

JavaScript: Arrays & Objects

**The
Complete
Web
Developer in
2019**

The Complete Web Developer in 2019
Zero to Mastery
Andrei Neagoie
Lecture Notes by Stephanie

Data Structures: Arrays

Array of strings

```
> var animalList = ["tiger", "cat", "bear", "bird"]
```

```
> animalList
< ▼ (4) ["tiger", "cat", "bear", "bird"] ⓘ
  0: "tiger"
  1: "cat"
  2: "bear"
  3: "bird"
  length: 4
  ▶ __proto__: Array(0)
```

```
> animalList[0]
```

```
< "tiger"
```

```
> animalList[2]
```

```
< "bear"
```

Arrays can hold anything..

Arrays can hold:

strings

numbers

booleans

functions

other arrays

etc.

Arrays can have mixed types too, although not advised

Array of functions

```
> var functionList = [checkDriverAge, checkDriverAge2]
< undefined
> functionList
< ▼ (2) [f, f] ⓘ
  ▶ 0: f checkDriverAge(age)
  ▶ 1: f ()
    length: 2
  ▶ __proto__: Array(0)
```

```
> functionList[0]
< f checkDriverAge(age) {
  if (age < 18) {
    return "too young";
  } else if (age > 18) {
    return "you may drive";
  } else if (age === 18) {
    return "happy 18th bday";
  }
}
```

Can even declare a function within an array

```
> var functionList = [function apple() {
  console.log("apple");
}]
< undefined
> functionList
< ▼ [f] ⓘ
  ▼ 0: f apple()
    arguments: null
    caller: null
    length: 0
    name: "apple"
    ▶ prototype: {constructor: f}
    ▶ __proto__: f ()
      [[FunctionLocation]]: VM9521:1
```

Can have multiple types within one array (not advised, can lead to performance issues)

```
> var mixedList = ["apples", 3, undefined, true, function  
  apple() {  
    console.log("apples")  
  }];  
< undefined  
  
> mixedList  
< ▼ (5) ["apples", 3, undefined, true, f] ⓘ  
  0: "apples"  
  1: 3  
  2: undefined  
  3: true  
  4: f apple()  
  length: 5  
  ▶ __proto__: Array(0)
```

Array containing arrays

```
> var animalList = [ ["tiger", "cat"], ["bear", "bird"] ];
```

```
> animalList  
< ▶ (2) [Array(2), Array(2)]  
  
> animalList[1]  
< ▶ (2) ["bear", "bird"]
```

To reach item [0] of array [1] in animalList

```
> animalList[1][0]  
< "bear"
```

Array Methods

Javascript has predefined methods to use with arrays - we can think of them as functions we use on arrays

```
> var animalList = ["tiger", "cat", "bear", "bird"];  
< undefined  
> animalList  
< ▶ (4) ["tiger", "cat", "bear", "bird"]
```

.shift() - delete first item of array

.pop() - delete last item of array

```
> animalList.shift()  
< "tiger"  
> animalList  
< ▶ (3) ["cat", "bear", "bird"]
```

.push() - add item to end of array

```
> animalList.pop()  
< "bird"  
> animalList  
< ▶ (2) ["cat", "bear"]
```

```
> animalList.push("elephant")  
< 3  
> animalList  
< ▶ (3) ["cat", "bear", "elephant"]
```

.concat() - returns multiple arrays joined together (does not change original arrays)

```
> animalList
< ▶ (3) ["cat", "bear", "elephant"]
> animalList.concat(["bee", "deer"])
< ▶ (5) ["cat", "bear", "elephant", "bee", "deer"]
> animalList
< ▶ (3) ["cat", "bear", "elephant"]
```

can save concatenated array:

```
> animalList
< ▶ (3) ["cat", "bear", "elephant"]
> newAnimalList = animalList.concat(["bee", "deer"])
< ▶ (5) ["cat", "bear", "elephant", "bee", "deer"]
> newAnimalList
< ▶ (5) ["cat", "bear", "elephant", "bee", "deer"]
> animalList
< ▶ (3) ["cat", "bear", "elephant"]
```

.sort() - sorts array

```
> animalList
< ▶ (3) ["cat", "bear", "elephant"]
```

```
> animalList.sort()
< ▶ (3) ["bear", "cat", "elephant"]
> animalList
< ▶ (3) ["bear", "cat", "elephant"]
```

.reverse() - reverses array

```
> animalList  
< ▶ (3) ["bear", "cat", "elephant"]
```

```
> animalList.reverse()  
< ▶ (3) ["elephant", "cat", "bear"]  
  
> animalList  
< ▶ (3) ["elephant", "cat", "bear"]
```

.splice() - add/removes elements from array

```
> fruitArray = ["Apples", "Blueberries", "Oranges", "Kiwi"]  
< ▶ (4) ["Apples", "Blueberries", "Oranges", "Kiwi"]
```

Remove 1 element from array at index 2

```
> fruitArray.splice(2,1)  
< ▶ ["Oranges"]  
  
> fruitArray  
< ▶ (3) ["Apples", "Blueberries", "Kiwi"]
```

Add element at index 2

```
> fruitArray.splice(2,0,"Pineapple")  
< ▶ []  
  
> fruitArray  
< ▶ (4) ["Apples", "Blueberries", "Pineapple", "Kiwi"]
```

Delete 2 elements at index 1, add 3 elements at index 1

```
> fruitArray.splice(1,2,"Mango", "Lemon", "Lime")  
< ▶ (2) ["Blueberries", "Pineapple"]  
  
> fruitArray  
< ▶ (5) ["Apples", "Mango", "Lemon", "Lime", "Kiwi"]
```

JavaScript Array Reference

[< Previous](#)[Next >](#)

Array Object

The Array object is used to store multiple values in a single variable:

```
var cars = ["Saab", "Volvo", "BMW"];
```

[Try it Yourself »](#)

Array indexes are zero-based: The first element in the array is 0, the second is 1, and so on.

