IMPORTANT NOTICE

FINAL PROJECT MILESTONE

This week, you should begin writing pseudo code and a draft of the HTML/CSS for your application - this should be turned into your instructor by the end of week 7.

FEWD - WEEK 5

WILL MYERS

Freelance Front End Developer

SLIDES

http://www.slideshare.net/wilkom/fewd-week5-slides

YOUR WEEKLY FEWD GITHUB REPOSITORY

- Use the '+' button in the top-left of GitHub Desktop (*Create* tab)
- Create a new repository called *'FEWD_Week5'*
- Choose the [home]/FEWD folder for the local path
- Open this repo folder in your editor
- Commit the changes and publish the *FEWD_Week5* repository to github.com

YOUR WEEKLY WORKING FILES FROM ME

To get the *week5_working_files* you should just be able to select the *ga-fewd-files* repository in GitHub Desktop and press 'Sync'. This should pull in this weeks folder from github.com.

If you any difficulties you should just re-clone the *ga-fewd-files* repository.

Copy the whole *week5_working_files* into your *FEWD_Week5* repository and commit and publish to github.com

AGENDA

- Review
- Variables
- Conditionals
- Lab Time

REVIEW

THE CONSOLE

https://repl.it/languages/javascript

```
console.log("something", 123)
```

VARIABLES

- We can tell our program to remember values for us to use later on.
- The action of saving a value to memory is called assignment
- The entity we use to store the value is called a variable
- The action of getting the value from a variable is called *accessing* the variable

VARIABLES DECLARATION

var keyword and = assignment operator

Declaration:

```
var age;
```

Assignment:

```
age = 21;
```

Both at the same time:

```
var age = 21;
```

Order of operation matters! The right-hand-side of the assignment operator gets assigned to the left-hand-side.

VARIABLE RE-ASSIGNMENT

```
var name = "Jo";
name = "Amir";
```

`name` is now "Amir"

VARIABLE CONVENTIONS

- Variable names should start with a lower case letter
- If they contain multiple words, subsequent words should start with an upper case letter.

```
var numberOfStudents = 10;
```

VARIABLES & DATA TYPES

- **String** : *A stringing-together of text characters or words*

```
var myString = 'The quick brown fox';
```

- **Number** : *A (signed) numerical value with or without a decimal point (float)*

```
var myNumber = 77; var myNumber = 77.77; var myNumber = -77.77;
```

- **Boolean** : **true** *or* **false** *keywords*

```
var myBoolean = true; var myBoolean = false;
```

- **Object** : *A mapping of property names to property values*

```
var myObject = {numProp: 10, strProp: "the quick brown fox"};
```

VARIABLES & DATA TYPES

Variables are dynamic. The same variable can be used for different types:

```
var x;                  // Right now x is undefined
var x = 5;              // Look Ma I'm a Number
var x = "John";         // Now I'm a String!
```

Use the `typeof` keyword to find out the type of a variable

```
var myString = "hello";
typeof myString === "string";
```

SCORE KEEPER

MORE ON STRINGS

- Stores textual information
- String literal is surrounded by quotes

```
"How is the weather today?"

'Warm'
```

STRINGS

Double vs single quoted strings:

'They "purchased" it'

"It's a beautiful day"

STRINGS

Escaping

"They \"purchased\" it"

'It\'s a beautiful day'

CONVERSION: STRING TO NUMBER

```
var intString = "4";
var intNumber = parseInt(intString, 10);
var floatString = "3.14159";
var floatNumber = parseFloat(floatString);
```

CONVERSION: NUMBER TO STRING

```
var number = 4;
number.toString(); => "4"
```

OR

use concatenation and a number will become part of a string

```
number + ""; => "4"
```

Why would you need to convert datatypes?

MORE ON NUMBERS

Represent numerical data

Can be an integer or a float, they are both Numbers

```
var myInt = 42;

var myFloat = 3.14159265;
```

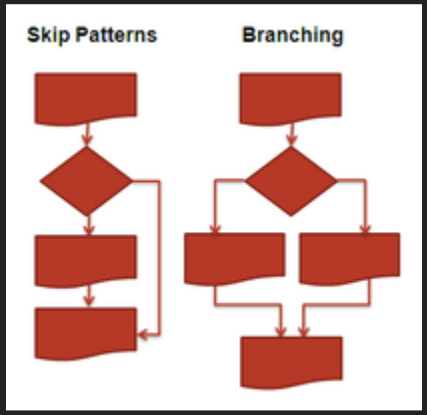
ARITHMETIC IN JAVASCRIPT

Operator	Meaning	Example
+	Addition	8 + 10
-	Subtraction	10 – 8
*	Multiplication	12 * 2
/	Division	10 / 5
%	Modulus	10 % 6

Also:

- Increment: ++ and Decrement: ––
- Use parentheses ( ) like in algebra when necessary

CONDITIONALS



Conditional statements use an `if` or `if else` block and boolean operators to determine a course of action.

