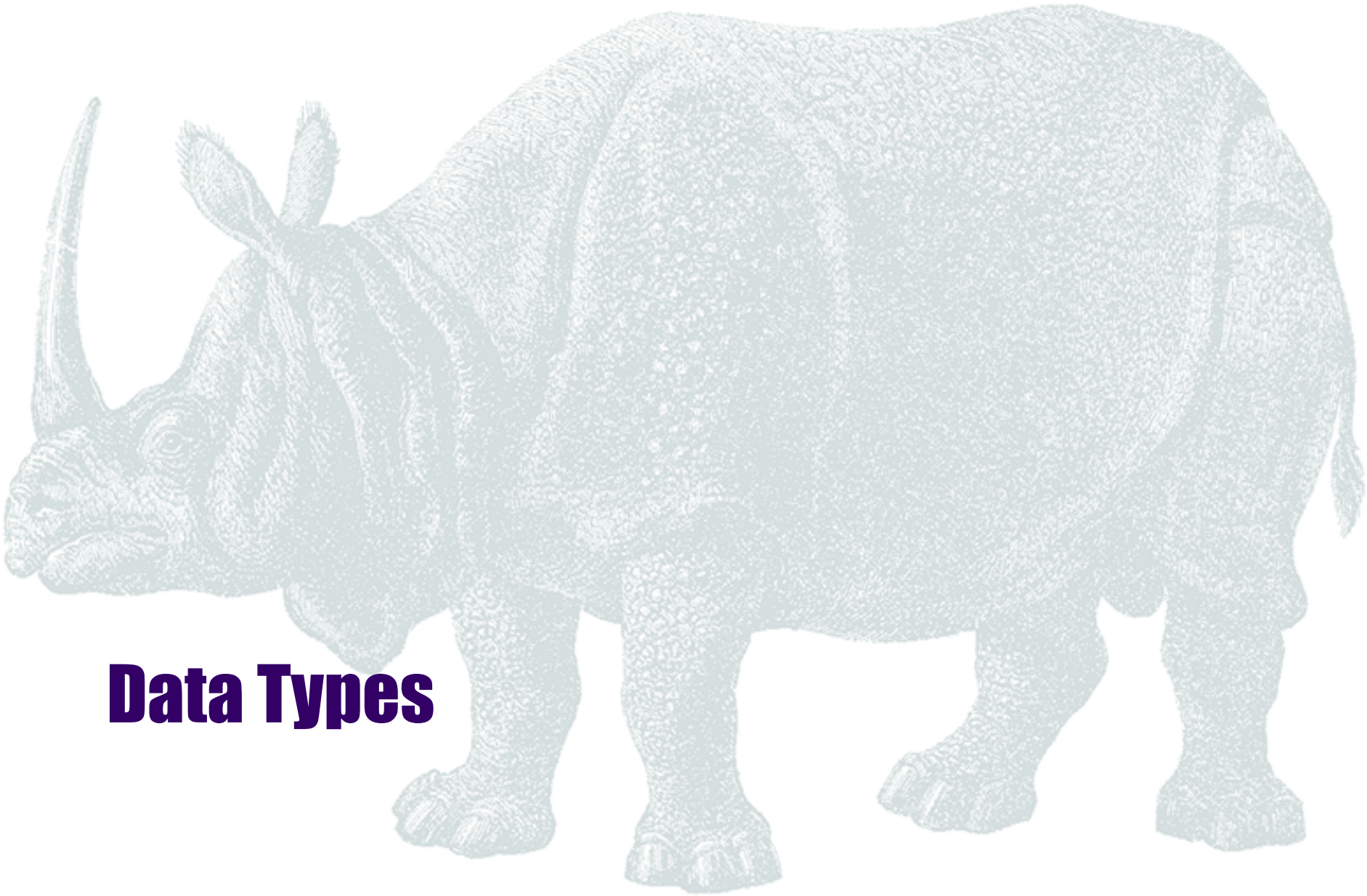
- Value assignment
  - direction is right to left
  - x = 5;
  - x = y; (x ← y)
- Execute a function
  - console.log ("Hello, world!");

Variables get copied values

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

Exercise

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



Data Types

Data types - primitives

- Number (integers, floating point, scientific notation)
- String (text)
- Boolean (true, false)

- Undefined
- Null

Numbers

- Has ability to run special functions

```
(123).toString();  //"123"
(1.23).toFixed(1); //"1.2"
```

Number conversion from text

- Number is a library of functions.

```
Number.parseInt("1")          //1
Number.parseInt("text")       //NaN
Number.parseFloat("1.234")    //1.234
Number("1")                   //1
Number("1.234")               //1.234
```

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 an object value

NaN

- Non-primitive 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?

Strings (text)

- defined with ' ' or " "
- Single quotes are a little better to read than doubles, can be confusing when used with other languages.
- ES6 – `backticks` allow non-printable characters

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
  - ' \' ' , 'It\'s here!' – single quote
  - "It's \"my life\"" – double quote
  - "\t" , "One\ttwo\tthree" – tab
  - "First line\nSecond line" – new line
  - "First line\n
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

- let isHappy = true;
- let isSad = false;
- let hasHappiness = 1;
  - any non-zero number is coerced to a true value when needed
- let 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 falsey values, if(<valueBelow>)
    - false
    - 0 (zero)
    - "" (empty string)
    - null
    - undefined
    - NaN
  - test with - <value> ? true result : false result

Variables in scripts

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

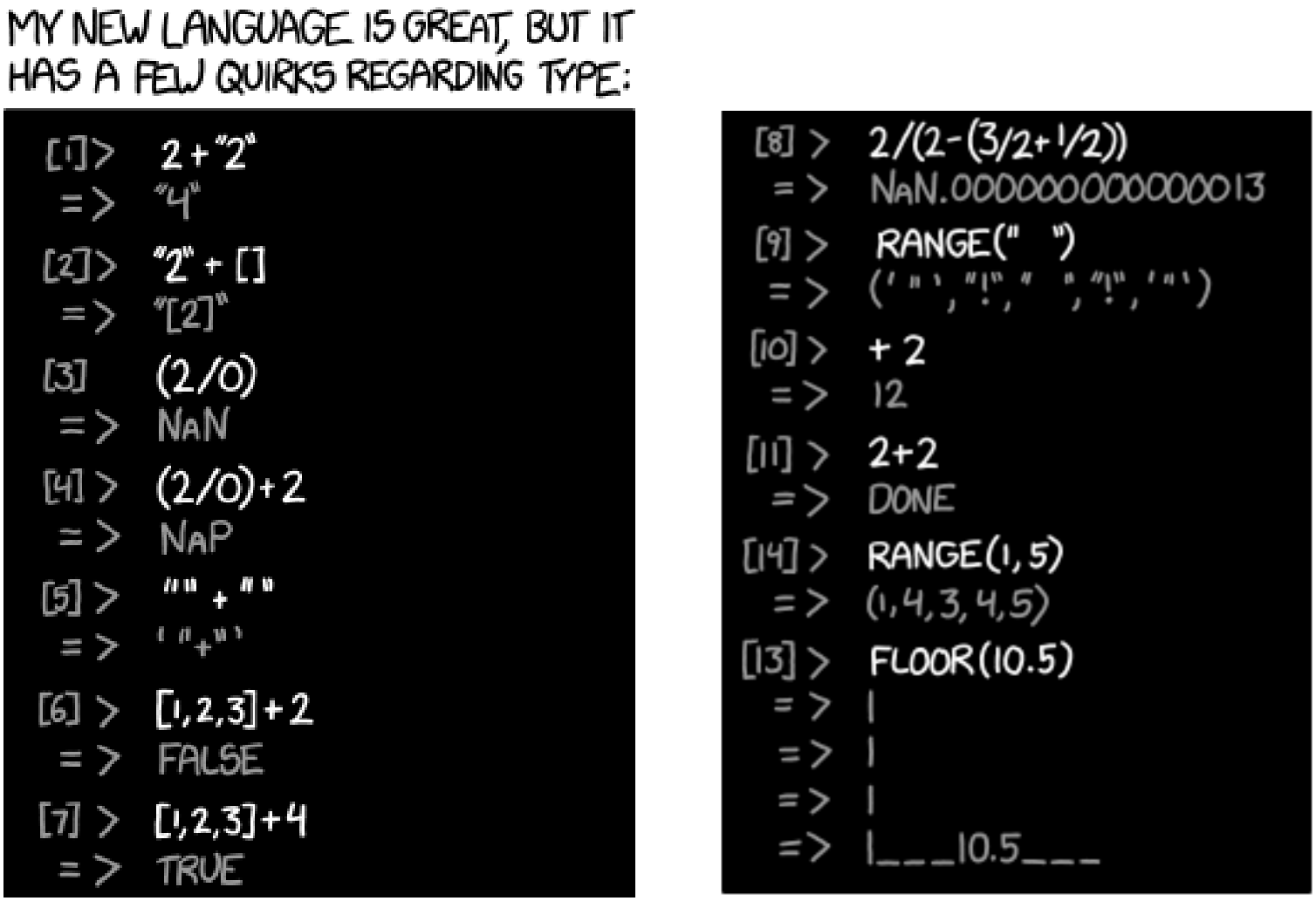
Exercise

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

Exercise

- let dollar = 1.00;
- let dime = dollar/10;
- console.log(dime + dime + dime + dime + dime + dime + dime + dime + dime + dime);

