## Intro to Javascript

Welcome to class! A few notes:

* This course is full of practical web development topics (more than just JS!)
* Technical goals: understand web env, a bit about static sites, and fundamental JS. This course is typically used as prep for Hack Reactor
* Group goals: supportive, collaborative learning environment - feel free to ask questions to each other and instructors (Savannah and Peter).
* Google is a great friend - ask google many questions. Knowing how to Google something is a very important skill.
* Pedagogy: in this course, we will learn by **doing**. Practical skills vs lecture based approach.
* There is too much material here for us to get through in this session. I left as much content as I did so that people of different levels have enough to work on. Don't worry or feel bad if you don't finish a section, you're not expected to - rather, get the most solid understanding possible and play around with the concept :)
* You should have the [Google Chrome](https://www.google.com/chrome/) browser and [Atom code editor](https://atom.io/) installed before we begin - if you already use another text editor that you can use to write Javascript, feel free to use that instead.
* No experience is required to be successful in this course, open to all levels
* Note: there are references and tasks throughout this course that reference 'Divshot'. Divshot is a static file hosting site, that has recently been deprecated. Please ignore all references to Divshot, and if you have extra time throughout the exercises to deploy, set up [Heroku](https://devcenter.heroku.com/articles/getting-started-with-nodejs" \l "introduction) instead
* We will break around 12:30 for lunch. After lunch it is up to you if you want to continue working through the assignments or partake in the hackathon.

This course typically lasts a few weeks. Since we don't have time to cover everything today, we will focus on content from weeks 2 and 3 which I have trimmed down for this session. Click [here](http://morning-bastion-77397.herokuapp.com/" \l "06.html) to get started.

# JavaScript & the DOM

Objectives:

* Use the CLI to navigate the file system
* Use Node Package Manager to install software (libraries)
* Deploy sites to a static hosting service using the CLI

[download Node](https://nodejs.org/download/). (It's a JavaScript interpreter with other helpful tools)

### A Brief Detour Through the Terminal

The Terminal is a Command Line Interface which we can use to issue instructions to our computer. More specifically it's a REPL: Read Evaluate Print Loop.

A REPL waits for the user to input an instruction and press ENTER to signal they're done. The interpreter then READs the command, EVALUATEs (runs/ executes/ resolves) it, PRINTs the result, and LOOPs back to wait for another command.

If you've ever played a text adventure game, you've used a REPL:

You are in a small room with a window and a table.

> leave room

The door is closed.

> open door

The door is locked.

> look at table

There is a box on the table.

> open box

*"Thanks, for the history lesson, Grandpa!"*, but **what's it good for**? Well, hasn't deploying your HTML documents live using the GUI and the Divshot website been a pain? We can accomplish the same task with a simple command using our CLI: divshot push Let's find out how.

Let's open our Terminal after finding it with our Operating System [⌘ + SPACE] and issue some commands

date

cal

Often times when you see an example of commands to type into the Terminal, it will start w/ a $ or a >, which represents the command prompt. You don't actually type this in, the computer wouldn't understand, it's just an indicator to the programmer that this command should be typed into the command line, not written in a file to be run later.

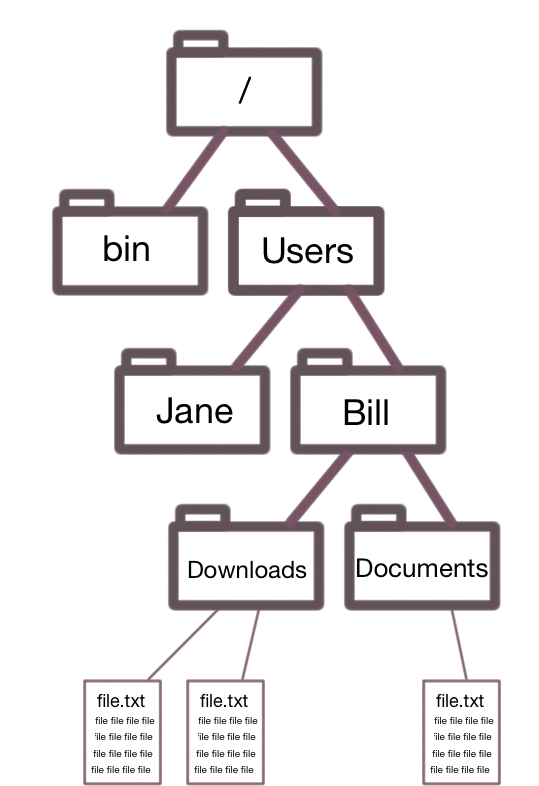
$ date

$ cal

First we'll have to **navigate the file system** to find the folder with our HTML documents. Type in the following to **P**rint your current **W**orking **D**irectory.

$ pwd

The result /Users/myName is a [file path](javascript:void(0)) representing your **home directory**- by default where you go when you login to your computer. If your computer has multiple logins, each user has their own home directory.



Starting with the root directory, file paths for the above file system might be/Users/Bill/Documents/file.txt to reference Bill's text document or /Users/Jane to reference Jane's home folder.

[⌘ + TAB] to switch to your GUI (called Finder) and open your home directory there too [⌘ + SHIFT + H]. There are all the files and folders. How do we **L**i**S**t them in the Terminal?

$ ls

The above lists the contents of the current directory by default.

$ ls /Users/YourName/Desktop

The piece of text after the command is an [argument](javascript:void(0)) to the ls [subroutine](javascript:void(0)) (AKA a function). In this case a file path indicating which directory to list the contents of.

According to the rules of the Terminal, when you press ENTER after typing in your command, your input is split on each space character, and Terminal interprets the first word as the name of the subroutine (lines of code) to be run. It invokes (runs) it, passing in all additional words as arguments (inputs).

When working from your home directory, the below should all do the exact same thing. Mouse over to see how the computer interprets each file path and **write your own ls commands to test it out!**

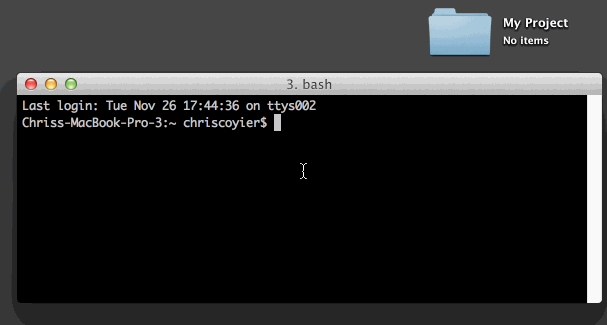
$ ls [/Users/YourName/Desktop](javascript:void(0))  
$ ls [./Desktop](javascript:void(0))  
$ ls [Desktop](javascript:void(0))  
$ ls [../YourName/Desktop](javascript:void(0))  
$ ls [../../Users/YourName/Desktop](javascript:void(0))

Do you think the last three file paths are ABSOLUTE or RELATIVE?

So we can click on folders to open them in our GUI. What about in Terminal? Let's **C**hange **D**irectory into the Desktop folder. (P.S. you can start typing Desk and then press [TAB] to autocomplete file and folder names).

$ cd Desktop

**Exercise:** Let's get really comfortable with file paths. Type cd and a space into the Terminal, and then using your GUI, try drag and dropping a folders into Terminal. Hit enter to run the change directory subroutine and then ls to see the contents of that folder. **Do this at least 5 times** with a variety of folders.



### Lots More!

You'll want to understand the following, at least at a conceptual level. **Try each command:**

pwd

ls .

cd [directory]

cd /

cd ..

cd ../..

ls

ls [directory]

mkdir [directory] = make a new directory

mv [file] [new file] = move/rename file

cp [file] [new file] = copy file

rm [bad file] = remove file

rm -rf = remove a directory and all contents

cat [file] = show file in Terminal

atom [file] = open file in Atom

atom [folder] = open entire folder as a project in Atom

open [folder] = open a folder in the Finder

open [file] = open a file using the default application for its type

(*alternately we could say:* invoke the application and pass this file as an input)

↑/↓ = scroll through command history

history = see full command history

!32 = re-run previous command #32

sudo [command] = run command as Administrator (root) <= fixes many installation errors

sudo !! = rerun previous as Administrator

[TAB] = autocomplete

node = start the node JavaScript REPL

^C = interrupt process (quit back to CLI)

^A/^E = go to start/end of line

⌘T = open new Terminal tab

Paths:

/ = root

~ = home

. = current directory

.. = parent directory

/Users/JimBob = absolute path

./Dropbox/Projects = relative path

Dropbox/Projects = shorter relative path

Flags:

command [-flags] [file file file...]

-h, --help

-v, --verbose or --version

-r, -R, --recursive

-l, --list

-f, --force

### More Information:

Want to know more? Check out [Learn CLI the Hard Way](http://cli.learncodethehardway.org/book/)

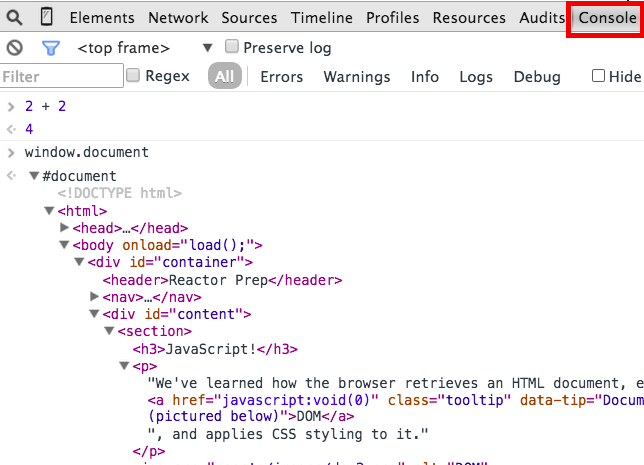
Objectives:

* Open and use the JavaScript Console REPL to evaluate simple expressions
* Command the browser to console.log, alert, and prompt
* Use <script> tags to add simple JavaScript functionality to our HTML documents

### JavaScript!

If we want more complex interactive behavior on a website we need to use a [scripting language](javascript:void(0)) like JavaScript to describe what should happen when and how.