MAKING DECISIONS

It's either TRUE or FALSE (like booleans)

If you are greater than 18 you are an adult

```
if (age > 18){
    document.write("You are an adult");
}
```

COMPARE THAT

COMPARISONS - EQUALITY

Are two things equal?

```
10 === 10 //true
10 === 5 //false
"hi" === "hi" //true
```

LOGICAL OPERATORS

x = 3

Logical Operators			
Operator	Description	Comparing	Returns
==	equal to	x == 8	FALSE
===	exactly equal to(value and type)	x=== “3”	FALSE
		x===3	TRUE
!=	is not equal	x!=8	TRUE
!==	is not equal(neither value nor type)	x!== “3”	TRUE
		x!==3	FALSE
>	greater than	x>8	FALSE
<	less than	x<8	TRUE
>=	greater than or equal to	x>=8	FALSE
<=	less than or equal to	x<=8	TRUE

CONDITIONAL SYNTAX

```
if(condition is true) {
    //Do cool stuff
}
```

CONDITIONAL SYNTAX

```
if(condition is true) {
    //Do cool stuff
}else{
    //Do other cool stuff
}
```

CONDITIONAL SYNTAX

```
    var topic = "JS";
    if (topic == "JS") {
        console.log("You're learning JavaScript");
    } else if(topic == "JavaScript") {
        console.log("You're still learning JavaScript");
    } else {
    console.log("You're learning something else");
}
```

MULTIPLE CONDITIONS

```
if (name == "GA"  && password == "YellowPencil"){
    //Allow access to internet
}
```

THE TRUTH TABLE

```
if (name == "GA"
&& password == "YellowPencil"){
    //Allow access to internet
}
```

THE TRUTH TABLE

AND - &&	TRUE	FALSE
TRUE	true	false
FALSE	false	false

THE TRUTH TABLE

```
if (day == "Tuesday"  || day == "Thursday"){
    //We have class today
}
```

THE TRUTH TABLE

OR - ||	TRUE	FALSE
TRUE	true	true
FALSE	true	false

JAVASCRIPT QUIRKS

Booleans are an example where you can find JavaScript quirks (unexpected behaviour).

JavaScript performs implicit type coercion on non-boolean data types, so by default non-boolean types are *truthy* or *falsy*. You can wrap a variable in `Boolean()` or prefix a double-bang `!!` to fully coerce a non-boolen value to a boolean value.

```
Boolean(0) === false;
Boolean(-1) === true;
Boolean("") === false;
Boolean({}) === true;
!!"" === false;
!!"a" === true;
```

BLACKOUT

WEATHER APPLICATION PART 1

AGENDA

- Functions
- Anonymous Functions
- Revisiting jQuery
- Weather Application

FUNCTIONS

'First-class' blocks of code that get executed (invoked)

FUNCTIONS

- Functions are first-class citizens in JavaScript. They have their own data-type, `function` they can be passed into other functions as parameters. They can be generated and returned by (e.g) factory functions.
- They can be methods of an object
- They have their own internal methods like call, apply, bind
- They have their own scope and therefore provide encapsulation
- They can create closures, an inner function that has persistent access to the outer (enclosing) function's scope

FUNCTIONS SYNTAX

The name of your function

function Keyword.
Like the “var” keyword.

```
function functionName(arg1, arg2) {
  //Body of function
}
```

FUNCTION CALLS

```
function helloWorld() {
  console.log("Hello Functions");
}

helloWorld(); //Prints “Hello Functions to the
console.
```

The brackets execute the function. Try calling the function without them to see what happens.

FUNCTION ARGUMENTS

Arguments let you pass data into the function

```
function functionName(arg1, arg2) {
  //Body of function
}
```

The functions executed code goes between the { } brackets. Much like an “if” statement.

FUNCTION ARGUMENTS

```
function addAndPrint(num1, num2) {
  var sum = num1 + num2;
  console.log(sum);
}

addAndPrint(1, 2); // Result is 3

addAndPrint(8, 2); // Result is 10
```

RETURN FUNCTIONS

These are functions that `return` something:

```
function someFunc(isTrue){
 if(isTrue){
   return true;
 }
 return false;
}
```

Some functions return other functions (factory functions):

```
function sumFactory(a) {
     return function(b){
        return a + b;
    }
}
```

CASH REGISTER

ANONYMOUS FUNCTIONS

These are functions without a defined name. They are generally used when you are only going to need a specific function once.

```
$('button').click(
 function(){
   alert('The button was clicked!')
 }
)
```