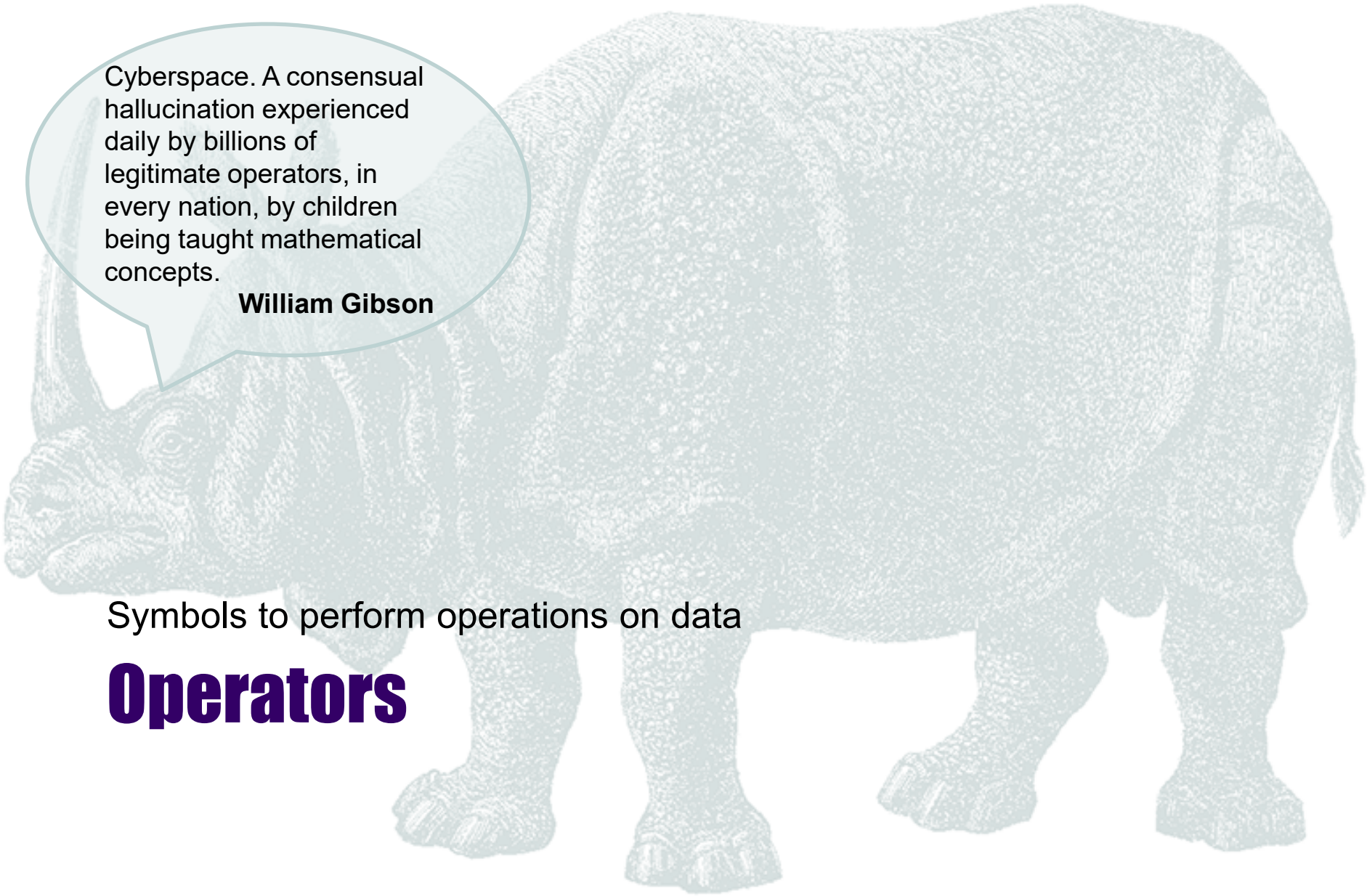
xkcd



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

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



Symbols to perform operations on data

Operators

Arithmetic

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

- 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

Logical operators - dual usage

- || = logical OR, the default operator
  - Allows assignment of value if other value is falsey
- && = logical AND, the guard operator
  - Prevents function from executing if argument doesn't exist

Exercise - || as default operator

- used for function args not passed in
- `function nameIt(myName) {`
  - `var myName = myName || 'No name';`
  - `console.log(myName);`
- `}`
- `nameIt( );`

Converting values

- truncated
- let float = 1.11111;
- let int = float | 0;
- int
- let roundedInt = (float + .5) | 0
- roundedInt

- int = +’123’;
- string = ‘’ + 123;

Compound operators

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

If-else replacement (ternary operator)

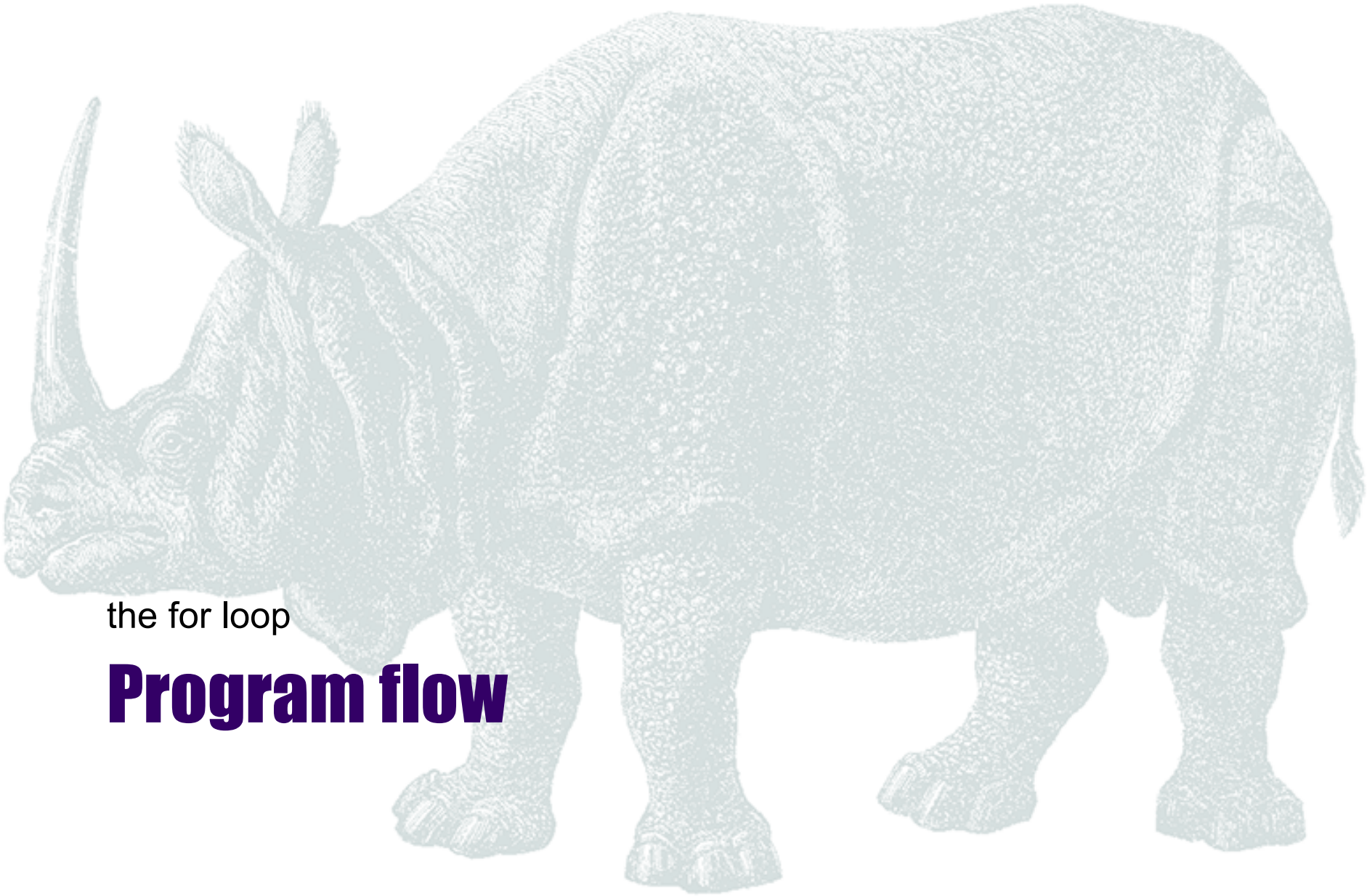
- (expression) ? value if true : value if false
- (“text”) ? “truthy” : “falsey”
- var items = 0;
- console.log(items + " item" +
  ((items===1) ? "" : "s")); // handle plural
- Results
  - 0 items
  - 1 item
  - 2 items

typeof

- Type returns as a string
  - typeof(5)
  - typeof(true)
  - typeof('text')
  - typeof([ ])
  - typeof({ })
  - typeof(function( ) { })
- Useful more for packages/libraries
- Check for function argument type

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

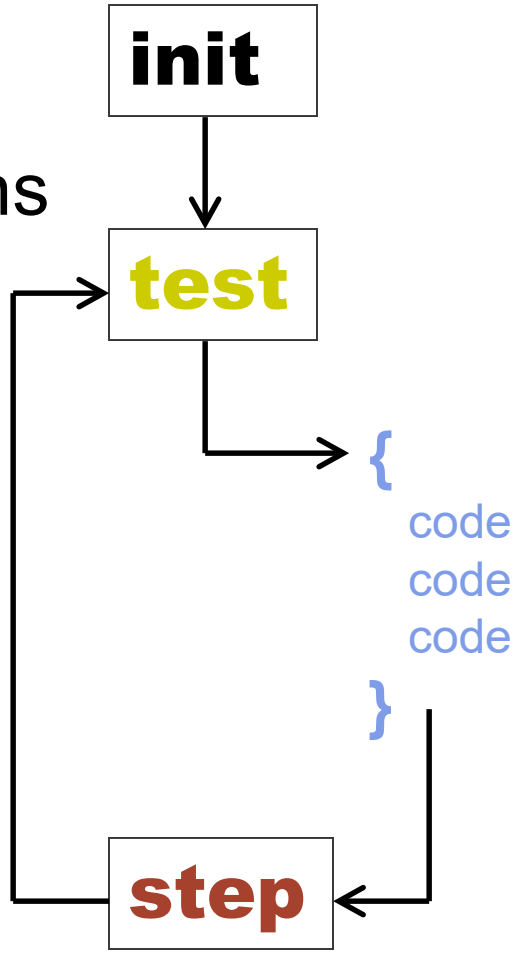
- Branching
  - Check the state of a rule and execute code based on results
- Iteration
  - Execute a unit of code multiple times all at once until a rule is met.