Before we start throwing down <script> tags or creating our own .js files, let's write a few lines by hand. Go ahead and open up your Chrome Developer Tools ([CMD + SHIFT + ?] if you forget how) and switch to the JavaScript Console. The Console is a [REPL](javascript:void(0)).



### The JavaScript Runtime Environment

First off let's prove to ourselves that we are still living in the same world. Use the JavaScript console to instruct the browser to show you the DOM. (The > indicates that we are typing into a JavaScript REPL, not writing code in a file to be run later, omit it)

* > window.document

Open it up and take a look! The window object holds the state of (information about) the currently open browser tab. It contains many **properties** (simply put: associated pieces of data), of which, the DOM (document) is one. Here are a few more, **check 'em out**

* > window.location
* > window.history

You can also look at the window object directly, but be warned, **it's intense!**

* > window

Any of the properties listed can be accessed using the **dot notation** property access format as seen above with the document, location, and history properties of the window object. **Try accessing a few.**

Great, so we see that we're living in the same world, but how do we actually interact with the page? Make it respond to user actions? Change over time according to our wishes?

Well, we've got to take a few steps back to answer those questions. So strap yourself in... **JavaScript here we come!**

### The Number data type

Exercise: Type each of the following lines of code ([expressions](javascript:void(0))) into the console one at a time and consider the result.

* > 2 + 3
* > 3 \* 5 + 4
* > 3 \* (5 + 4)
* > -300 \* 2.57
* > [25 % 6](javascript:void(0))
* > 1e4 \* 2

What have we learned? We can use the console as a simple caluculator by employing values with the [data type](javascript:void(0)) of number and mathematical [operators](javascript:void(0)) such as +, -, \*, /, %.

**Exercise:** From the console, calculate how many seconds old you are.

### The String data type

* > "Hello, how are you?"

The string data type denotes a sequence of alphanumeric characters which are often useful for communicating information to weak-minded humans who aren't able to interact using pure logic.

What happens if we try to type the following into the console without the quotes? (find out for yourself, before checking the answer)

* > Hello, how are you?

We can evaluate non-numerical expressions as well. **To what value does the following expression resolve when evaluted?**

* > 'Just fine,' + ' ' + 'thank you'

String concatenation is the operation of joining character strings end-to-end.

### Alerting, Logging, and Prompting, oh my!

Not only does the window object give us access to the current browser tab's state, we can also use it to issue commands to the browser. **What does this command do?**

* > window.alert("Jello, whirled")

Similar to the Terminal, the first part of the command references what [*function*](javascript:void(0)) we're looking to run. Next comes the invokation operator () - similar to a mathematical operator ( +, -, \* ) - which causes the JavaScript interpreter to actually run the function. Inside the parentheses we place the arguments we wish to pass as input. (In this case the string to be displayed)

The [expression](javascript:void(0)) 2 + 3 resolves to the **value** 5. What value does the following expression resolve to? What is its **data type**?

* > window.prompt("How are you feeling today?")

The window context is implied.

* > alert("Jello, whirled")
* > prompt("How are you feeling today?")

Okay great, but I thought a scripting language was going to let us automate our work right? Typing things into the console isn't web scale!

In the same way that a <style> tag tells the browser to switch from the HTML interpreter to the CSS interpreter to read code inside it, the <script> tag tells the browser to interpret its contents according to the rules of JavaScript.

<html>

<head>

</head>

<body>

<h1>mah payge</h1>

<p>yup, yup</p>

<script>

// the interpreter will skip any line that starts with a double slash

// these lines are explanatory comments to help squishy-brained humans

alert("Welcome to mah deng website, brah!")

prompt("How's it going?")

</script>

</body>

</html>

**directly before the closing body tag </body>** in your HTML document. Always. Always. If you follow this simple rule, you will save yourself several dozen frustrating hours of debugging over the course of your career. You're welcome. *(p.s. yes, always)*

### Exercise: My First JavaScript Enabled Web Page

1. Create a new HTML document, correctly set up with html, head, and body tags
2. Add a script tag **directly before the closing body tag**.
3. Inside of a script tag, alert visitors that your site is extremely awesome and should thus be enjoyed with caution.
4. Prompt the user for their name
5. Greet them by name using another alert

In steps 4 and 5, the word alert came before the word prompt. **Why did the prompt happen first?** (I highly recommend you not move past this point until you can answer this question for yourself. Ask for help if you need to. *It's okay!*)

Ok, though illustrative, alerts are annoying. That's why they're not often used. It is more usual to use the console to output information. **Try this:**

<script>

window.console.log("Where is this message displayed?")

</script>

Or, more succinctly:

<script>

console.log("This is mostly used for debugging.", "We'll learn how to write directly into the HTML document a little later.")

</script>

**Exercise:** Ask the user what their favorite sports team is. Secretly log a message trash-talking their team in the console.

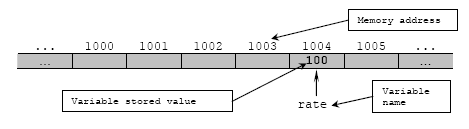
### Variables

In programming, it can be extremely useful to store values for later use. Variables are identifiers (labels, nametags) used to set and retrieve stored values. They are called variables because you can change what value they hold based on user input, the current program state, etc.

We use the var keyword to declare a variable. When first declared, variables have a value of undefined. The assignment [operator](javascript:void(0)) stores a value. **Type along in your console:**

* > var x
* > x
* > x = 1
* > x + 4
* > x
* > x = 5
* > var y = 2
* > x + y + 7
* > y = x + 7

So when we write var rate = 100 your JavaScript Runtime Environment jots down the variable "rate" in its ledger and gives it a pointer to a memory location:



Type the following in your console one line at a time (don't type the comments):

* > my\_variable // what does this error mean?
* > var my\_variable
* > my\_variable // what has changed from the first line?
* > my\_variable = "I'm a string value!"
* > typeof my\_variable

What is the value of first at the end of this sequence?

* > var first, second
* > first = "hello"
* > second = "goodbye"
* > first = second
* > second = "wait, you've forgotten your hat!"

In JavaScript, primitives (numbers, strings, booleans) are *pass by value*. This means that when you set one variable equal to another, the value stored there is copied over, and the two variables retain no lasting connection to each other. (objects are *pass by reference*, we'll learn how that's different later)

Now try some on your own:

#### Exercise 1:

1. Declare variables a, b and c. Assign a number to each.
2. Declare variable d and assign it the value of a, b, and c added together.
3. Re-assign the value of b to a. Then change the value of b. What is the value of a?

#### Exercise 2:

1. Create variables "hour" and "minute" to reflect the current time.
2. Write an expression to calculate how many seconds until midnight.
3. Change "minute" to be accurate again as time has probably passed.
4. Run the same calculation again (press the up key in the console to scroll back to it).

#### Exercise 3:

1. Declare at least 10 variables such as 'name', 'age', 'birthday', 'hometown', 'occupation', 'favoriteFood', 'bestFriend', etc, and set their values equal to strings containing the appropriate information about yourself.
2. console.log a paragraph about yourself using the appropriate variables to insert the relevant information in the appropriate places.
3. Create a new HTML document and re-write the variables and the console.log in a <script> tag so that you'll have it for later.

### Manipulating the DOM

console.log is great for debugging, but more frequently, we'll want our JavaScript to change the visible page. First, let's ask the document to return us the first element that it finds that matches a valid CSS selector.

Open [this page](http://morning-bastion-77397.herokuapp.com/students/examples/selecting.html) and try the following in the JavaScript console:

* > document.body
* > document.querySelector('p')
* > document.querySelector('#primary')
* > document.querySelector('.important')
* > document.querySelector('p.important').textContent
* > document.querySelector('#nav ol li').textContent
* > document.querySelector('h1, p').textContent = "I'm on the page!"
* > document.querySelector('h1').textContent = "Behold! For I am a mighty web document!"
* > var big\_shot = document.querySelector('#primary')
* > big\_shot.textContent = "Flippin' heck, I'm by far the most important element around!"
* > document.querySelector('.important').textContent = "Do all .important elements change or only one?"

**Exercise:** Design an HTML page with HTML elements to match the selectors used above. With that page loaded into the browser, type the above expressions into the console. **Do your changes persist if you reload the page? Why is this?**

Now move the JavaScript code into a script tag just before the </body>. **What's changed?**

### Greeter!

1. Create a fresh HTML document with correctly set up html, head, and body tags
2. Create a <div> with an id of greeting on your web page.
3. Prompt the user to enter their name. Set their response equal to a variable visitor\_name.
4. Create a variable welcome\_message and construct a welcoming message using their name.
5. Make your welcome message visible on the page using querySelector and textContent.
6. Prompt the user for their hometown and save their answer as a variable town
7. Put an empty <span> with the id of hometown on the page in the middle of an existing sentence already on the page letting them know how nice it is to have a visitor from *blank*.
8. Use the town variable and the #hometown span to make the compliment complete!

