FEWD - JAVASCRIPT

KARA YU

MIT, Meteor

AGENDA

- Expressions
- Conditionals
- `if` statements
- functions
- Arrays

WHY JAVASCRIPT? WE WANT HIGH PERFORMANCE WEB SITES!

- Interaction with the DOM.
- Respond to DOM events: click, submit...
- Send requests to the server.
- Act on response from the server.

INTERACTION WITH THE SERVER

- The server/backend holds the data and a lot of the logic/math for the site
- Sample requests to the server
  - get me account information for the customer named "Bob"
  - show all courses happening in July so I can so it on a page
- Sample actions after receiving information from the server
  - figure out how to show bob's info (address, picture, posts) in a way that makes sense on the page
  - figure out the format for showing courses happening in July

RESOURCES:

- Code School free course on the Developer Tools: **https://www.codeschool.com/courses/discover-devtools**
- JavaScript Alonge: **https://leanpub.com/javascript-allonge/read#leanpub-auto-a-pull-of-the-lever-prefaces**
- MDN JavaScript Reference: **https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference**
- W3Schools JavaScript Reference: **http://www.w3schools.com/jsref/default.asp**

VALUES & EXPRESSIONS

- Computers return values when you give them expressions.
- Example of expressions:

```
2 + 4
"hello" + "bob"
x + y
```

VALUES & EXPRESSIONS

- Give the computer a value and it returns a value, thus values are expressions as well.

```
// values are expressions
> 42
//=> 42

// addition is an expression
2 + 2
//=> 4

// string concatenation is an expression
"hello" + " world"
//=> "hello world"
```

CONDITIONALS:

ALWAYS USE TRIPLE EQUAL "===" OR "!=="

- Check to see if two values are identical with the "===" strict equality

```
console.log("Always use triple equal sign to test equality.")

console.log(42 === 42);

console.log(3 === "3");

console.log(3 == "3");

console.log(2 + 2 === 4);

console.log("foo" !== "bar");
```

UNDEFINED

- When something is "undefined" it has no value.
- Oddly enough "undefined" is a value.

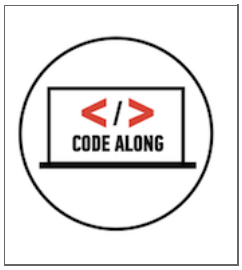
```
console.log("undefined is a value-type.");
console.log(undefined === undefined);
```

VARIABLES

- How do we create a variable?
- How do we set the value of a variable?
  - What if we don't set the value?
- When do we use them?

```
var x;
x = "hello";
x = 4;
y = x + 2;
```

EXAMPLE



http://scratch.mit.edu/projects/14085928/

IF STATEMENTS:

```
if (...) {
    //do something
}
else if (...) {
    //do something else
}
else {
    //do crazy things
}
```

FALSE

- 0, false, null, undefined, Empty String: ""

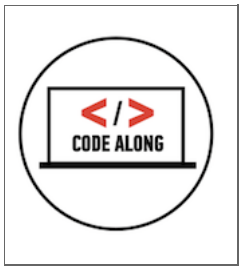
```
console.log("False Tester:");
// Try: 0, "", undefined, null, false.
if(0) {
 console.log(true);
} else {
  console.log(false);
}
```

FUNCTIONS

- What are functions?
- What can they do?
- Why would we use them?

In JS, functions are values.

SCRATCH EXAMPLE OF A FUNCTION



http://scratch.mit.edu/projects/13962187/#editor

FUNCTIONS

GENERAL SYNTAX

```
function f(...) {
    //do things
}
```

What is f? what is "..."? How do we "apply" f?

GENERAL SYNTAX

```
function doubleIt( arg1 ) {
    return arg1*2;
}

> doubleIt(2);
> var b = 5;
> doubleIt(b);
```

FUNCTIONS

TYPE-ALONG

In console:

```
> '3'
> (function () {});
```

Just like with the number `3`, functions are values!