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
      - Can be improved with 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:
  - let i = 0
  - let i = 0, j = 1

for syntax - test


```
for( init ; test ; step )
```

- Execute code **while** test is true
- occurs before each iteration
- false result goes to end of code block
- Examples:
  - i <10
  - i < array.length

for syntax - step

- occurs at the end of each iteration
- Examples:
  - ++i
  - 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

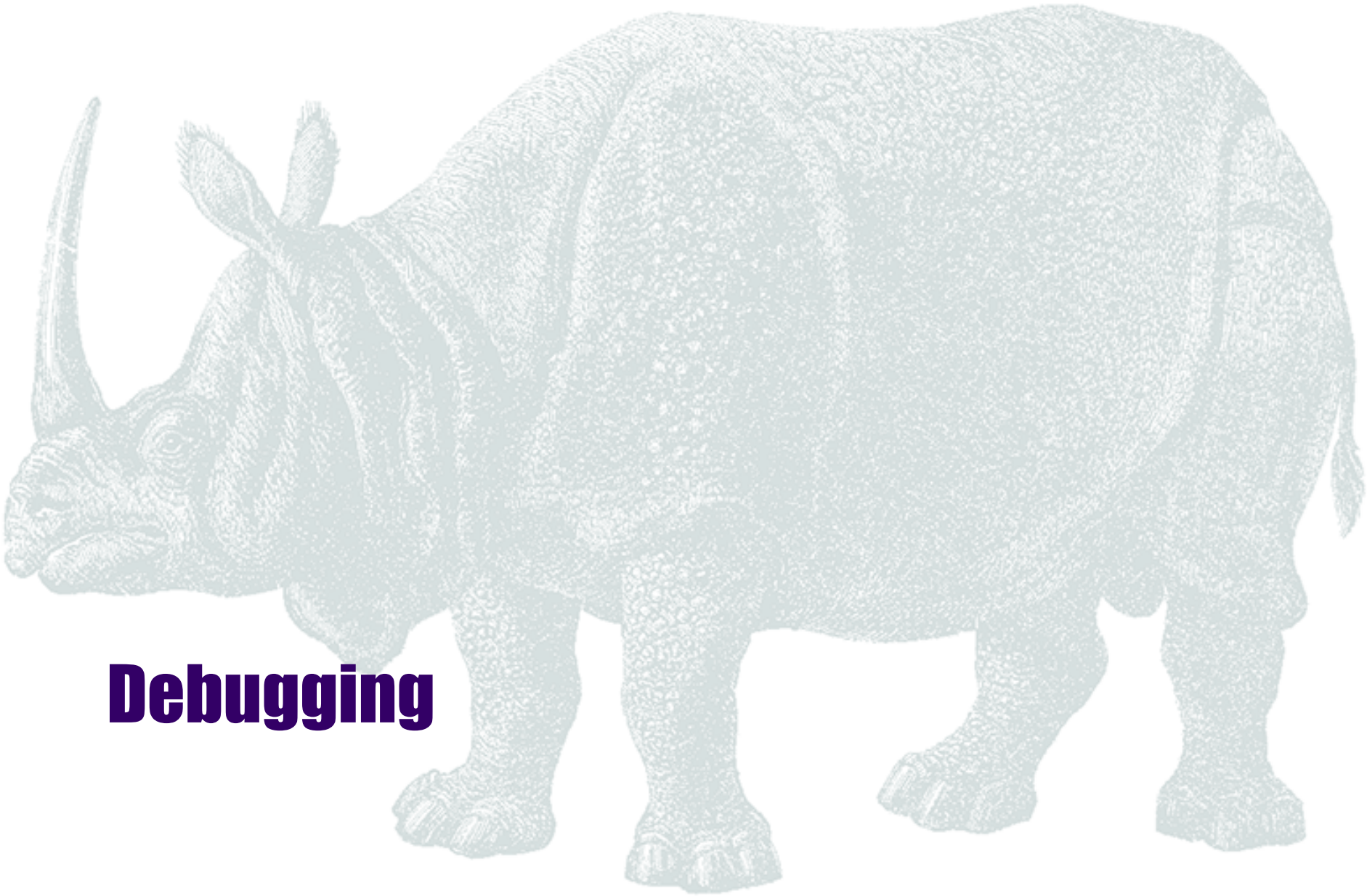
Exercise

- for (var i = 0; i < 10 ; i++) {
- 	console.log('i = ', i);
- }
- console.log(i) // only if you use var, not let

- 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
- Standard console messages to output while running
  - **console.log()**
  - **console.warn( )**
  - **console.error( )**

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

- console.log(arg, arg, …)
- console.log(history, window)
- Not [object History][object Window]

```
▶ Window {external: Object, chrome: Object, document: document, google: Object, _: Object…}
```

console.dir( )

- Use when the browser shows the DOM instead of object properties.
- `console.log(document)`
  - shows rendered DOM
  - for HTML debugging
- `console.dir(document)`
  - shows DOM properties
  - for JS debugging

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);`

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

Chrome Dev Tools

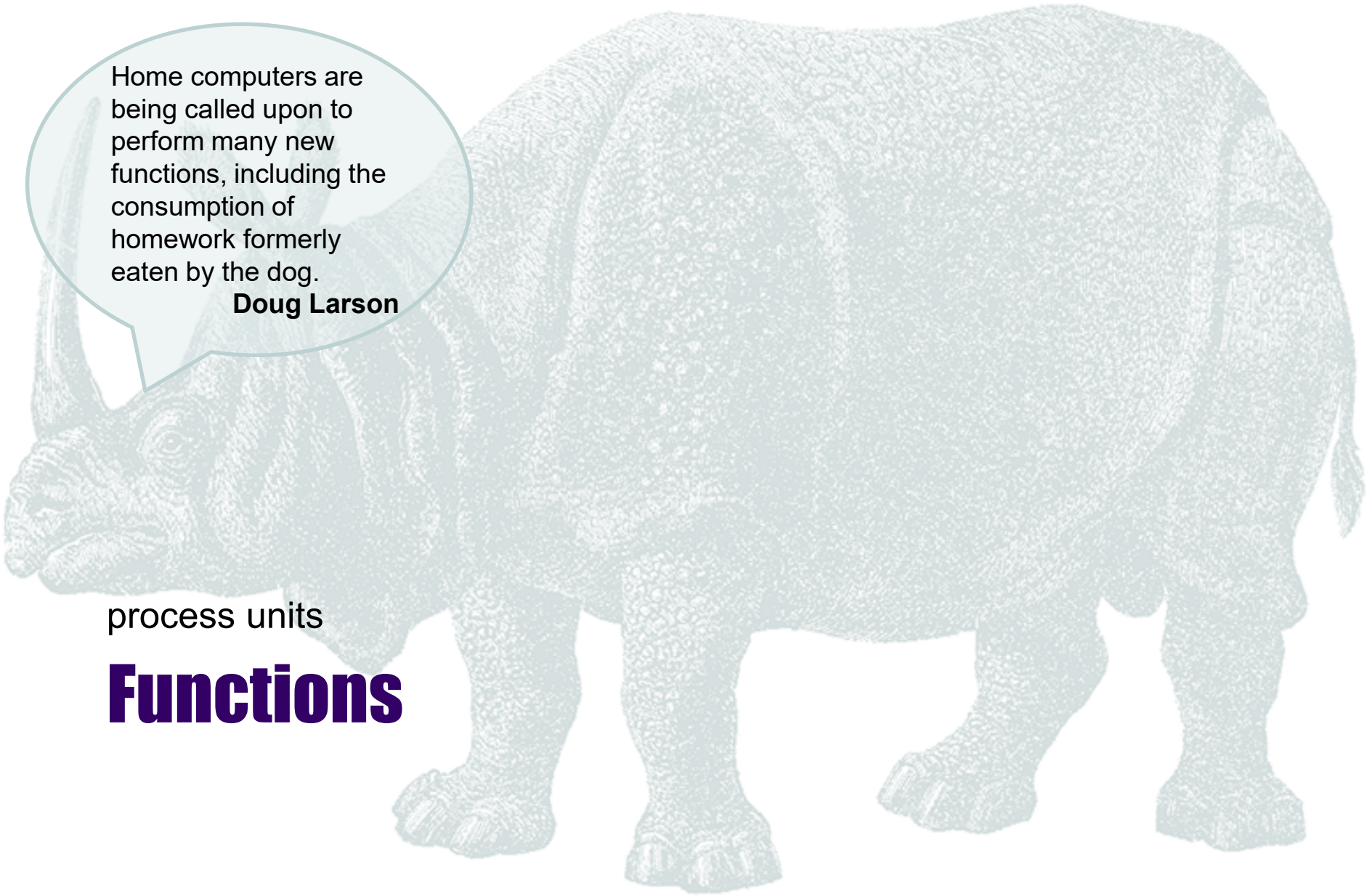
- Node 6.3 (May 2016)
- Debug
  - node --inspect index.js
  - node --inspect --debug-brk index.js (stop on 1st line)
- URL is returned to paste into Chrome
  - right-click and select Mark, click and shift-click first to last column on lines, then just right-click.
  - paste into browser, remove the space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

Visual Studio Code debugging

- // @ts-check – at the top of the file
- Debug mode for node apps
- Set up the Debugger for Chrome extension
  - use server extension or debug config

```
var myNumber = 42;
myNumber = "Some non-numeric text";
```