### Shuffle!

1. Create a <div> with an id of source, and one with the id of destination, each with some text inside.
2. Fetch the textContent of source and save it as a variable.
3. Use the variable to place that text in the destination div
4. Create <div>s with a ids of swap\_1 and swap\_2.
5. How can we swap the textContent of the two divs?

### Mad Libs!

1. Create a "Mad Libs" web page. It should tell a story, leaving empty <spans>'s with id attributes of 'noun1', 'adjective2', 'verb3', etc. Practice using the console along with `querySelector` and `textContent` to fill in the missing information one at a time.
2. After doing this manually, put these operations into a script tag, and use variables to store each word to be filled in before placing it into the web page, so that the template will be filled out automatically each time the page loads, but the information will be easy to change later.
3. Finally, prompt the visitor repeatedly to provide their own mad libs words. Save their responses as variables and then place those variables into the page to make a crazy story for them.

### [Optional Content]

#### Advanced Mathematical Operators

Play with the following in your console until you understand how they work:

num = 1

num = num + 5

// assignment

num += 5

num -= 1

num \*= 3

num /= 0.5

// increment and decrement

num++

num--

// operators before the variable

// what's the difference?

++num

--num

#### Escaping Strings

> "Well now...\nThis is " + "rather clever, \n\tdon't you think?"

An escape character indicates an alternative interpretation of the subsequent character in a character sequence. In JavaScript the slash character denotes that the character after it might be a tab (\t) or new line (\n).

> '"Single quotes work too" ,she said.' + " " + 'Though I\'d not thought it possible'

Escape characters may be needed to indicate that we're looking to print a literal single (\') or double quote (\") and not signal the end of a string of text.

**Fix the following so it will run successfully in your console:**

'"What's wrong with the pudding?", she'd say with a smile.'

**Now this one:**

'I'd often think' \* "to myself, "Well, what's the worst wrong that could go wrong?". I guess" \* "now we know..."

# If / Else

### Booleans & Comparators

The Boolean [data type](javascript:void(0)) has two possible values: true or false. We can type them in literally. Follow along in the console:

* > true
* > false

...but more typically we encounter them as the resultant value produced by evaluating [expressions](javascript:void(0)). Type out the following expressions, comprised of number values and comparator [operators](javascript:void(0)), and determine what**boolean** value each evaluates to:

* 5 > 3
* 5 < 3
* 5 >= 5
* 10 <= 100
* 5 === 5
* 5 !== 5
* 5 === "5"

What is the difference between 5 and "5"? Perhaps the typeof operator can help us out:

* > typeof "hello"
* > typeof true
* > typeof 5
* > typeof "5"

A piece of data of type string and another of type number will never truly be equivalent, no matter how similar they look to mushy, imprecise human brains.

#### Exercise:

1. Use the console to write an expression which compares two strings and produces a true value.
2. Write an expression comparing two strings which produces a false value.
3. Determine whether string comparison is case sensitive
4. Determine whether string comparison is space sensitive (does adding extra spaces change the outcome?)
5. Write an expression that considers the equivalency a string and a number and produces a false value
6. Write an expression that considers the equivalency a number and a string and produces a true value

### Conditional Statements

The if statement runs a series of lines of code (a **code block**) depending on whether a programmer-specified boolean condition evaluates to true or false. Since these are multi-line statements, they'll be hard to type into the console. Let's download a plug-in for our Atom code editor to evaluate JavaScript on the fly.

1. Open the Atom code editor and bring up the preferences [CMD + ,]
2. On the left, select "Install"
3. Search for "javascript-eval" and install the package of that name
4. You should now be able to select any valid JavaScript (make sure you have your language set to JavaScript or HTML) and press [CMD + I] to evaluate it.

Now try out some of the following in a new JavaScript document (don't foget [CTRL + SHIFT + L] to activate JavaScript mode):

if (true) {

console.log('oh yeeeeahhhhh!!')

}

if (false) {

console.log('wait, where am I??')

}

if (5 > 4) {

console.log('Makes sense..')

}

if (2 === 5) {

console.log("if you are seeing this message, something's gone terribly wrong")

}

if ( prompt("How you doing?") === "good" ) {

console.log("Well, I'm very glad to hear that...")

}

**Exercises:**

1. Write a if statement that always runs its code block
2. Write a if statement that never runs its code block
3. Write a statement that logs a message of encouragement if a visitor, when prompted, responds that they are sad.
4. Write a statement that alerts a warning if a visitor, when prompted, does not guess that the secret number is 3.
5. Prompt the visitor to enter a password when they visit your site. If it is correct, welcome them. Otherwise, direct them elsewhere using this line of code:

window.location = "http://www.google.com"

**Exercise:**

1. Type out each of the following examples. No copy & paste
2. Add JavaScript and/ or HTML code to activate alternate paths through the code blocks
3. Write your own version of each conditional statement structure (from scratch) and test its behavior

var hero, bad\_guy

if (hero === "strong") {

if (bad\_guy === "weak") {

console.log('Ah-ha, an easy victory!')

}

console.log("Let us battle to the death!")

}

var response, image\_path,

secret\_saying = document.querySelector('#message').textContent

if (secret\_saying === "An Eagle Flies at Midnight") {

response = "You may pass"

image\_path = "images/open\_door.png"

} else {

response = "Release the hounds!"

image\_path = "images/attack.png"

}

document.querySelector('#result').textContent = response

document.querySelector('img').src = image\_path

var user\_input = document.querySelector('input#search').value,

output\_el = document.querySelector('#output')

if (user\_input !== "happiness") {

if (!user\_input) { // this tells us something important about empty strings

// What would happen if we tried to use? user\_input =

document.querySelector('input').value = "hey you, type here :z"

} else {

output\_el.textContent = "You're searching for the wrong thing"

}

} else {

output\_el.textContent = "ahh, the enlightened one!"

}

// another approach

if (user\_input === "happiness") {

output\_el.textContent = "ahh, the enlightened one!"

} else if (!user\_input) {

document.querySelector('input').value = "hey you, type here :z"

} else {

output\_el.textContent = "You're searching for the wrong thing"

}

if (age < 21) {

output\_el.textContent = "sorry, son"

} else if ( !(age > 35) ) {

output\_el.textContent = "id please"

} else if (name === "Bob") {

output\_el.textContent = "you're banned, bob!"

} else {

output\_el.textContent = "come on in!"

}

What would happen if Bob was 35 or younger? How can we fix this?

### Logical And && , Logical Or ||, Ternary Operator

Type each of the following into the console. What do they each evaluate to?

* > !true
* > !false
* > !!false
* > !!true
* > !!"type coercion"
* > !!"all strings are 'truthy' except for..."
* > !!""
* > true && true
* > true && false
* > false && true
* > false && false
* > true || true
* > true || false
* > false || true
* > false || false
* > true && "what does this evalute to?"
* > false && "and this?"
* > true || "sigh, i'm not needed"
* > false || "default value"

[SHIFT + ENTER] let's you type multiple lines into a single console input.

* > if ( !name || !email || !phone ) {

form\_message = "Please enter all required fields"

} else {

form\_message = "Form accepted, thank you!"

}

### Pseudocode:

As your logical formulations get more complex, it can be helpful to write out your plan-of-attack in comments and then turn each piece into code.

// If asked on a date, I'll be excited to go if

// they are both financially stable and not a jerk

// Additionally, if I have less than three dates this week or don't have weekend

// plans, I'll still go, but not be as enthusiastic

// Otherwise I'll politely turn them down

if (financial\_status === "stable" && personality !== 'jerk') {

console.log("Absolutely, I'm free on Sunday!")

} else if ( dates\_this\_week < 3 || !weekend\_plans ) {

console.log('meh, why not...')

} else {

console.log("Oh, so nice of you to think of me, but my friend is in town and...")

}

**Exercise:** In a script tag, create variables name, age, gender, and is\_member, thirsty anddrink\_of\_choice and assign them appropriate values or no values at all. Then compose a paragraph in a div#greeting according to the following rules: (**Write it in pseudocode first, then JavaScript**)

1. If they are not a member or if they are under the age of 21, address them as sir or ma'am as appropriate and apologetically deny them access to the club.
2. Otherwise, greet them as Mr. or Mrs. (as appropriate by gender) and their name, and welcome them. If they are thirsty, prompt them as to what drink they would like.
3. If their favorite drink is anything but "Coors Light", tell them "yes" (name) "one" (drink\_name) "coming right up!". If they ask for a Coors, kick them out.

The logical or can be used as a **Default Operator** to provide a fallback in case a particular value is undefined or an empty string.

* > var greeting = "Hello there, " + (prompt("What's your name?") || "anonymous")

The logical and can be used as a **Guard Operator** which only reveals its second value if the first value is truthy.

* > var secret\_code = admin && "The password is 'cheese'"

The ternary operator is structured *conditional ? value\_if\_true : value\_if\_false*.

var response = user\_input ? "Response Received" : "Empty string, please type in a value"

document.querySelector('#message').textContent = (name === 'Bob') ?

"Damnit Bob, I've told you " + (++number\_of\_rejections) + " times..."

: "Oh hello, " + name + ". Welcome!"

**Exercise:** Create a "choose your own adventure" style text adventure game by using multiple prompts and branching if/ else conditional statements. Write a story into the HTML document on the basis of the user's responses to the prompts.

# While Loops

## Fun With Loops!

The **while loop** looks a lot like an **if statement**. They both execute their associated **code block** based on the result of their **conditional expression**. The difference being, the while loop will repeatedly check its conditional expression and continue to run its code block as long as it evaluates to true. **Give it a try:**

var n = 0

console.log("I am called the Count... because I really love to count!")

while (n < 10) {

console.log(n, "ha-ha-ha")

n++

}

console.log('fin!')