What's the name of the function we just typed?

FUNCTIONS

WHAT CAN YOU DO WITH VALUES?

- store references to values
- perform operations on values with operators
- pass them around between methods

FUNCTIONS

How to use functions:

"Apply the function to the arguments... x, y, z"

Let's apply the function we wrote before to zero arguments.

```
> (function () {})();

> function f() {};
> f();
```

WHY DOES THE ABOVE RETURN `UNDEFINED`?

Let's dissect a function first.

- defining a function means writing the word `function`
- followed by a list of 0 or more arguments
- followed by the body of the function,
- which may have 0 or more JS statements

```
function () {

}
```

THE `RETURN` KEYWORD.

- Functions that don't have a `return` statement return `undefined`.
- What does returning something mean?
- How do we make the following return something?

```
> (function () {})();

> (function () {return "hello!";})();
```

The `return` keyword creates a return statement that immediately terminates the function application and returns the result of evaluating its expression.

LET'S DISSECT A COMPLICATED EXAMPLE

Why does applying the function below to any argument always return `true`?

```
(function (value) {
  return (function (copy) {
    return copy === value;
  })(value);
})("Hello World");

(function (value) {
  return (function (copy) {
    return copy === value;
  })(value);
})([1, 2, 3]);
```

FUNCTIONS

Let's apply a function to two arguments:

What does this do?

```
function circ(pi, radius) {
  return 2 * pi * radius
}

circ(3.14159265, 1 + 2);
```

- we are applying a function to more than one argument
- take a look at what happens to `1 + 2`. That expression is evaluated, before the function is applied to the value
- from this, we infer that functions are applied to values
- another way of saying this is that JS uses the "call by value" evaluation strategy

WHAT ARE ENVIRONMENTS?

What do people mean when they say environment?

What does this do?

```
(function (x) {
    return (function (y) {
        return x;
    })
})(4)(2);
```

WHAT ARE ENVIRONMENTS?

Let's dissect what just happened.

- The first x, the one in function (x) ..., is an argument.
- The y in function (y) ... is another argument.
- The second x, the one in { return x }, is not an argument, itâs an expression referring to a variable.
- Arguments and variables work the same way whether we're talking about function (x) { return (function (y) { return x }) } or just plain function (x) { return x }.

WHAT'S IMPORTANT IS THAT EVERY TIME A FUNCTION IS INVOKED, A NEW *ENVIRONMENT* IS CREATED

WHAT ARE ENVIRONMENTS?

- An environment is a possibly empty dictionary that maps variables to values by name.
- How does the value get put in the environment? Well for arguments, that is very simple. When you apply the function to the arguments, an entry is placed in the dictionary for each argument.

So when we write this, what happens?

```
(function (x) { return x })(2)
  //=> 2
```

FREE VARIABLES AND ENVIRONMENTS

What we arrive at is the idea that a function always has a reference to its immediate parent environment.

Let's take a look at the environments created by each function in our example from the exercises:

```
(function (x) {
    // { x: 4, '..': global environment }
    return (function (y) {
        // { y: 2, '..': { x: 4 } }
        return x;
    })
})(4)(2);
```

Notice the double ellipses... they reference one level up. It's kind of like `..` in the terminal.

FUNCTION NAMING

VAR

What does the `var` keyword do? What can it store?

It can store references to functions too!

```
var circumference = function (pi, radius) {
    return 2 * pi * radius;
};

// Let's apply the function to two arguments:
circumference(3.14, 2);

//Let's try one more thing:
circumference.name;

//what does this do?
circumference = 5;
circumference(3.14, 2);
```

NAMED FUNCTION EXPRESSIONS VS FUNCTION DECLARATIONS

Let's implement a named function. What is circumference in this case?

```
var c = function circumference (radius) {
    var Pi = 3.14;
    return 2 * pi * radius;
};

//try
> circumference(2);
> c(2);
> c.name;
```

What's going on???