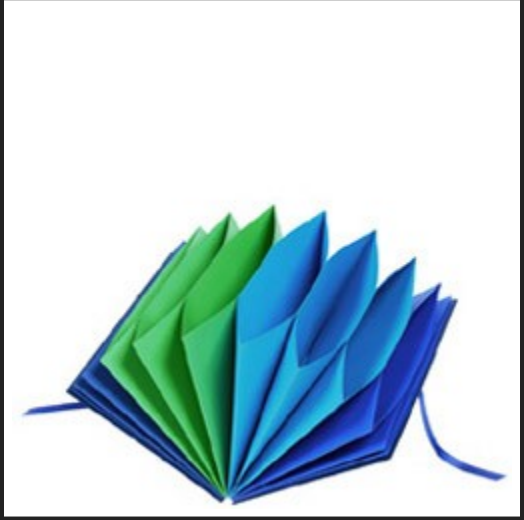
ANONYMOUS CASH REGISTER

WEATHER APPLICATION - PART 2

AGENDA

- Collection Of Data
- Manipulating Collections

ARRAYS COLLECTIONS



ARRAYS

What if we had a collection of images that we wanted to display to the screen one at a time?

How could we store all the images?

ARRAYS

An array is a list **object** with **built in methods** for things like:

- adding to the list
- removing from the list
- traversal of the list.

DECLARING ARRAYS

```
var myArr = new Array();
```

- declaring an empty array using the Array constructor.

DECLARING ARRAYS

```
var myArr = [ ];
```

- declaring an empty array using literal notation.

DECLARING ARRAYS

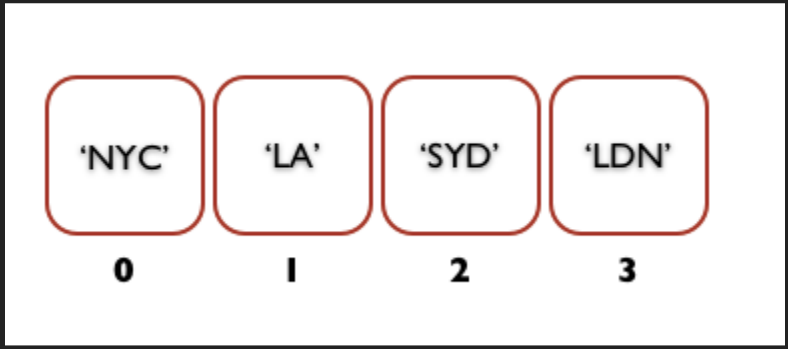
```
myArr = ['Hello', 54.3, true];
```

- Arrays are filled with elements: i.e. myArr3 = [element, anotherElement];
- Elements can contain strings, numbers, booleans, and more.

DECLARING ARRAYS

If you leave a blank spot in an array it creates a blank shelf space (undefined) placeholder.

ARRAYS INDEXING



ARRAYS INDEXING

Array elements can be fetched by their index number (starts from 0).

```
myArr = ['Hello', , 54.3, true];

console.log(myArr[0]); //prints Hello
console.log(myArr[1]); //prints undefined
console.log(myArr[2]); //prints 54.3
console.log(myArr[3]); //prints true
```

ARRAYS INDEXING

We can insert new values into any space in the array using the positions index.

```
myArr[1] = 'Stuff';
```

ARRAYS INDEXING

We can overwrite all the elements of an array simply by giving the array new values or by setting an array equal to a different array.

```
var fruits = ['Apples', 'Oranges', 'Pears', 'Bananas'];
var myArr=[1,2,3];
myArr = fruits;

console.log(myArr); //prints Apples, Oranges, Pears, Bananas
```

ARRAY LENGTH

What if I would like to know how long my array is (how many elements)?

```
console.log(myArr.length); //prints 4
```

ARRAY METHODS

The Array object has many built in methods for doing stuff with arrays. Here are two common methods:

`Array.push()` *adds an item to the end of an array*

```
var myArr = [1,2,3];
myArr.push(4); //myArr === [1,2,3,4]
```

`Array.pop()` *removes an item from the end of an array*

```
var myArr = [1,2,3,4];
var popped = myArr.pop(); //myArr === [1,2,3]; popped = 4;
```

ARRAYS EXERCISE

ITERATE OVER ARRAY

- Computers can repeatedly execute lines of code very quickly (in milliseconds and nanoseconds)
- Combined with conditions (if) computers can process large quantities of data quickly and make "intelligent" decisions based on this data.
- Sequentially processing a list of data and doing something with the data is one of the most common activities in programming.

ITERATE OVER ARRAY - REPEAT LOOPS

`for` loop:

```
for (var i = 0; i < 5; i++) {
    //i runs from 0 to 4 in this loop.
};
```

`while` loop:

```
var n = 10;
while(n--){
    console.log('n is', n); //n runs from 9 to 0
};
```

ITERATE OVER ARRAY

The `Array.forEach` method also allows you to run code using each element from the array as a value

You pass an **anonymous function** with pre-defined arguments

```
var fruits=["Banana","Apple","Pear"]
    fruits.forEach(function(element,index){
    console.log(element, "is at position", index);
});
```

`element` is the item from the array

`index` is the item's position in the array

MORE ON ARRAYS

For many more Array methods see:
https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array

CAROUSEL