**Exercise:** Write a while loop that counts from 0 to 50 by twos

var n = 100, song = ''

while ( n >= 0 ) {

if ( n !== 0 ) {

song = song + ' ' + n + " bottles of beer on the wall! chug, chug, chug"

} else {

song+= " no bottles of beer! stumble, stumble, stumble..."

}

n--

}

document.querySelector('#lyrics').textContent = song

Create a super-quick HTML document that displays the above. Pretty messy right? Now try the following:

var n = 100, song = '<ul class=lyrics>'

while ( n >= 0 ) {

if ( n !== 0 ) {

song = song + '<li>' + n + " bottle" + (n === 1 ? '' : 's')

+ " of beer on the wall! chug, chug, chug</li>"

} else {

song+= "<li>no bottles of beer! stumble, stumble, stumble...</li>"

}

n--

}

song += "</ul>"

document.querySelector('#lyrics').innerHTML = song

**Exercise:** Fencepost Loop

1. Create a variable n and assign it an integer.
2. Code a while loop that builds as a single string, the numbers 1 through n, separated by commas
3. How can we make sure not to have a comma after the last number? e.g., for n = 9 print "1, 2, 3, 4, 5, 6, 7, 8, 9"
4. Use .textContent to place that string into a div#target in the HTML document.
5. Now change your code to create a string full of <p> tags, with the number inside, and with <hr> dividers between them (but not after the last one), and use .innerHTML to place them into the page.
6. Give odd numbers a class of "odd", even numbers a class of "even." Have a css rule inside your <style> tag that styles them differently.

var n = 1, sum = 0

while ( n <= 10 ) {

sum = sum + n

n++

}

// Why n - 1 ?

console.log("The sum of the numbers 1 through " + (n - 1) + " is " + sum)

**Exercise:** Cumulative Sum

1. Write a while loop that computes the product of multiplying the first n positive integers: product = 1 \* 2 \* 3 \* ... \* n.
2. Use .innerHTML to place a formatted piece of text into the HTML document that explains what nwas and what the product of the numbers computed to.
3. Give the tags different classes and ids that match pre-written CSS with some styling to make it a bit prettier.

var n = 0

while ( n <= 20 ) {

if ( n !== 13 && n % 2 === 0 ) {

console.log("I like the number " + n + ", a nice even number.")

} else if ( n !== 13 && n % 2 !== 0 ) {

console.log("I like the number " + n + ", odd is okay too!")

} else {

console.log("la-la-la, nothing to say here")

}

n++

}

**Exercise:** Refactor the code above. What improvements can be made?

**Exercise:** FizzBuzz

1. Write a program that console.log's the numbers from 1 to 100
2. Now for numbers that are multiples of three print "Fizz" instead of that number
3. For the multiples of five print "Buzz" instead of that number
4. For numbers which are multiples of both three and five print "FizzBuzz"
5. Instead of console.logging construct a long string and use textContent to place it in a div
6. Now put each number in a div tag. Give 'fizz', 'buzz', and 'fizzbuzz' different classes and style appropriately. Use innerHTML to place it all on the page after the looping is done.

### Nested While Loops

var i = 0, j, n = 5

while ( i < n ) {

j = 0 // What happens if we remove this line?

while ( j < n ) {

console.log("i is", i, "and j is", j)

j++

}

i++

}

**Exercise:** n numbers are coming to a number party. Introduce them each to all the guests as they arrive. You should end up with output looking something like this one for n = 5:

"welcome 1"

"welcome 2, meet 1"

"welcome 3, meet 1 and 2"

"welcome 4, meet 1, 2 and 3"

"welcome 5, meet 1, 2, 3 and 4"

The **Math** object holds [properties](javascript:void(0)) and [methods](javascript:void(0)) related to Mathematics. Figure out what each of the following does:

* > Math
* > Math.PI
* > Math.E
* > Math.pow(9, 2)
* > Math.random()
* > Math.floor(7.2)
* > Math.ceil(7.2)
* > Math.ceil( Math.random() \* 10 )
* > Math.ceil( Math.random() \* 10 )

How about?

* > parseInt("23")
* > parseInt(23)
* > parseInt("boink")

var input

while( !(input = prompt('write something...')) ){

console.log("you didn't write anything")

}

console.log("yay, you did it! you wrote: " + input)

**Exercise:** Write code to ask a user to enter an integer greater than 0. Keep asking for a number until it is valid.

**Exercise:** Write code to play rock, paper, scissors with the user.

1. Ask for input until the user enters either "R", "P", or "S"
2. Use Math.random() to choose a play for the computer
3. Tell the user what the outcome was
4. Ask the user if they'd like to play again

**Exercise:** Guess a Number

1. Prompt the user for an input to determine what the highest integer n should be. use parseInt(n) to convert it from a string to a number. As long as the result is NaN, ask them again.
2. Generate a random number between 1 and n.
3. Ask the user to guess the number. Tell them if their guess is too high or too low.
4. Loop until they guess correctly
5. Use .innerHTML to show them something fun as a reward for guessing correctly.
6. Show them what number was the correct number in the end.
7. Use a counter to let them know how many guesses it took them to get the right number.

# jQuery

jQuery is a library (a bunch of pre-written code), written in JavaScript, designed to make it easier to manipulate a web page and do various web-related work.

**Exercise:** [Open it up](https://code.jquery.com/jquery-2.1.1.js) and skim over it. How many lines of code is it?

Consider downloading a [source code beautifier](https://chrome.google.com/webstore/detail/javascript-and-css-code-b/iiglodndmmefofehaibmaignglbpdald) chrome extension to auto-indent and syntax highlight source code [CMD + OPT + U]

<script src="https://code.jquery.com/jquery-2.1.1.js"></script>

<script>

$('body').text('hello?')

</script>

**Exercise:** Place the above in a fresh HTML document, just above the closing body tag. The first tag includes the jQuery libary and makes it available to the page. The second script uses it to do some work.

### The $ Character

In the same way that window stores an object with a bunch of [properties](javascript:void(0)) and [methods](javascript:void(0)) related to the web page, jQuery uses the $ variable to store an object that contains the library's functionality. To see if jQuery is included on the page, type $ into the JavaScript console. You should see the following in reply:

* function jQuery(selector, context)

$() is a [function](javascript:void(0)) designed to take in a string- a css selector- such as $('#primary') as input, and return all matching elements, wrapped with jQuery super powers.

**Exercise:** Referencing [the documentation](http://oscarotero.com/jquery/), **Create an appropriate HTML document on which to try out the following:**

$('body').css('background-color', 'red')

$('img').attr('src','http://i.imgur.com/nWGCwiE.gif')

$('h1').text('hello')

$('#notification').html('<div class="alert">Hold on! <b>an error</b> has occured!</p>')

$('.container').append("<p>What's the difference between <i>.append()</i> and .html()?</p>")

**Exercise:** Using [the provided HTML document](http://morning-bastion-77397.herokuapp.com/students/examples/jquery-selecting.html)

1. Include the jQuery library in the file
2. Write an expression using jQuery that targets all h1 elements and makes their font-color blue
3. ... that targets the body element and turns its background red
4. ... that targets the element with the id "everything" and increases its font-size
5. ... that targets all elements with the class "holder" and adds a black border
6. ... that targets all li elements within ul elements and bold them
7. ... that targets p elements coming immediately after an h1 element and makes it green
8. ... that targets the element with the id of "secret" and changes its display property to 'none'
9. ... now change #secret's display property back to block
10. ... that targets the element with the id of "change-me" and changes its text to a new sentence
11. ... that appends new text to the end of the sentence in #change-me
12. ... that replaces #change-me's contents with two new paragraph tags with content included
13. ... that appends a new paragraph to #change-me
14. ... that targets the all li elements and changes their text to a new sentence
15. ... that targets the all li elements and appends new text to all of them
16. ... that targets the element with the id of "everything" and appends an ordered list with three list elements inside

### Exercise: Theme Changer

1. Create a web page that asks the user what color theme they'd like to view the web page in.
2. Keep asking until they enter a valid input (hint: You can end the while loop manually using thebreak keyword)
3. Use jQuery to change the css of various elements on the page to use at least three shades of that color. e.g. If they choose "green", show a light green background, with green text and dark green borders
4. Use jQuery to display a message letting the user know what theme has been applied to the page
5. Have at least three color schemes for a visitor to choose from (maybe green, blue and red)
6. Place different images on the page depending on the theme
7. **Extra Credit:** Allow the user to enter in a hex value and programmatically create lighter and darker tones of that color.

### jQuery Effects

**Exercise:** Learn a little about [jQuery Effects](http://learn.jquery.com/effects/) Try each of the following:

* $('h1').show() - convenience method to set element's 'display' property to 'block'
* $('p#secret').hide() - set element's 'display' property to 'none'
* $('ol li').fadeIn() - fade in a set of matched elements over time
* $('p .comments').fadeOut() - fade out as set of matched elements over time
* $('div#thing').slideDown() - animate an element sliding down over time
* $('#message').slideUp() - animate an element sliding up over time
* $('.alert').slideToggle() - toggle between an element sliding up or down over time depending on whether it's visible or not

### Creating HTML elements using jQuery

Open an HTML document and type the following in, one line at a time.

var new\_div = $('<div class="important">Got a new div coming in here!</div>')

// can we see the new div on the page yet?

new\_div.css('color','red').attr('id','primary').text("Hi, I'm new here")

// is it on the page yet?