-

FUNCTION DECLARATIONS

We have indeed created a named function, whose name is circumference, we've not actually *declared* a function. Instead, what we had done was that we wrote a named function expression.

An actual function declaration:

```
function circumference (radius) {
    var Pi = 3.14;
    return 2 * pi * radius;
}

//try
> circumference(2);
> c(2);
> c.cname;
```

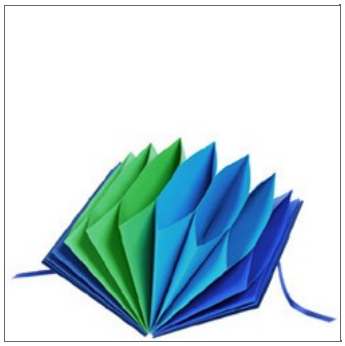
FUNCTION REVIEW

- Functions are a way to group a series of tasks to perform
- In JS, functions are values
- You can store a reference to a function in a variable

Syntax:

```
function f (...) {
    //do things
    //return a result;
}
```

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

What is an array?

I think about it as a line of buckets you have in a row, kinda like a week-long pill box

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

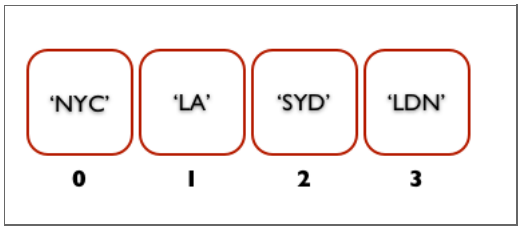
myArr = ['Hello', 54.3, true];

- 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.

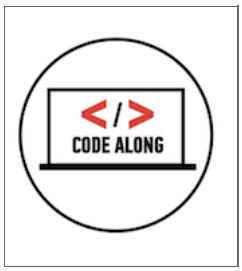
```
var fruits = ['Apples', 'Oranges', 'Pears', 'Bananas'];
var myArr=[1,2,3];
myArr = fruits;

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
```



ARRAYS

ITERATE OVER ARRAY

FOR LOOP

```
var mixed = [1, "two", "three", true];

console.log("For Loop:");

// Most common mistake is using commas instead of "semicolons" inside the loop declaration.

for(var i = 0; i < mixed.length; i++) {
    console.log("The element at index " + i + " is: " + mixed[i]);
}
```

ITERATE OVER ARRAY

WHILE LOOP

```
var a = [1,2,3,4];
var b = [1,2,3,4];

console.log("While loop:");

var i = 0;

while(i < a.length)
{
    console.log("The element at index " + i + " is: " + a[i]);
    i++;
}
```

ITERATE OVER ARRAY

Allows you to run code using each element from the array as a value Syntax:

```
Array.forEach
```

What do these words mean: `element`, `index` ?

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

MORE ON ARRAYS

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

COMPARING REFERENCE TYPES:

EVEN IF THEY HAVE THE SAME VALUES AND ARE THE SAME TYPE, REFERENCE TYPES ARE NOT STRICTLY EQUAL.

- Arrays are unique structures.

```
console.log("Arrays are reference-type data structures.")
console.log([1,2,3] === [ 2-1, 1+1, 2+1]);
console.log([1,2,3] === [1,2,3]);
```

COMBINING FUNCTIONS AND ARRAYS

```
// when applied to a value referencing an array this function
// logs every element in the array to the console
(function (array) {
  for (var i = 0; i < array.length; i++) {
    console.log(array[i]);
  }
})([1,2,3]);
```

alternatively:

```
function f(array) {
  array.forEach(function(element) {
    console.log(element);
  });
})([1,2,3]);
```

Implement a routine that checks to see if two arrays are identical. Print "true" to the console if they're equals, "false" if they are unequal

DIVIDED TIMES