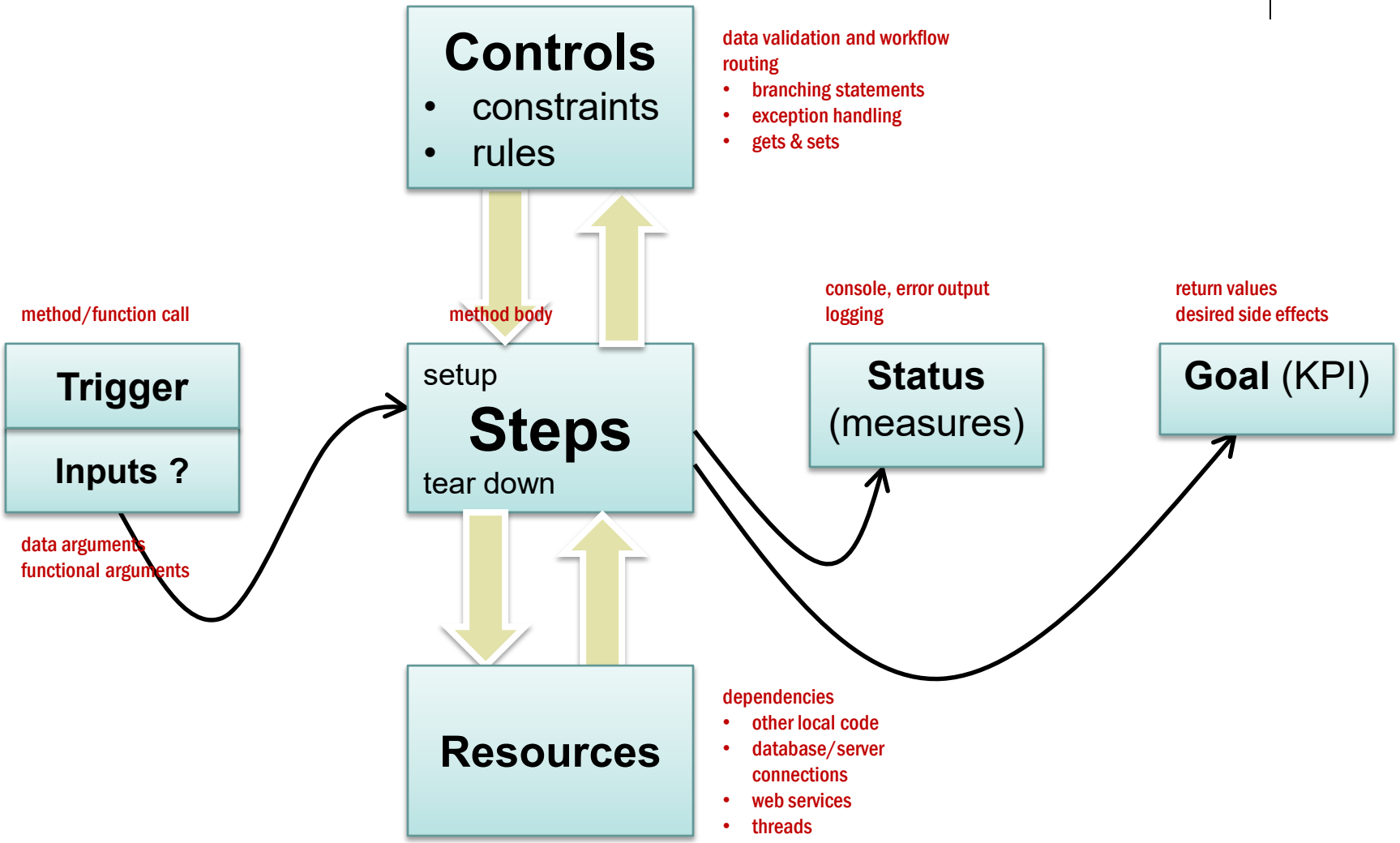


process units

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 – function declaration

- Declaration
- { code block for statements; }
- White space is not important so structure how you like.

```
function doSomething( ) {  }
function doSomething( parameterVar ) {  }
function doSomething( parameterVar,
parameterVar) {  }
```

Naming

- Use variable naming rules.
- Be descriptive.
- 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
- Scope
  - variable is dead after code block completes
- Arguments are optional to parameters

```
function doSomething( howManyTimes ) {   }

doSomething(5);
doSomething( );
```

Calling functions

- Some function calls become values / return a value
  - console.log(returnSomething( ));
- Functions can have no return value
  - doSomething( );
  - doSomething(5);
- Function code body executes and then script continues from the call.

Calling functions

- Functions can have a return value
  - return is last statement in code block.
  - function call becomes the value returned

```
function doSomething( ) { return 5; }

var number = 3 + 4 + 5 ;
var number = 3 + 4 + doSomething( ) ;
```

return

- function with no return statement returns undefined
- value returned from function is the value of the call.

functions/doubleMe.js

Exercise

```
function doubleMe(aNumber) {
 return aNumber*2;
}
doubleMe;   // show the function
doubleMe(); // call the function
doubleMe(3);
doubleMe(3,4);

var f1 = doubleMe;// make a copy named f1
f1(3);            // call the copy
```

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

```
// Multiple copies of functions
function doFunction1( ) {
     return "function1";
}
doFunction1( );
let 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.

Syntax – function expressions

- function is not the first keyword on the line

```
let doSomething = function() {}
let doSomething = function dumbName() {}
```

Syntax – arrow function

- Sugar syntax for anonymous function
- no **this** or arguments

```
let doSomething = function() {}
        let doSomething = () => {};
```

Function calling

- Invoking
  - function, method, constructor or apply forms
- Call with any number of arguments
  - not tied to declaration

```
doSomething(arguments)
theObject.doSomething(arguments)
new Constructor(arguments)
```

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 contains several reference values

```
Math.sqrt(4)
Math.random()
Math.pow(2,8)
Math.round(2.4)
Math.abs(-2.7)

Math.E
Math.PI
```

Exercises

- Write and call a function that will
  - Calculate the area of a circle (pi * r2)
    - in: radius
    - out: area of circle
  - Calculate the volume of a sphere (4/3 * pi * r3)
    - in: radius
    - out: volume of sphere
- The Math object

```
Math.pow(2,8)
Math.PI
```

Exercise - && as guard

- guards against functions running without data

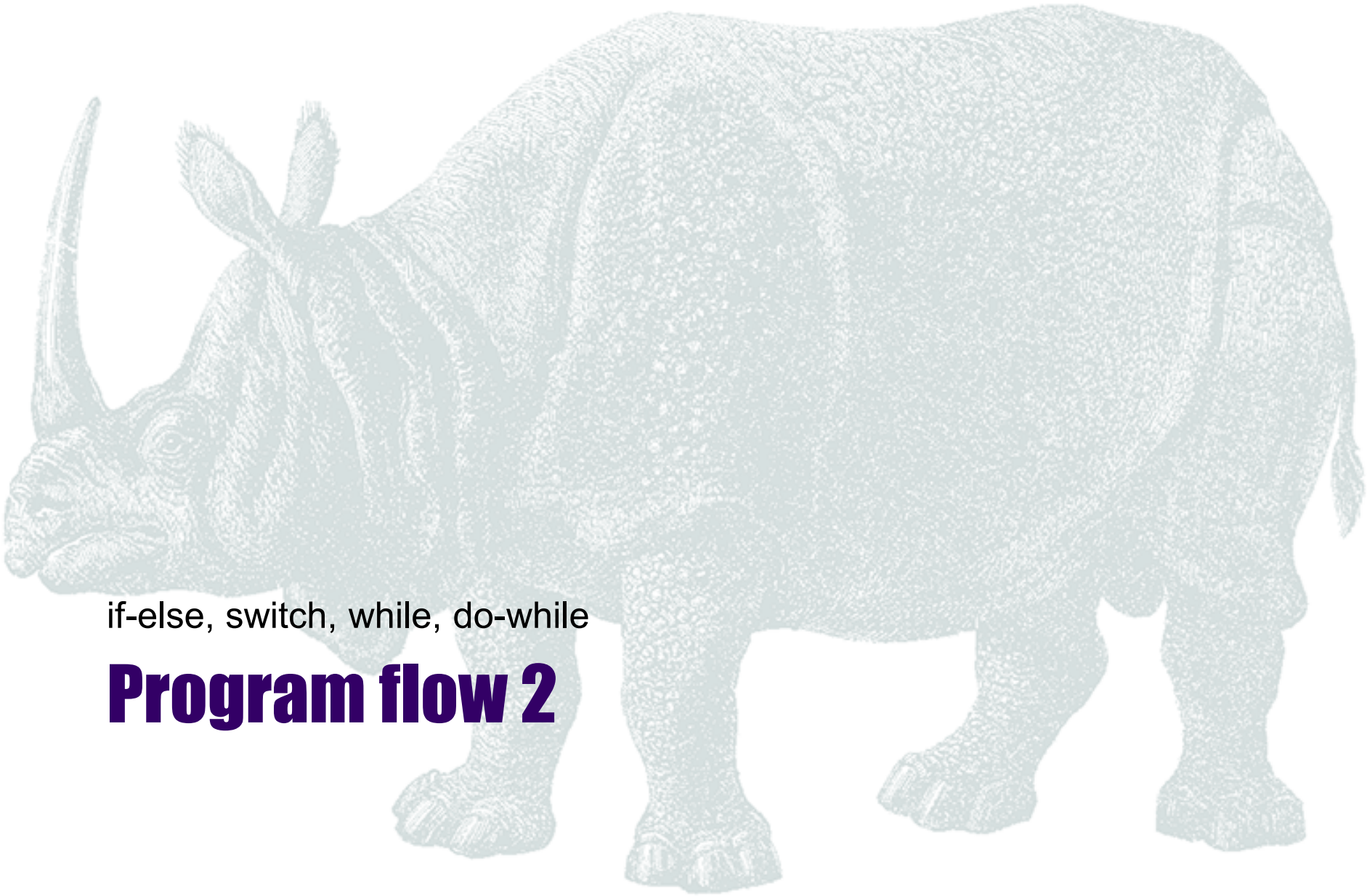
```
isTotallyTrue = true;
if (isTotallyTrue) { console.log('dude!');
}  // boring
isTotallyTrue && console.log('duuuude!');
// better
shouldNotBeFalse = false
!shouldNotBeFalse && console.log('oh no,
duuuude!');
console.log && console.log("This function
exists!");
```

Higher order function

- a function can take a function as an argument
- check to make sure the function is loaded
- execute the function with parens
- use the function with a function argument