$('body').append(new\_div)

// how about now?

// what's this do?

new\_div.fadeOut(2000).delay(1000).fadeIn().slideUp(4000)

Here we are creating new jQuery-wrapped HTML elements in JavaScript memory, manipulating them, then appending them to the DOM, which makes them appear on the page. We are also using [method chaining](javascript:void(0)) to run multiple consecutive operations on a jQuery wrapped DOM node.

**Exercise:** Construct an HTML document entirely using jQuery...

1. Start with an empty HTML document with jQuery included.
2. Place a second <script> tag on the page to contain your JavaScript
3. Use jQuery to create a new <div> tag with an id of container and append it to the body
4. Use jQuery to create a new <img> tag and place it inside #container
5. Give the img a src attribute with the URL of a funny picture
6. Use jQuery to change the image to be really small
7. Now make it really big
8. Now add a few more images and some text captions and use various jQuery effects to make them dance around!
9. Use Math.random() and position:fixed to randomize the placement of the various HTML elements on your page. Every time you hit refresh, your page should rearrange itself.window.innerHeight and window.innerWidth might come in handy.

# Functions

A function is an object containing a subroutine that defines a sequence of lines of code, packaged as a unit. When defined and saved as a variable, we can reference it by that variable name, and use the invocation operator () to run it 0, 1, or many times.

// definition

var greeter = function () {

console.log( "Hello, nice to meet you." )

}

// do we see a console.log yet?

// reference

greeter

// do we see a console.log yet?

// invocation

greeter()

// do we see a console.log yet?

greeter()

greeter()

greeter()

// what about now?

The above is a function that creates side effects when run. The function's side effect is a message logged in the console. Run each of the above lines of code one at a time. At what point does the greeting actually appear in the console?

#### Exercises:

1. Define a function that console.logs a simple message and the time: new Date()
2. Invoke your function multiple times from the console.
3. Place your function in a script tag in an HTML document.
4. Invoke it multiple times in the script tag.
5. Open the HTML document in your browser and hit reload a few times.
6. Define a function that generates an enthusiastic string of text and uses jQuery to .append() it to the HTML document (in an #output div for example).
7. Invoke it a few dozen times from the console.

#### Submit Your Code

A function w/ a return value, but no side effects

var greeter = function () {

return 'Hello'

}

// saving the return value

var greeting = greeter()

// what value does the variable greeting hold?

// using the return value to compose larger expressions

console.log(greeting + ", nice to meet you.")

// what's the difference here?

console.log(greeter() + ", nice to meet you.")

The result of evaluating an expression consisting of a function reference followed by an invocation operator is the value to the right of the keyword return inside the function when the code is run.

var saying\_generator = function () {

var phrase = "Heeey, " + "it's the " + " Fonz."

return phrase

}

// What is the return value?

var saying = saying\_generator()

var broken\_saying\_generator = function () {

var phrase = "Heeey, " + "it's the " + " Fonz."

phrase

}

// What about now?

var broken\_saying = broken\_saying\_generator()

#### Exercises:

1. In a script tag, define a new function called get\_sentence
2. get\_sentence should select a sentence randomly between two or more options and return it
3. Save the return value to a variable called new\_sentence
4. Use jQuery to add the sentence inside a new <p> tag to the end of the HTML document
5. Refactor your code so that it doesn't need the new\_sentence variable
6. Use a while loop to run get\_sentence ten times in a row so we can make sure the sentence really is random.
7. What is the statistical likelihood that we randomly get the same sentence ten times in a row?

#### Submit Your Code

During every function invocation, you have access to the arguments keyword, which contains all the inputs to the function invocation. Play with this concept until *you're sure* you understand it.

var inspector = function () {

console.log(arguments)

}

// try each invocation individually and ponder the result

inspector(3)

inspector(3 + 7)

inspector(3, 7)

inspector("hello")

inspector("hello" + " " + "how are you")

inspector("hello", "how are you")

inspector("hello", 7, true, undefined, null, 3 + 12, "nice to" + " meet you")

#### Exercises:

1. Create a function log\_and\_return that console.logs all of its inputs and then returns them from the function invocation.
2. Store the return value as a variable returned\_values
3. Pass that variable as an argument to a second invocation of log\_and\_return

#### Submit Your Code

It's unwieldy to work with the arguments keyword directly. Usually we use named **parameters** to give our inputs (arguments) variable names for the length of the function invocation

var value\_logger = function (value) {

console.log(value)

}

value\_logger("Howdy ho, neighborino!")

// parameters and variables defined in function invocations are local to that invocation

value // ReferenceError: No variable 'value' exists

value\_logger(3 + 7)

// where's the seven?

value\_logger(3, 7)

var doubler = function (num) {

return num \* 2

}

// is it ten?

var should\_be\_ten = doubler(5)

var double\_value\_logger = function (value1, value2) {

console.log(value1, value2)

}

double\_value\_logger("hello", "how are you")

// what is value2?

double\_value\_logger("hello")

var add = function(num1, num2){

return num1 + num2

}

var sum = add(7, 12)

**Exercises:** Simple Math

1. Write a function called tripler that takes a number and returns triple the value.
2. Create a function multiply that takes two numbers as inputs and returns their product
3. Create a function divide that takes two numbers as inputs and returns the result of dividing the first by the second
4. Create a function remainder that takes two numbers as inputs and returns the result of modulo the first by the second
5. Using only the functions you wrote above, and no operators, calculate the value of tripling 5, multiplying that by 12, dividing by 2 and then finding the remainder of dividing that by 3.
6. Write 4 more lines of code that use all of your math functions in a single expression

#### Submit Your Code

**Exercise:** Constructing sentences

1. Write a function called string\_printer, that takes a string as an argument and uses$('#output').text() to place it into a specific <div> on the web page. Call it multiple times with different strings from the console.
2. Does string\_printer use a *side effect* or a *return value*?
3. Write a function called funny\_sentence that takes a noun, an adjective, a verb, and an adverb as inputs, and constructs a string of html text and uses $('#output').append() to place it on the page.
4. Put each word argument you pass in into spans that have css rules that styles them differently to make them stand out.
5. Invoke funny\_sentence 5 times from within a <script> tag and see the result on the page.
6. **Extra Credit:**Create a version of funny\_sentence that takes no inputs, but rather constructs a funny sentence on its own from randomly chosen words

#### Submit Your Code

**Exercise:** Remember while loops?

1. Write a function called factoral\_of\_n that takes a number and returns the [factoral](http://en.wikipedia.org/wiki/Factorial) i.e.factoral\_of\_n(5) should return 120 (5x4x3x2x1).

#### Submit Your Code

**Exercise:**Fibonacci

By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. For example, the first ten Fibonacci numbers are:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Write a function that accepts a number and returns the number at that position in the fibonnaci sequence.

#### Submit Your Code

# Events

// we've defined a function and set it equal to a variable

var fn = function () {

console.log('heeey')

}

// of course, we can invoke it manually

fn()

// Alternately, we can ask the browser to run it at a later time

// The below is called an **event handler**, take a guess at what it does...

// Add it to an HTML document containing a div#target and test it out

document.querySelector('#target').addEventListener('click', fn)

// Often times, instead of using a variable, we'll just define the function inline

document.querySelector('#target').addEventListener('click', function() {

console.log('same deal')

})

An **event handler** is a function (subroutine) that is run each time a specific type of input is received from the user.

#### Exercise:

1. Make the above code work inside a script tag in an HTML document (you'll need a #target element on the page)
2. Define a function beeper
3. Create a button on the page
4. Every time the user clicks a button, add an HTML element to the bottom of the page with the word "beep"
5. Also change the body's background color each time a user clicks the button. Hint:

var randomColor = '#' + Math.random().toString(16).slice(2, 8)

How does this work?

#### Submit Your Code

Ok, that's the native JavaScript syntax for adding **event handlers** to the page. Now let's see jQuery's way:

$('div').on('click', function(){

console.log('A div has been clicked!')

})

// shortcut

$('#target').click(function(){

$('#target').show().css('color', 'red').text('boom goes the dynamite!')

})

#### Exercise:

1. Make the above code work inside a script tag in an HTML document
2. Create a div with the text "show me the money". If the user clicks on it, show them the money!
3. Create a div with the text "show me the Oprah". If the user clicks on it, show them the Oprah!
4. Create a div with the text "show me the Opera". If the user clicks on it, show them the Opera!
5. Make it so that when a user clicks on one item to show it, all the others hide first if they are already shown.
6. When a user clicks on an item that is already shown, hide it instead.

#### Submit Your Code

$('#target').on('mouseover', function(){

$('#target').addClass('highlighted')

})

$('#target').on('mouseleave', function(){

$('#target').removeClass('highlighted')

})

#### Exercise:

1. Place the above code into a script tag in an HTML document
2. Make sure there is a #target element on the page and that it has a non-zero height and width
3. Define CSS rules for the 'highlighted' class
4. Make sure everything is wired up correctly and works as you would expect
5. Place an image on the page that disappears when you place your mouse over a particular div#magic. Maybe this one of [Margaret Hamilton](http://en.wikipedia.org/wiki/Margaret_Hamilton_(scientist)" \l "/media/File:Margaret_Hamilton.gif).
6. How many different ways can you think of to accomplish that task?

#### Submit Your Code

var mouse\_tracker = function(event){

console.log(event.pageX, event.pageY, !!event.which)

})

$('body').on('mousemove', mouse\_tracker)

#### Exercise:

1. How can we inspect the event object in the console? What properties and methods does it have?
2. Do click events and mouseover events get arguments passed into each invocation of their event handler functions as well? Do they have the same properties?
3. Create a small position: fixed <img> that follows the cursor around the screen. Maybe a [gif of a puppy walking](https://www.google.com/search?q=puppy+walking+gif&es_sm=91&source=lnms&tbm=isch&sa=X&ved=0CAcQ_AUoAWoVChMIno7mzKHexgIVUSmICh3S4Q_N&biw=1305&bih=732&dpr=1.1).
4. Figure out how to hide the cursor with CSS
5. Make the image follow the mouse only if the mouse button is pressed

#### Submit Your Code

In addition to the arguments keyword, each function invocation also has access to the **calling context** via the this keyword. Event handlers are called in the context of the element that received the event.

var el\_finder = function(event){

console.log("calling el\_finder for:", this, event.target)

$(this).attr('id', 'active');

}

$('div').click(el\_finder)

#### Exercise:

1. Place the above code in an HTML document with at least one div and test it out.
2. Create an HTML document [with at least 5 paragraphs](http://www.catipsum.com/)
3. Give each paragraph a unique id attribute
4. Create a div#output at the top of the page
5. Create a click handler that shows the id attribute of the last paragraph that was clicked on in the div#output
6. Change the color and background-color of a clicked-on div to make it stand out
7. When you click on another div, clear the special styling of any perviously clicked-on divs
8. Is this easier to do by styling directly with .css() or by adding and removing a class name? Try both ways

#### Submit Your Code

$('button#go').on('click', function(){

$('#output').text( $('input#color').val() )

})

#### Exercise:

1. Guess what does the code above does. How would you go about testing your theory?
2. Place the above code in a fresh HTML document
3. Create the correct HTML elements referenced by the click handler
4. When a user types a color into the input and presses the go button, change the background color of the page to that color

#### Submit Your Code

$('input#type').on('keypress', function (e) {

$('label').text(e.keyCode)

})

$('body').on('keypress', function(e){

String.fromCharCode(e.keyCode)

})

#### Exercise:

1. Try out the above code in an appropriately set up HTML document
2. What's the difference between the event's keyCode property and the actual letter?
3. What's the difference between putting the keypress handler on the input and on the body?
4. If the keypress handler is placed on the body and we type into the input, does the event handler run? Why is this?
5. Create a keylogger that concatenates every letter the user types to a string called "nsa\_surveillance"
6. Show the user everything they've typed in a div#output
7. Create your own version of [Type Racer](http://play.typeracer.com/)

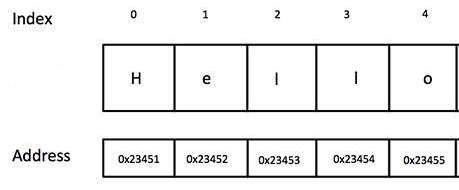
#### Submit Your Code

**Exercise:** There exist jQuery event handlers .focus() and .blur() [Figure out what they do](http://oscarotero.com/jquery) and then use them productively in an HTML document

#### Submit Your Code

# Arrays

Strings are secretly collections of letters stored at sequential memory addresses.



We can access each letter using square brackets and a zero-indexed number. **Try typing out each of the following for yourself, using your own values and variable names:**

var word = "Hello"

word[0]

word[1]

word[4]

word[5]

var index = 3

word[index]

// strings have a length property

var len = word.length

word[len]

word[len - 1]

Works for larger strings too.

var greeting = "Hello and welcome to my webpage!"

greeting[0]

greeting[1]

greeting[28]

greeting[29]

var index = 2

greeting[index]

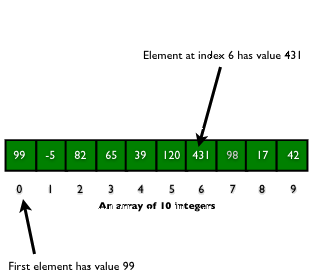
var len = greeting.length

greeting[len]

greeting[len - 1]

Sometimes we want collections of items more complex than single letters. Here's a collection of numbers contained in a [data structure](javascript:void(0)) of type array- a data type defined at the language-level by JavaScript.

var numbers = [99, -5, 82, 65, 39 120, 431, 98, 17, 42]



var nums = [5, 10, 20, 50, 100, 250]

nums[0]

nums[1]

nums[5]

nums[6]

var index = 4

nums[index]

var len = nums.length

nums[len]

nums[len - 1]

An array of strings:

var words = ["Hello", "and", "welcome", "to", "my", "webpage!"]

words[0]

words[1]

words[5]

words[6]

var the\_number\_three = 3

words[the\_number\_three]

var len = words.length

words[len]

words[len - 1]

Arrays can hold all kinds of stuff:

var all\_kinds\_of\_stuff = ["Hello", 3, undefined, true, ["woah", "dude"], null]

all\_kinds\_of\_stuff[0]

all\_kinds\_of\_stuff[1]

all\_kinds\_of\_stuff[5]

all\_kinds\_of\_stuff[6]

var another\_array = all\_kinds\_of\_stuff[4]

// ?

another\_array[0]

// ?

all\_kinds\_of\_stuff[4][1]

Okay, now we've seen some arrays, let's create a few of our own.

// creating a new instance of an array is called **instantiation**

var arr = []

// we can use bracket access in conjunction w/ the assignment operator

// to store a new value at that index

arr[0] = "0 is the new 1"

arr[1] = 45

arr[2] = null

arr[3] = true

arr[1] = "overwrite 45, just forget about that one, okay?"

**Exercise:** Favorite Recipe

1. Instantiate a new array and save it to a variable recipe
2. Use bracket notation to add the steps to create your favorite meal.
3. Access the first step of your recipe using bracket notation.
4. Create variable called last\_step and assign to it a number which represents the index of the last item of your array
5. Access the last step of your recipe using the last\_step variable

#### Submit Your Code

Arrays have associated with them some [properties](javascript:void(0)), like .length, and [methods](javascript:void(0)), like .push(), to help us use our collections more effectively.

var arr = ['stuff', 'more stuff', 'even more stuff']

arr.length

arr.push("More on the end!!!")

var lastItem = words.pop()

arr.unshift("More at the beginning!!")

var firstItem = words.shift()

**Exercise:** Bucket List

1. Instantiate a new array and save it to a variable bucket\_list
2. Use bracket notation with the assignment operator to add two life goals to your array.
3. .push() two goals onto the end of your list.
4. .unshift() two goals onto the beginning.
5. What is the array's length at this point?
6. .pop() off one goal and save it to a variable dying\_wish
7. .shift() off one goal and save it to a variable do\_it\_now
8. What is the array's length at this point?

#### Submit Your Code

Often we want to consider every element in the array in turn.

var i = 0

while (i < words.length) {

console.log(words[i])

i++

}

**Exercise:**

1. Use a while loop and jQuery to append each item in your recipe and your bucket list onto a web page.
2. Refactor your code such that the while loop lives inside a function called append\_strings that takes an array as an input (argument)
3. Invoke append\_strings twice, passing in recipe the first time and bucket\_list the second time
4. Refactor append\_strings such that it takes two arguments: an array of strings, and a string representing the CSS selector of the element into which the strings should be appended. (i.e.bucket\_list may be appended into a #bucket\_output div)

#### Submit Your Code

var str = "one two three"

var arr = str.split(' ')

var new\_str = arr.join(' and a ')

// now our turn...

var scrambled\_poem = "roses red are bacon crispy i bacon love and is blue violets are"

**Exercise:**

1. use .split(' ') on scrambled\_poem and save the result as an array scrambled\_array
2. Create a new array unscrambled\_array
3. While the length of scrambled\_array is greater than 0, remove the FIRST and LAST value and store add them to unscrambled\_array.
4. Turn the values in unscrambled\_poem into a string by using the array's native .join(' ') method.

#### Submit Your Code

We do this type of iteration so often, that JavaScript provides us a special kind of loop- the for loop.

for (var i = 0; i < words.length; i++) {

console.log(words[i])

}

**Exercise:**

1. Write a function reverser which will take an array of values as an input and return a new array of the same values in reverse order.
2. Inside reverser instantiate a new array reversed\_array, use a for loop to populate it, then return it from the function.

#### Submit Your Code

Often it's cleaner to use Array's native .forEach() method to run a given function once for each item in the array, passing that item in as an argument

var log\_me = function(word){

console.log(word)

}

words.forEach(log\_me)

**Exercise:**Refactor your append\_strings function to use .forEach() instead of a while loop.

**Exercise:**Write a function biggest\_smallest that takes an array of numbers as an input, uses.forEach(), and returns an array containing the smallest number in the zeroth position and the largest number in the first position.

#### Submit Your Code

Arrays are pass by reference, this means when you do the following, you get two arrays pointing at the same object in memory. This can lead to the unexpected results

var arr2 = arr

arr2.pop() // both arr and arr2 are changed!!

var arr3 = arr.slice() // clones the array

arr3.pop() // only arr3 is changed

**Exericise:** Create a function array\_duplicator, that takes an array as an input, and returns a copy as the output.

var array\_duplicator = function(inputArray) {

var outputArray = []

// code here

return outputArray

};

var original = [1, 2, 3, 4]

var duplicated = array\_duplicator(original)

duplicated.pop()

duplicated.pop()

duplicated.pop()

console.log(original, duplicated) // should not be the same!