```
function doFunction(f) {
      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!
  - x === 3

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: …
```

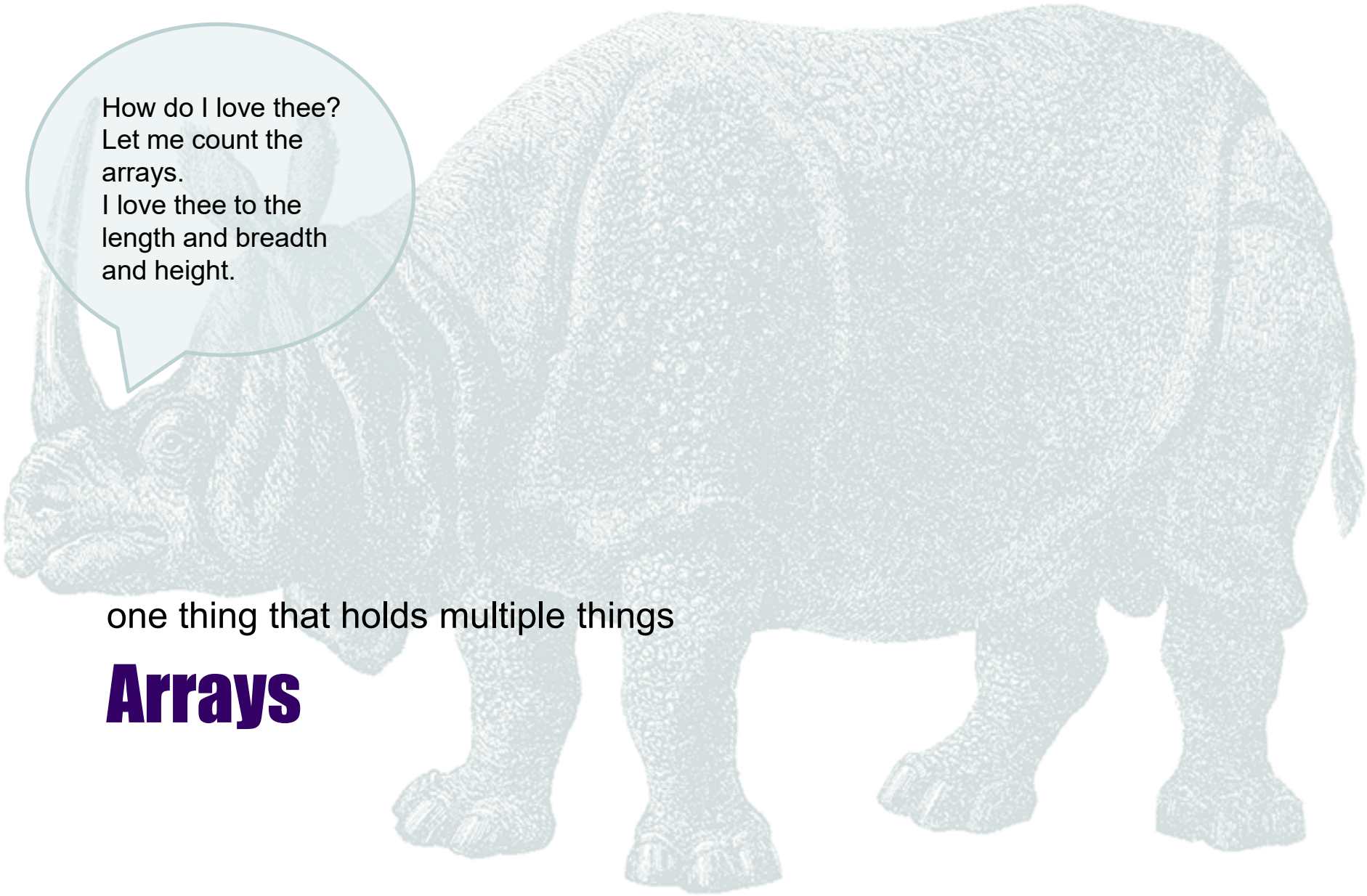
while loop

- **init** a variable
- while (**test** is true) {
  - do code;
  - **step** the variable;
- }

```
let 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)



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

```
let anArray = [1, ' 2 ', true];
let emptyArray = [ ];
let 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)

Index in array

- check for element index with value
- returns boolean

```
0 in arrayOfValues
1 in arrayOfValues
```

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 has few values but many spaces

```
// store a number in index 0
var almostEmptyArray = [1];
almostEmptyArray[100000] = 1;
almostEmptyArray.length;

var bunchesUndefined = [1,,,,,,,,,,,, ];
bunchesUndefined.length;
```

Indexes & properties

- Arrays can be initialized with either indexes
- Or properties (associative)
- Pick one style but not both - confusing

```
helloArray[0] = 'hello'
helloArray[1] = 'hi'

helloArray['cajun'] = "how y'all are?"
```

Property (associative) arrays

- An array based on properties rather than numbers.
  - Also known as a map, dictionary, or key-value pair
- Think of it as storage like an object field or metadata to an array.

```
poolBalls['yellow'] = 1;
```

Array indexes & properties

- Use a variable for its value
- Alternative dot notation syntax
  - not as flexible but easier to type

```
var two = 2
lettersWithMetadata[two]

lettersWithMetadata['charset']
lettersWithMetadata.charset
```

Array indexes & properties

- Positive integers and types coercable to positive integers (strings, floating point numbers) become indexes.
- Anything else is a property

```
var lettersWithMetadata = ['a', 'b', 'c'];
lettersWithMetadata['charset'] =
'Unicode';
lettersWithMetadata
lettersWithMetadata['charset']
lettersWithMetadata['0']
lettersWithMetadata[1.000]
```

for loop


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

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(key in window) {
  console.log(key, window[key])
}
```

Exercise


```
let eggCarton = new Array(12);
let 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


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

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)
```

The arguments array

- arguments = data passed to a function
- parameters = variables used to name the arguments
- the keyword **arguments** is an array-like object used in a function

```
function foo() { console.log(arguments); }
foo(1,2,3)
```

Exercise

```
function printArgs(a,b,c){
  for (argKey in arguments) {
       console.log(argKey, '=',
arguments[argKey]);
   }
};

printArgs(1,2,3,4,'five');
```

Array of functions


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

Array functions

- concat(arr1, arr2, ...)
  - Returns copy of target array and joins arguments
- join(separator)
  - Returns string of delimited values, comma default.

```
let a1 = [1,2,3]
let a2 = [4,5,6];
let c = []
c = a1.concat(a2);
c = a1.concat(a2, [7,8,9]);
console.log(c.join(' - ');
```

Array functions

- toString()
  - Returns the string form of an array.
- indexOf(value)
  - Returns int of position or -1 if not found.
- lastIndexOf(value)
  - Returns int of last position or -1 if not found.

Array functions

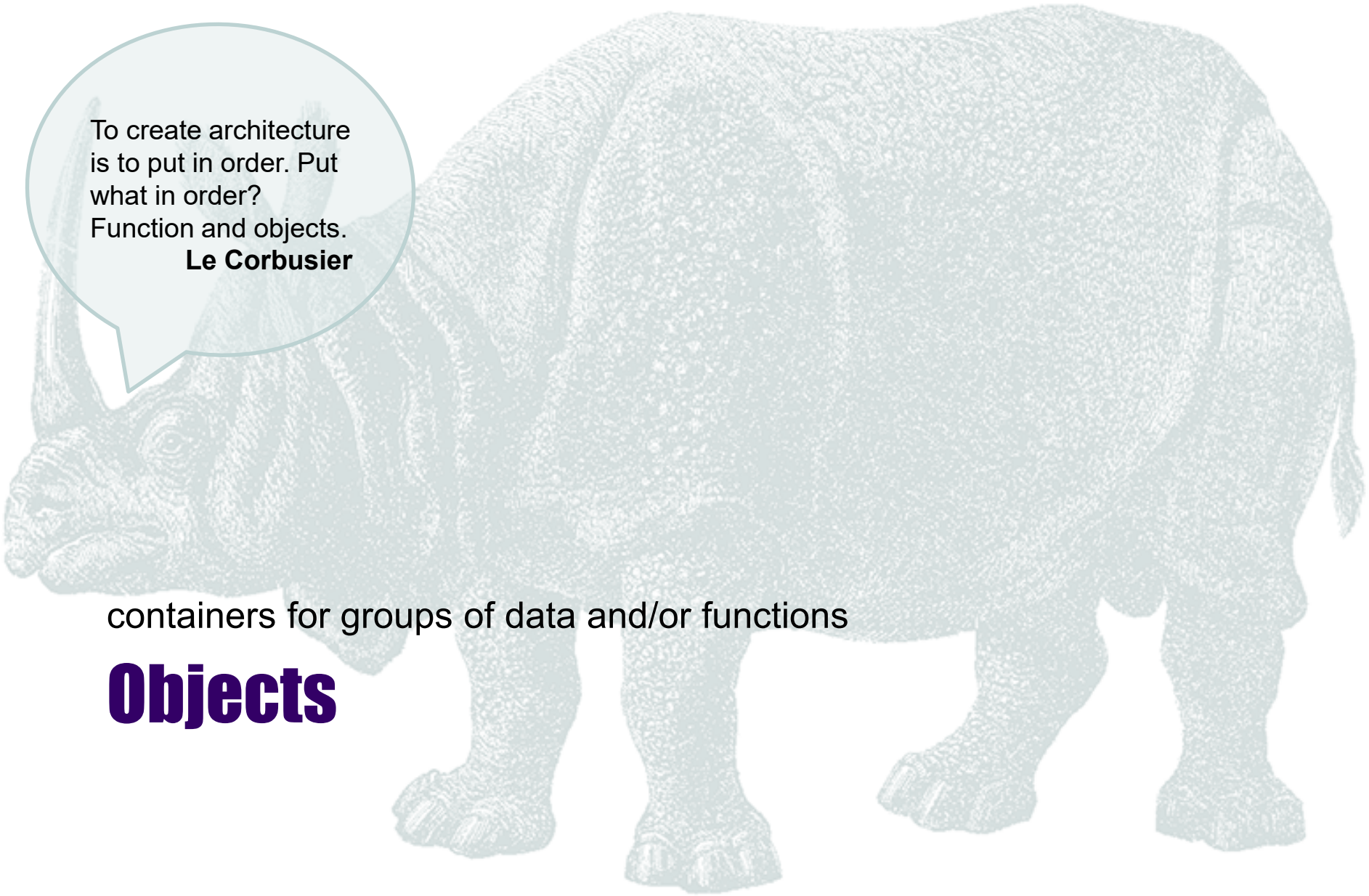
- shift()
  - Removes and returns first array element
- pop()
  - Removes and returns last array element
- push(item1, item2, ...)
  - Adds new elements to array end, returns length.
- unshift()
  - Adds new elements to array start, returns length.