#### Submit Your Code

.concat() is short for concatinate. It smooshes two arrays together.

var friends = ['ada', 'will', 'bianca', 'abe']

var enemies = ['john', 'alice']

var frenemies = friends.concat(enemies)

console.log(frenemies)

console.log(friends, enemies) // does not change original arrays!

**Exericise:** Write a function array\_combiner that takes two arrays as inputs and returns the two of them smooshed together as an output

var array\_combiner = function(array1, array2) {}

var combined\_array = array\_combiner([1, 2, 3], [4, 5, 6])

console.log(combined\_array)

#### Submit Your Code

.splice() is another confusing one. It destructively removes a number of elements from an array starting at a particular index.

var frenemies = ['ada', 'will', 'bianca', 'abe', 'john', 'alice']

var starting\_at\_index = 3

var number\_to\_remove = 2

var removed\_elements = frenemies.splice(starting\_at\_index, number\_to\_remove);

console.log("We took out ", removed\_elements)

console.log("The remaining array ", frenemies)

**Exercise:** Write a function array\_plucker that takes an array and an index as inputs and destructively removes the single value at that index. Return that value.

var array\_plucker = function(array1, index\_to\_remove) {

var removed\_value

// code here

return removed\_value

}

var days\_of\_school = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

var day\_off = array\_plucker(days\_of\_school, 4)

console.log("I'll go to school on: ", days\_of\_school)

console.log("but I'm taking vaction on ", day\_off)

#### Submit Your Code

var rand = Math.floor( Math.random() \* words.length )

words[rand]

#### Exercise: Magic 8-ball

1. Create a Magic 8-ball that allows a user to type a question into an input field.
2. When the user clicks the "ask question" button, register a click handler function that randomly selects a reply from an array of appropriately vague stock phrases.
3. Display the answer inside a glowing circle or something crazy like that.
4. Allow the user to click an "ask again" button which clears their previous question and output.
5. Refactor your code so that the user doesn't receive the same reply twice until all the replies have been used at least once.

**Exercise: Shuffler** Create a function shuffler that takes an array as input and returns a new array of the same elements in a random order. Is your shuffle perfectly mathematically random or only pseudo-random? How computationally efficient is it- i.e. how many lines of code need to be run?

#### Submit Your Code

#### More Exercises:

1. Write a function that takes an array of values and returns an boolean representing if the word "hello" exists in the array.
2. Write a function that takes an array of values and a target value and returns how many times that target value exists in the array.
3. Write a function that takes an array and returns a new array containing only the values at odd indexes in that array.
4. Write a function called sum\_array that takes an array of numbers and returns the sum of all of those numbers added together.
5. Write a function called array\_math that takes an array of numbers and a string with a mathmatical operator and applies that operator to all the numbers together. array\_math([4,2,3], "\*") should return 24
6. Write a zipper function that combines two arrays in alternating order, zipper([1,2,3],['a','b','c']) should return [1, 'a', 2, 'b', 3, 'c']. Make sure to handle cases where one array is longer than the other.
7. Write a function that merges two sorted lists into a new list. merge\_sorted([1, 2, 5, 9], [3, 7, 19]) should return [1, 2, 3, 5, 7, 9, 19]
8. Write a function that creates an array of the first 100 Fibonacci numbers.

#### Submit Your Code

#### Exercise: Magnetic Poetry

1. Have a grid of words in span.word tags for the user to select from.
2. Allow users to click on the spans to add that word to a sentence array.
3. When the user clicks on a "make sentence" button use words.join(' ') to create a string and show it.
4. Create a "show magic word" button. If the user has used a particular magic word in their sentence highlight it for them. Otherwise, tell them that they have not used the magic word.
5. Have an array of magic words, color the magic words differently.
6. Allow users to create multiple sentences. Store all of the sentences in an array.
7. Have "magic word" work on all sentences at once.

#### Submit Your Code

### [Optional Content] Advanced Concepts in Arrays

var arr = [52, 27,31]

// pass by reference

arr2 = arr

arr2.push(44)

arr[0] = "new value"

// why?

console.log(arr, arr2)

// what does slice do?

arr2 = arr.slice()

arr2.push(44)

arr[0] = "new value"

// why?

console.log(arr, arr2)

var arr = [ ['hello', 'and', 'hi'], [2,3,4] ]

arr2 = arr.slice()

arr2[1].push(44)

arr[0] = "new value"

// ?

console.log(arr, arr2)

var array\_of\_functions = [ function(){ console.log('hi') }, function(){ console.log('bye') } ]

array\_of\_functions.push(function(val1){ console.log(val1 \* 2) })

var adder = function(val1, val2){ console.log(val1 + val2) }

array\_of\_functions[3] = adder

array\_of\_functions.forEach(function(func){

func(3,4)

})

// Reimplementing JavaScript Array's native .forEach() method is rather trivial

var my\_each = function (array, iterator) {

for (var i = 0; i < array.length; i++){

iterator( array[i], i )

}

}

my\_each(some\_array, some\_function)

# Objects

JavaScript has two built-in [associative array](http://en.wikipedia.org/wiki/Associative_array) data types to help us store and work with *collections* of values: Arrays, which are collections of values accessible at sequential integer keys...

var my\_array = ["one", "two", "three"]

my\_array[0]

my\_array[1]

my\_array[2] = "not three"

... and Objects, which are collections of values accessible at arbitrary string keys:

var person = {

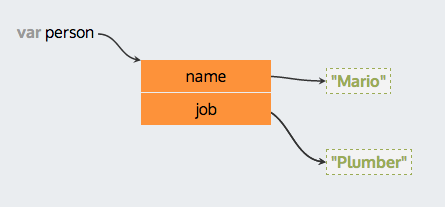
"name": "Mario",

"job": "Plumber"

}

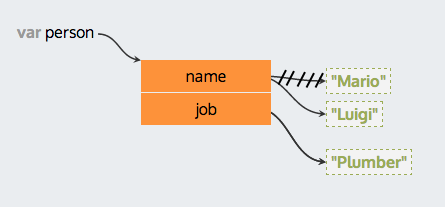
person["name"]

person["job"]



Like arrays, we use the bracket syntax to access and set values

person['name'] = 'Luigi'



#### Exercise:

1. Create a variable called my\_cat and assign to it an empty object
2. Assign my\_cat a name property, which should hold a string value
3. Add an age property holding an integer value
4. Add an alive property holding an boolean value
5. Add at least four more properties such as favorite\_toy, and napping\_spot

#### Submit Your Code

Something to note- because property keys are always string values, quotation marks are optional:

var person = {

"name": "Mario",

"job": "Plumber",

"age" : 55,

"licenced": true

}

var person = {

name: "Mario",

job: "Plumber",

age : 55,

licenced: true

}

Often times we access properties of our objects using variables instead of string literals:

var human = {}

human['name'] = 'bob'

human['age'] = 75

var property = 'age'

// what property key are we referencing here? what is its value?

human[property]

property = 'location'

// at what *key* will we find the value "San Francisco"?

human[property] = 'San Francisco'

#### Exercise:

1. Create a variable called my\_dream\_life and assign to it an empty object
2. Create a variable called property\_key and assign to it the string value "job"
3. Use the property\_key variable, not the literal string "job" to add a job property to your dream life object.
4. Create a variable another\_property\_key and give it a value "salary".
5. Use another\_property\_key, not the literal string "salary", to quantify the value of human life.
6. What does my\_dream\_life[property\_key] return?
7. What does my\_dream\_life["job"] return?
8. Assign a string "car" to the variable next\_property
9. Use the variable next\_property to add your dream car to your dream life object.
10. What does my\_dream\_life[next\_property] return?
11. What does my\_dream\_life["car"] return?
12. Assign the my\_dream\_life three more properties using the property\_key variable rather than literal strings and access them each using variables as well

#### Submit Your Code

We use the person["job"] syntax so often, JavaScript decided to include a more confusing way to do the same thing: person.job. It's called **dot notation** as opposed to the **bracket notation** we're used to.

var car = {

make: "Toyota"

}

car["model"] = "Camry"

car["model"]

car.model

car.model = "Prius"

car["model"]

car.model

var prop = 'year'

// how can we set the car's year using the variable prop?

#### Exercise:

1. Which should we use above: car[prop] = 1992 or car.prop = 1992? Try both ways!
2. What happens when you ask for a property that is not registered to the object? i.e. to what value does the expression car['nothing\_here'] resolve to? what about car.nothing\_here?
3. Create your own dream\_car object.
4. **Use dot notation** to add properties make, model, and color with values of type string, year with a value of type number, and registered with a value of type boolean.

#### Submit Your Code

**Exercise:** On lines ending '??', figure out what the expression will evaluate to. **First form a hypthosis**, then **try it in the console**.

var demo\_object = {

one: 1,

two: 2,

three: 3

}

var one = "three"

demo\_object['two'] // ??

demo\_object[one] // ??

var state\_capitals = {

California: 'Sacramento',

Texas: 'Austin'

}

var place = 'California'

state\_capitals['place'] // ??

state\_capitals.place // ??

state\_capitals[place] // ??

state\_capitals['California'] // ??

state\_capitals.California // ??

state\_capitals[California] // ??

Although it's less common than with arrays, sometimes we'll want to iterate over every property in an object. There is a special language-level statement to handle this, the for in loop.

var obj = {

greeting: 'howdy',

direction: 'down',

color: 'red',

'spaces-and-dashes can be used': ' but need to have quotes'

}

for (var key in obj) {

console.log(obj[key]) //why don't we need quotes around key??

// what would happen if we logged obj.key instead? Why? Test it.

}

#### Exercise:

1. Use a for in loop to iterate over your cat, dream\_car and dream\_life objects. Re-write the loop from scratch each time and use a variable name other than key for each one.
2. Use a for in loop to search obj for the key color and use jQuery to append its value to a special div on your web page.
3. Search obj for the value "down", and append its key into the div.
4. Create a variable target\_key and assign it a string "greeting". Use the target\_key variable andfor in to find the greeting property and append its value into the div.

#### Submit Your Code

var numbers = [2, 4, 5, 37, 0]

var doubled\_numbers = {} // should be { 2: 4, 4: 8, 5: 10, 37: 74, 0: 0 }

**Exercise:** Take the array numbers and place each value into an empty object where the object's keys are the numbers and the values are double that number.

#### Submit Your Code

#### String Operations

var greeting = "Hello and welcome!"

greeting[0] // "H"

!!greeting.match("welcome") // true

!!greeting.match("goodbye") // false

var words = greeting.split(' ')

**Exercise:** Loop through the famous array add only people whose names start with 'a' to your favorites

var favorites = []

var famous = ['alex smith', 'amy whinehouse', 'cameron diaz', 'brad pitt',

'ashton kutcher', 'mark whalberg', 'morgan freeman', 'mila kunis']

#### Exercise:

1. Loop through the politicos object and add each person's name to the famous array.
2. Loop through the politicos object and add the names of jobs that people with Obama in their name hold to the obama\_jobs array.

var obama\_jobs = []

var politicos = {

secretary\_of\_state: 'hillary clinton',

potus: 'barack obama',

flotus: 'michelle obama',

vice\_prez: 'joe biden',

gov\_of\_california: 'jerry brown'

}

#### Exercise:

1. Create a new empty object b\_named\_politicos
2. Add any politico whose first OR last name starts with a 'b' as a property of the b\_named\_politicosobject with their job title as the key and their name as the value
3. Create a new empty object reverse\_politico\_lookup
4. Create a reverse lookup table by iterating over the politicos object and adding each person to thereverse\_politico\_lookup object with their name as a key and their job title as the value.

#### Submit Your Code

Examine the unexpected behavior regarding arrays and objects...

numbers[0]

numbers["0"]

doubled\_numbers[0]

doubled\_numbers["0"]

typeof 37 // 'number'

typeof "hello" // 'string'

typeof undefined // 'undefined'

typeof function() {} // 'function'

typeof { a: 1 } // 'object'

typeof [ 1, 2, 3 ] // 'object' (?!?! not useful)

Array.isArray({ a: 1 }) // false

Array.isArray([ 1, 2, 3 ]) // true

**Exercise:**

1. Using the below as a starting point, write a function that takes any value and tells you what type it is.
2. Make sure this handles Arrays correctly!

var type\_tester = function(input) {

console.log(input)

}

type\_tester(3)

type\_tester('hello')

type\_tester({})

type\_tester([])

#### Submit Your Code

Just like arrays, objects can hold values of any data type. That includes arrays, objects, and functions.

var complex\_object = {

string: "I'm a string!",

number: 42,

array: ["all sorts of stuff", 10, true, undefined, {me: "an object", more: "find me in here!"}, function(){console.log('beepity-beep')}],

simple\_object: {name: 'bob', location: 'basement'},

fn: function(){

console.log("fn has been invoked! Checking both kinds of invocation-time inputs...")

console.log("what is my calling context?", this)

console.log("what are my arguments?", arguments)

}

}

complex\_object.string

complex\_object.array[4]

complex\_object.simple\_object.name

complex\_object.fn("stuff", "more stuff", "even more stuff!")

#### Exercise:

1. Load complex\_object into a script tag in an HTML document.
2. Use jQuery to append complex\_object's' string and number properties into a div#primative\_data\_types
3. Iterate over each property in complex\_object.array and use jQuery to append its value to a div#array\_values. Decide for yourself what to do in the case of objects or functions.
4. Place bob's name and location in their own div
5. Make the code that logs 'beepity-beep' run
6. Understand the various aspects of the complex\_object.fn method
7. Create your own complex object and try placing some of its properties on the page. How complex can you make it?
8. Give your object a property whose value is an array of other objects that themselves each have at least one function and one array property.

#### Submit Your Code

Arrays and objects are often used in combination

var them\_dogs = [{

name: 'Henry',

age: 0.5,

breed: 'Aussie',

food: 'kibble',

toys: ['tennis ball', 'chew toy'],

picture: 'http://rubyriverminiaustralianshepherds.com/wp-content/uploads/aussie-puppy-for-sale-940x412.jpg'

}, {

name: 'Tilly',

age: 5,

breed: 'Mutt',

food: 'kibble',

toys: ['bone', 'kong', 'squeaky toy'],

picture: 'http://www.dogchannel.com/images/zones/top\_promo/excited-mixed-breed.jpg'

}, {

name: 'Apollo',

age: 10,

breed: 'Labrador',

food: 'absolutely anything',

toys: ['old sock', 'other old sock', 'more old socks'],

picture: 'http://media.cmgdigital.com/shared/img/photos/2014/08/01/5a/66/LadyLabrador.jpg'

}]

#### Exercises

1. Iterate over them\_dogs and log each dog's name
2. Iterate over them\_dogs and add all the toys to a new all\_toys array
3. Write a function that takes an array of dogs and a toy as an input and returns the name of the dog that owns that toy.
4. Using jQuery, place the pictures of each dog on a web page with its name underneath.
5. Create "Dogbook" (which is of course Facebook for dogs) and populate it using the above

#### Submit Your Code

Take a moment to appreciate all the hard work you've put in to become a programmer. You're doing a great job! Here, [this is for you...](http://cdn.iwastesomuchtime.com/632014032559.jpg)

# Hackathon

**Congratulations!** We are almost done with our class. Let's celebrate with a hackathon! A hackathon is a great opportunity to work on a larger, real-world project, while reviewing and solidifying the core concepts that we've covered.

### Step 1: Make a Plan

1. Take a few minutes to brainstorm some project ideas. This could be related to your community site, a stand-alone project, or just a crazy mash-up site to help you practice different coding concepts
2. Review the **Step 3: Coding** section below to consider the different concepts that we hope to practice during this project.
3. In groups of four, hold a *standup* meeting to present your plan of what you'd like to accomplish, and potential stumbling blocks.
4. If someone mentions being concerned about a concept that you feel solid on, offer to be a resource for them if they have questions.

### Step 2: Set up a GitHub repo

1. Open the **Terminal** and run:
2. git config --global user.name "YOUR NAME"
3. git config --global user.email "YOUR EMAIL ADDRESS"
4. Go to [GitHub](http://github.com/) and sign in or sign up for an account
5. Create a new repository on GitHub by clicking the **new repository** menu, upper-right
6. Follow the instructions presented in the new repository
7. If you'd like to know more about the philosophy behind version control, check out [The Git Parable](http://tom.preston-werner.com/2009/05/19/the-git-parable.html)

### Step 3: Coding

The following lists the core web development skills that you should feel comfortable using in your project thus far:

* Use alert, prompt, and console.log where appropriate
* Set and retrieve values from JavaScript variables
* Use document.querySelector in conjunction with textContent and innerHTML to retrieve and place content on the page
* Use if, elseif and else to implement branching logic on the basis of comparing variables and values
* Use logical and (&&), logical or (||), and the ternary operator
* Use while loops to do work multiple times until a condition is met
* Use the Math object and parseInt
* Use jQuery to select existing HTML elements on the page
* Use jQuery's .css(), .attr(), .text(), .html(), .append(), and .val() to both get and set values
* Use some jQuery effects like .show(), .slideDown() and .fadeOut()
* Use functions and events to make your page change
* Use arrays and objects to store and manipulate data

#### Stretch Goals

1. Use some of [Bootstrap's JavaScript utilties](http://getbootstrap.com/javascript/)
2. Try incorporating some [jQuery Plug-ins](http://jqueryhouse.com/50-jquery-bootstrap-plugins-for-your-next-projects/)

### Step 4: Deploy & Promote

1. Deploy your code to Heroku using your command line tools
2. Send it to your friends and family members so they can be proud of you
3. Social media?

### Step 5: Profit?

1. Monetize
2. Monetize
3. Monetize

### A note regarding best practices

Try to decipher the code below. What does it do?

var i = 1, n = 5, j

while (i <= n){

j = 1;

var msg = '';

if (i === 1) {

msg += 'Welcome ' + i;

} else if (i > 1) {

msg += 'Welcome ' + i + ', meet';

}

while (j < n) {

if( j > 0 && j < i) {

if (j === i - 1) {

if (j === 1) {

msg += ' ' + j;

} else {

msg += ' and ' + j;

}

} else {

if (j === 1 && i === 3){

msg += ' ' + j;

} else {

msg += ' ' + j +',';

}

}

}

console.log(msg)

j++;

}

Pretty tough right? Here are some problems:

1. No indentation
2. Obscure variable names
3. A bit verbosse

What happens when we clean it up?

var guest\_number, announcement\_message, arrival\_number = 1, total\_number\_of\_guests = 5

while (arrival\_number <= total\_number\_of\_guests){

guest\_number = 1, announcement\_message = 'Welcome ' + arrival\_number

if (arrival\_number > 1) {

announcement\_message += ', meet ' + guest\_number

while (++guest\_number < arrival\_number ) {

if (guest\_number !== arrival\_number - 1) {

announcement\_message += ', ' + guest\_number

} else {

announcement\_message += ' and ' + guest\_number

}

}

}

arrival\_number++

console.log(announcement\_message)

}