Array functions

- sort(sortFunction)
  - Sorts the elements of the array. sortFunction is optional.
- reverse()
  - Reverses the order of array elements.
- valueOf()
  - Returns the value of an Array object.

Array functions

- slice(start, end)
  - Returns elements between start and end indexes.
- splice(index, count, item1, item2...)
  - At the index specified, removes count number of items and then inserts at index any optional items passed in as arguments.



containers for groups of data and/or functions

Objects

Intro

- Arrays are implemented using objects
- Functions are objects
- Indexes (keys) are converted to strings

Creating an object - object initializers

- Simpler than the constructor

```
objectName = {propertyKey:value, pk:v, …}
```

```
var myBear = {color:"brown",
state:"Alaska"};
myBear.color
```

Creating an object - object initializers

- Array syntax is clumsier
- But you can use variables

```
var myBearArray = [ ];
myBearArray['color'] = 'brown';
myBearArray['state'] = 'Alaska';
```

Creating an object - constructors

- Looks like a function
  - no return value
- Arguments passed to parameters are assigned to on-the-fly properties in the code block

```
function Bear(color, weight, state){
      this.color = color || 'brown';
      this.weight = weight || '1000';
      this.state = state || 'AK';
}
```

Creating an object - constructors


```
let myBear = new Bear(“black", 550)
```

- uses pre-existing fields

```
let 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 to single objects

- 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 = ( ) => {console.log ('Grrrrrr'); }
  - myBear.growl( )

Adding props/functions to all objects

- The prototype affects all Bear objects
  - current and future
- Open the __proto__ property in the browser

```
Bear.prototype.showDispleasure  = growl;
Bear.prototype.mood = 'happy';
whitey.showDispleasure()

var booboo = new Bear('brown', 500, 'ND');
booboo.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") ;
  - }

for-in with hasOwnProperty

- hasOwnProperty(key) will show only properties not shared in __proto__

```
for (const key in yogi) {
    if (yogi.hasOwnProperty(key)) {
        const bearValue = yogi[key];
        console.log(key, ' = ', bearValue,
'\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

- the keyword **this** refers to the object it's associated with

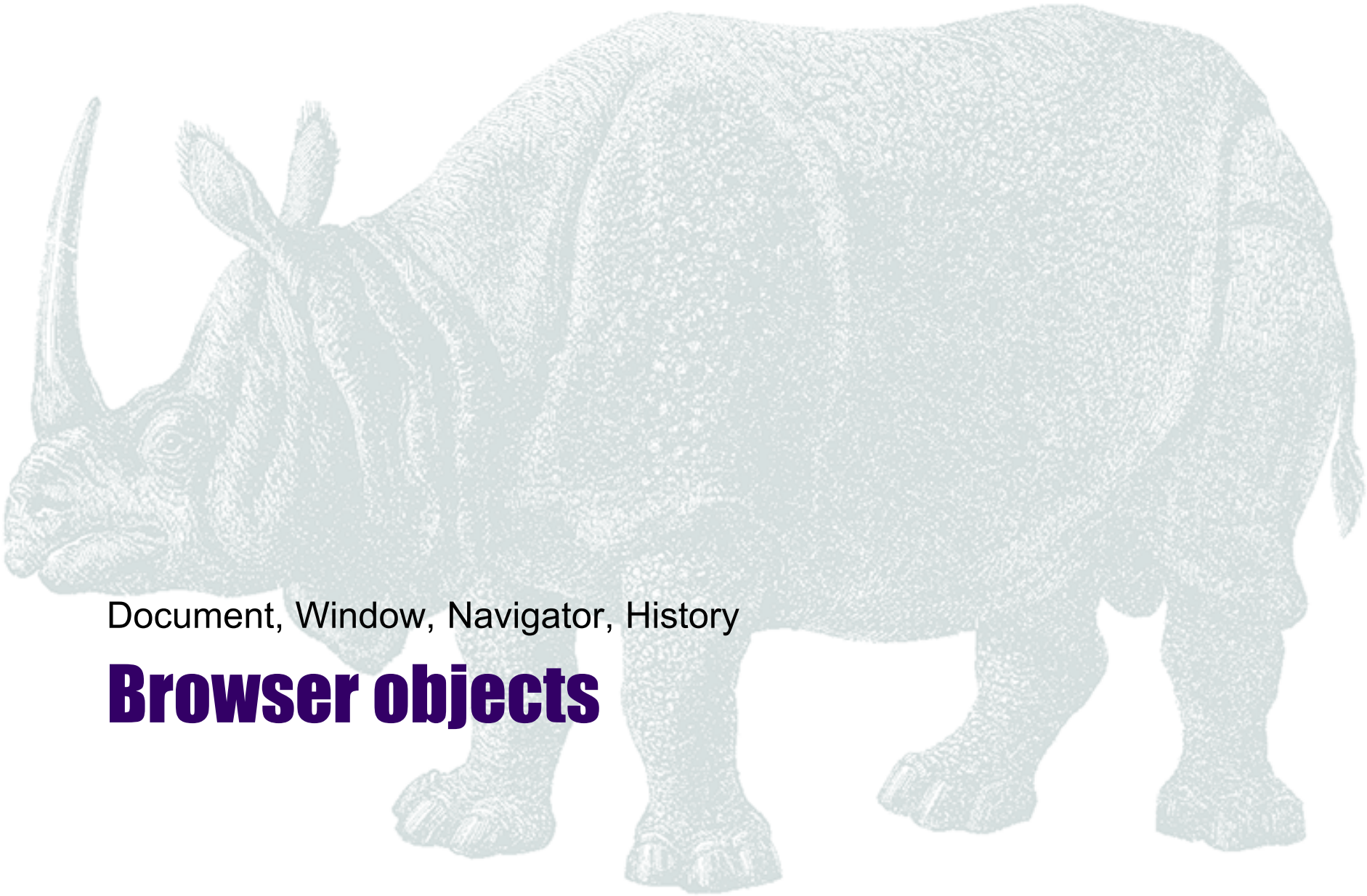
```
function listAllProps(){
  for(key in this){
    console.log(key, '=', this[key]);
  }
}
polarBear.listAll = listAllProps
polarBear.listAll( )
```

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

Best practice – use objects over arrays

- Object keys will overwrite each other
- Better for managing lists of things



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(#) // amount of pages
- 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

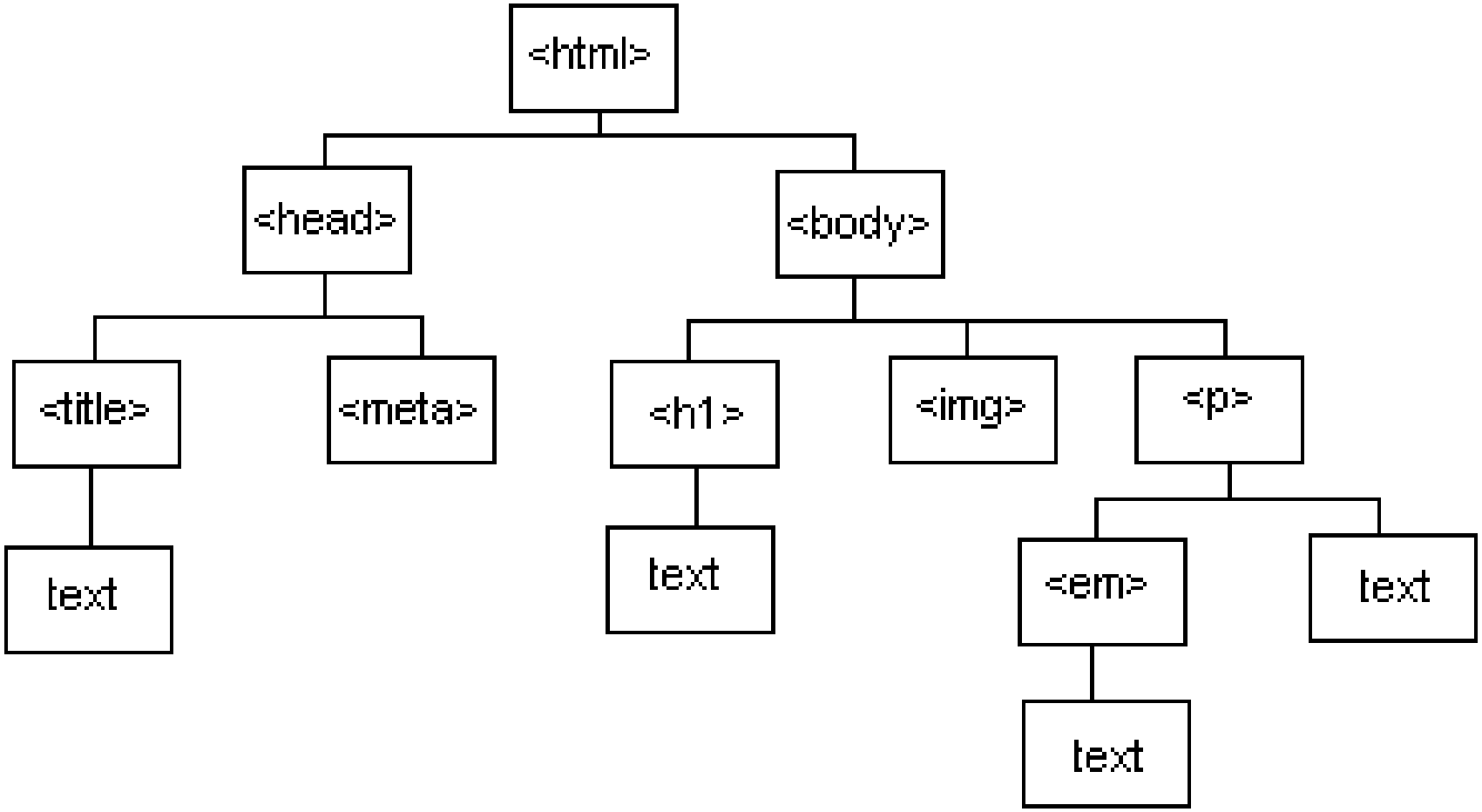
- a **redirect** is a client-side request after a response
- a **forward** is a server-side action returning a different page

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 (no hyphens allowed)
  - jQuery: $('#tb_name')
- **getElementsByName**(‘rb_removeItem')
  - will find
    - <input name= rb_removeItem type=radio value='1'>
    - <input name= rb_removeItem type=radio value='2'>
  - jQuery: $("[name=‘rb_removeItem']")

Selecting nodes

- **getElementsByTagName**("div")
  - will find
    - …
    - …
  - 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
    - …
    - …
  - jQuery: $('.subhead')
- multiple class names
  - separate them with spaces

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');

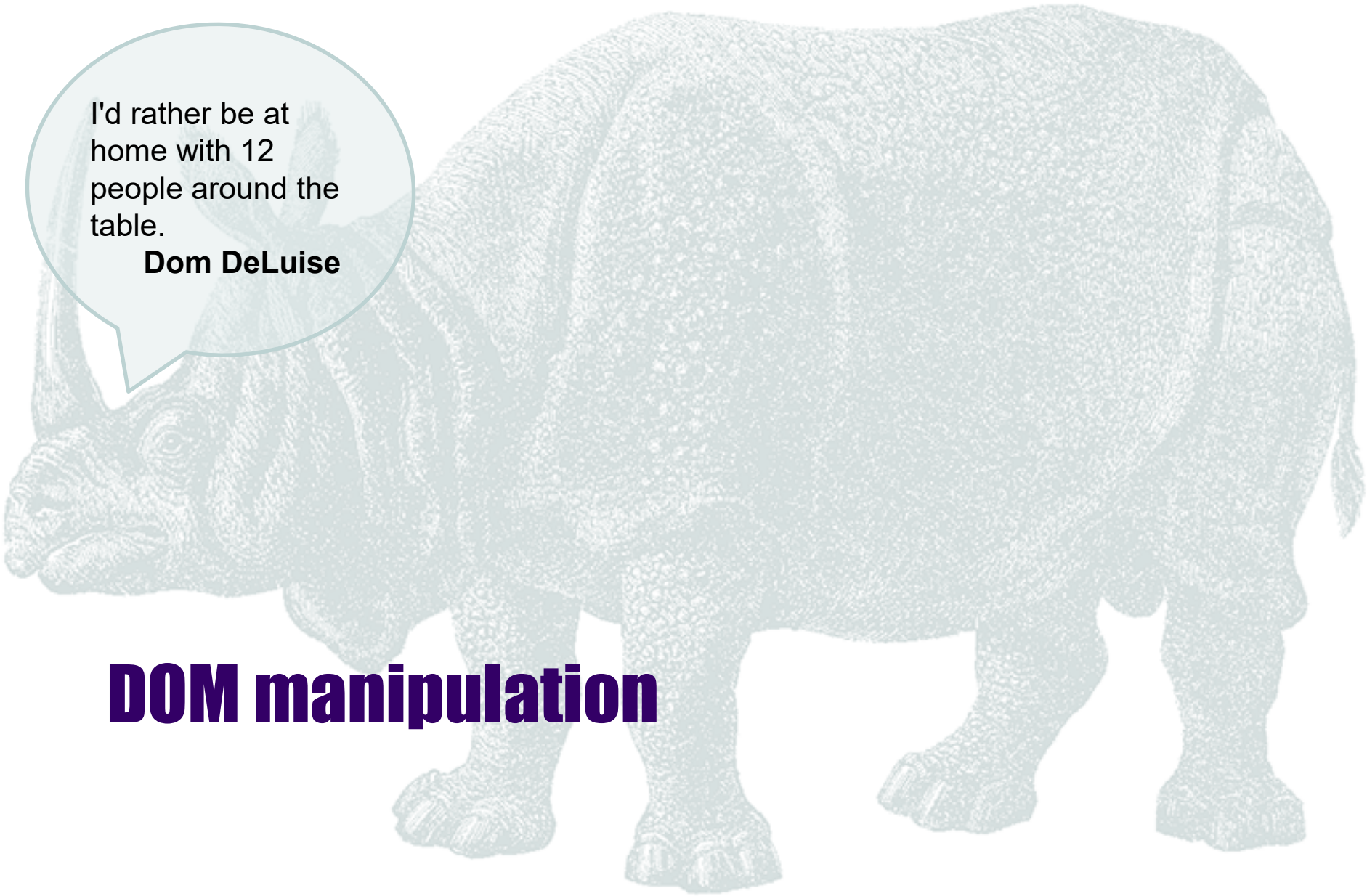
Other console shortcuts

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

Exercise

- on the Centriq.com home page try:

```
document.getElementsByClassName('menu-item')
document.querySelector('.menu-item')
document.querySelectorAll('.menu-item')
document.querySelectorAll('li.menu-item')
document.querySelectorAll('.menu-item a')
document.querySelector('#gf_5')
document.querySelectorAll('h1, h2, h3’)
document.querySelectorAll(‘div div div div
div’);
document.querySelectorAll('input[value][type=
"checkbox"]:not([value=""])');
```



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
  - or use centriq.com
- Select an image
  - you can try $0 first
  - then change to querySelector style selection
- 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’

```
corpLink.title = ‘new title’;
corpLink[‘title’] = ‘new title’;
let att = ‘title’;
corpLink[att] = ‘new title’;
```

Exercise: change style

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

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 – not in properties

- 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:

```
let divs = document.querySelectorAll('div');
for (let eachKey in divs) {
   divs[eachKey]['style'] =
     'border: 5px fuchsia dotted';
}
```

getComputedStyle( )


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

dom/events/fade-elements.html

Exercise

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

//repeat
newOpacity *= .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

- element.remove( )
- node.removeChild(node)
- process
  - find the immediate parent (not ancestor)
  - remove the immediate child

```
var body = $('body')
body.querySelector('#element').remove();
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

- positions =
  - beforebegin
  -
    - afterbegin
    - beforeend
  -
  - afterend

```
nodeVar.appendChild(node)
nodeVar.insertAdjacentHTML(position, text)
nodeVar.insertAdjacentElement(position, element)
```

Move an element

- Use a selection then insert to another place in the document.

```
list.append(cloneli);
list.prepend(cloneli);
```

Exercise

- // move Centriq header to bottom
- // get refs to elements

```
let header =
document.querySelector('header');
let body = document.querySelector('body');
```

// change CSS and move header to bottom of page

- `header.style.position = 'relative';`
- `body.insertAdjacentElement('beforeend', header);`

Exercise


```
// using https://www.showmeboone.com/
let list = document.querySelector('div.top-services ul');
let firstItem = list.querySelector('li');

// clone and modify
let cloneli = firstItem.cloneNode(true);
let innerA = cloneli.querySelector('a').innerHTML =
'JavaScript<br>class';
innerA

// remove first button
firstItem.remove( );

// insert clone as first list item
list.insertAdjacentElement('afterbegin', cloneli);

// ..or insert clone as last list item
// if already inserted, this will move the list item
list.appendChild(cloneli);
```

Useful value properties

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

Forms

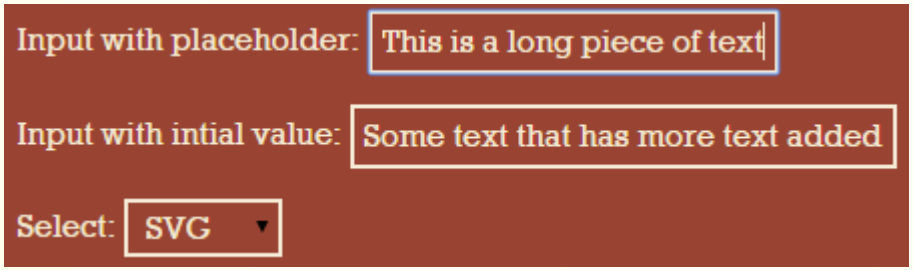
- get/set values of text box input
  - document.querySelector('[name=q]').value;
  - document.querySelector('[name=q]').value = 'JavaScript';
- textareas
  - document.querySelector('#textarea').value;
  - document.querySelector('#textarea').value = 'This is a text area.';
- drop downs
  - document.querySelector(‘#select-choice option').selected
  - document.querySelector(‘#select-choice option[value="Choice 2"]').selected = true;

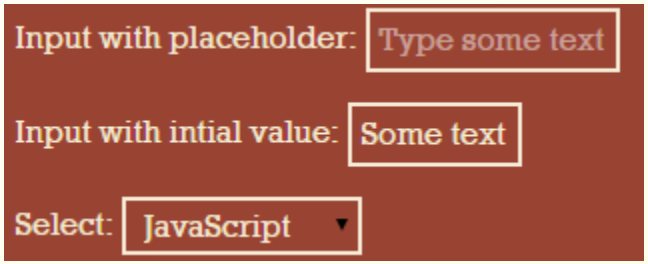
Forms

- radio buttons
  - document.querySelector('#radio2').checked
  - document.querySelector('#radio2').checked = true;
- checkboxes
  - document.querySelector('#checkbox').checked;
  - document.querySelector('#checkbox').checked = true;

Packages - forms

- Stretchy – autosizing
  - http://leaverou.github.io/stretchy/
- Autosize.js
  - http://www.jacklmoore.com/autosize/
  - textareas only





The user or browser did something. Do something back.

DOM 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();" …

Three essentials

- Write function to execute
- Pick element/node to attach/listen to
- Bind on available properties (onclick…)

Node event properties

- **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

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.

Binding/registering event handlers

- Give your element an ID
  - <input id="btn_submit" …
- Select 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

```
let submitButton =
document.getElementById("btn_submit");

submitButton.onclick = function ( ) { } ;
submitButton.onclick = ( ) => { } ;

function doWhenSubmitted() { }
submitButton.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

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, **target**
  - srcElement, srcElement.id
  - **screenX, screenY**
  - pageX, pageY, clientX, clientY (x,y)
  - offsetX, offsetY
  - which (keyboard key)

Exercise

- Add an event handler to Google's logo or doodle to tell you when you move the mouse over it.
  - console.log the message
  - use a mouseover event
- Extra credit:
  - Increase the size of the logo by 10% on mouseover using JavaScript
    - stylesheet: .grow { transition: all .4s ease-in-out; }
      (put in style)
    - stylesheet: .grow:hover { transform: scale(1.1); }
      (put in mouseover, revert to orginal size on mouseout)

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

Exercise

- Use the fade function to fade all the paragraphs using the button provided.

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)

Default action / bubbling / propagation

- e is the event object passed into the event handler
- e.stopPropagation( )
  - <a> will not bubble/propagate the click event to its parent
- e.preventDefault ( )
  - <a> will not request a page.
- put these at the beginning of the handler

Exercise

- 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;
}
```

attachEvent( )

- same as addEventListener() but will always bubble

```
elementVar.attachEvent('event', function)

detachEvent('event', function)
```

return false;

- combines
  - e.stopPropagation( )
  - e.preventDefault ( )
- traditional way to do it
- not best practice, but very common

```
// no propagation, no default
return false;
```

void(0)

- the void unary operator, requires argument
- the smallest script possible that evaluates as undefined
- used for Print to return to the same page
- otherwise returns a page of the evaluated expression

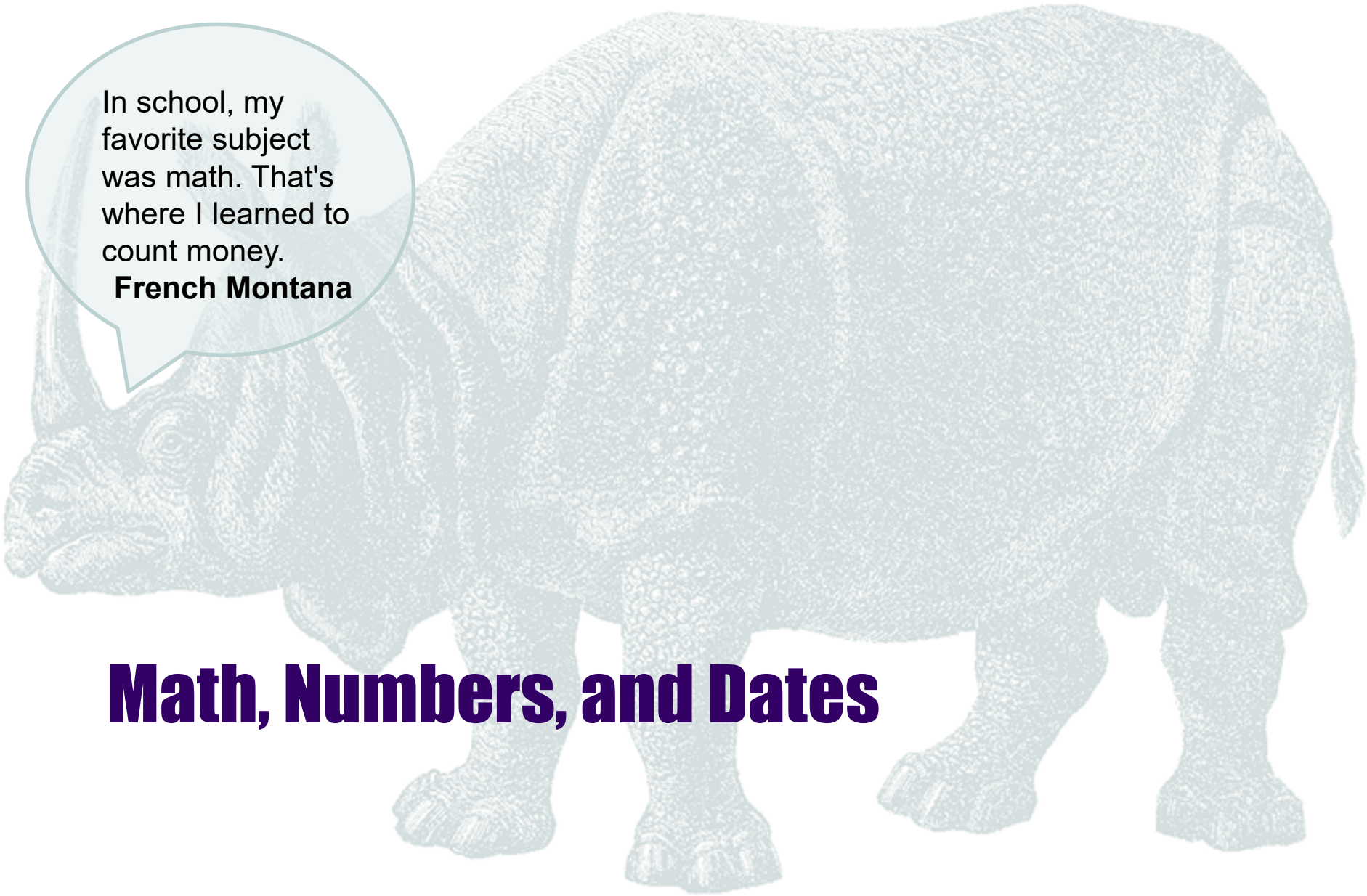
Exercise - form

- Create a web page
  - include a text box
  - include a button to bind an event to
- Append textbox text to the bottom of the document.
  - after append clear text field

Exercise - scroll

- Use kansascity.com or your own page
- Change the color of the screen gradually based on how much of the page has scrolled off the top

```
var newColor = 'hsl(' + pixelsScrolledOff
% 360 +', 50%, 50%)';
```



Math, Numbers, and Dates

Randomizing


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

randomBetween(0,5);
randomBetween(3,5);
randomBetween(6,5);
```

Random link


```
var urls = ["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

- Test numbers for validity with constants
- Can’t use Number.NaN

```
Number.MAX_VALUE
Number.MIN_VALUE
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 syntax
- new Date() is the current time/date
- Try it

```
new Date()
new Date().toTimeString()
new Date().toDateString()
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

Manipulation

- match() / search() – see regular expressions
  - http://regexlib.com
- 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

Manipulation

- toLowerCase() / toUpperCase()
  - Converts to lowercase/uppercase letters
- valueOf() - Returns the numeric value of a String object

```
"text".substring(1,3) //"ex"
"text".indexOf('x')   //2
"text".concat(" end") //"text end"
```

Combine Strings

- + operation
- concat()
- var word1 = "Today ";
  var word2 = "is ";
  var word3 = "tomorrows\' ";
  var word4 = "yesterday.";
- var sentence1 = word1 + word2 + word3 + word4;
- var sentence2 = word1.concat(word2, word3, word4);

Searching a String for a Substring

- indexOf( )
  - character not found = -1
- var myStr = "I search, therefore I am.";
  if (myStr.indexOf("search") != -1) {
    console.log (myStr);
  }

Replacing a Word in a String

- var username = "Brad";
- var prompt = "<username>, please enter your password: ";
- prompt.replace("<username>", username);

Splitting a String into an Array

- var time = "12:10:36";
- var timeArray = time.split(":");
  - var hours = timeArray[0];
  - var minutes = timeArray[1];
  - var seconds = timeArray[2];

Strings as arrays

- To iterate over a string you can use the for-in loop
- var s = ‘abc’
- for (key in s) {
  - console.log(s[key]);
- }

Selecting a partial string on input

- Leverage Google searches
  - https://support.google.com/websearch/answer/3284611?hl=en
  - https://support.google.com/websearch/answer/6276008?hl=en&ref_topic=3036132
  - https://support.google.com/websearch/answer/2364942?hl=en&ref_topic=3036132
- <form action="https://www.google.com/search">
- <input name="q" value="? reais in dollars " onfocus="this.setSelectionRange(0,1);"></form>

Libraries

- StringJS
  - http://stringjs.com/

Other languages

jQuery

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

jQuery 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)

Check for jQuery on site

- $ and jQuery are aliases
  - $ is reused in the browser
  - use jQuery to see if you get a function back

TypeScript

- https://www.typescriptlang.org/

TypeScript

- Install node.js to get npm
  - Download from https://nodejs.org
- Install typescript with npm
  - `npm install -g typescript`
- Create config file
  - tsc --init
- Run build task…
- or tsc
- or tsc <filename>

Code Runner

- Code Runner uses
  - npm install -g ts-node

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
  - Fetch - 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/
- Britecharts - http://eventbrite.github.io/britecharts/
- Chart.js - canvas
  - http://www.chartjs.org/
- *D3 - complex SVG
  - https://d3js.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 $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/