For a tutorial about Arrays, read our [JavaScript Array Tutorial](#).

Array Properties

| Property | Description |
|-----------------------------|--|
| constructor | Returns the function that created the Array object's prototype |
| length | Sets or returns the number of elements in an array |
| prototype | Allows you to add properties and methods to an Array object |

Array Methods

| Method | Description |
|------------------------------|---|
| concat() | Joins two or more arrays, and returns a copy of the joined arrays |
| copyWithin() | Copies array elements within the array, to and from specified positions |
| entries() | Returns a key/value pair Array Iteration Object |
| every() | Checks if every element in an array pass a test |
| fill() | Fill the elements in an array with a static value |

| | |
|----------------------|--|
| <u>filter()</u> | Creates a new array with every element in an array that pass a test |
| <u>find()</u> | Returns the value of the first element in an array that pass a test |
| <u>findIndex()</u> | Returns the index of the first element in an array that pass a test |
| <u>forEach()</u> | Calls a function for each array element |
| <u>from()</u> | Creates an array from an object |
| <u>includes()</u> | Check if an array contains the specified element |
| <u>indexOf()</u> | Search the array for an element and returns its position |
| <u>isArray()</u> | Checks whether an object is an array |
| <u>join()</u> | Joins all elements of an array into a string |
| <u>keys()</u> | Returns a Array Iteration Object, containing the keys of the original array |
| <u>lastIndexOf()</u> | Search the array for an element, starting at the end, and returns its position |
| <u>map()</u> | Creates a new array with the result of calling a function for each array element |
| <u>pop()</u> | Removes the last element of an array, and returns that element |
| <u>push()</u> | Adds new elements to the end of an array, and returns the new length |
| <u>reduce()</u> | Reduce the values of an array to a single value (going left-to-right) |
| <u>reduceRight()</u> | Reduce the values of an array to a single value (going right-to-left) |
| <u>reverse()</u> | Reverses the order of the elements in an array |
| <u>shift()</u> | Removes the first element of an array, and returns that element |
| <u>slice()</u> | Selects a part of an array, and returns the new array |
| <u>some()</u> | Checks if any of the elements in an array pass a test |
| <u>sort()</u> | Sorts the elements of an array |
| <u>splice()</u> | Adds/Removes elements from an array |
| <u>toString()</u> | Converts an array to a string, and returns the result |
| <u>unshift()</u> | Adds new elements to the beginning of an array, and returns the new length |
| <u>valueOf()</u> | Returns the primitive value of an array |

[< Previous](#)
[Next >](#)

// Exercise 6

// var array = ["Banana", "Apples", "Oranges", "Blueberries"];

```
> var array = ["Banana", "Apples", "Oranges",  
  "Blueberries"];  
< undefined
```

// 1. Remove the Banana from the array.

```
> array.shift()  
< "Banana"  
> array  
< ▶ (3) ["Apples", "Oranges", "Blueberries"]
```

// 2. Sort the array in order.

```
> array.sort()  
< ▶ (3) ["Apples", "Blueberries", "Oranges"]  
> array  
< ▶ (3) ["Apples", "Blueberries", "Oranges"]
```

// 3. Put "Kiwi" at the end of the array.

```
> array.push("Kiwi")  
< 4  
> array  
< ▶ (4) ["Apples", "Blueberries", "Oranges", "Kiwi"]
```

// 4. Remove "Apples" from the array.

```
> array.splice(0,1)  
< ▶ ["Apples"]  
> array  
< ▶ (3) ["Blueberries", "Oranges", "Kiwi"]
```

// 5. Sort the array in reverse order.
(Not alphabetical, but reverse
// the current Array i.e. ['a', 'c', 'b']
becomes ['b', 'c', 'a'])

```
> array.reverse()  
< ▶ (3) ["Kiwi", "Oranges", "Blueberries"]
```

// using this array,

// var array2 = ["Banana", ["Apples", ["Oranges", "Blueberries"]]];

// access "Oranges".

```
> var array2 = ["Banana", ["Apples", ["Oranges",  
  "Blueberries"]]];  
< undefined  
> array2[1][1][0]  
< "Oranges"
```

Data Structures: Objects Also a Javascript Type

Object - collection of properties + values

```
> var userObject = {  
  name: "John",  
  age: 34,  
  hobby: "Crossfit",  
  isMarried: false,  
}
```

```
> userObject  
< {name: "John", age: 34, hobby: "Crossfit", isMarried: false}  
  age: 34  
  hobby: "Crossfit"  
  isMarried: false  
  name: "John"  
  __proto__: Object
```

```
> userObject.name  
< "John"
```

Object { } vs array []:

```
> fruitArray  
< ▶ (5) ["Apples", "Mango", "Lemon", "Lime", "Kiwi"]
```

```
> fruitArray[0]  
< "Apples"
```

Object: property "name" holds value "John" at userObject.name
Array: index [0] holds value "Apples" at fruitArray[0]

To add property to object, just declare it

```
> userObject.favoriteFood = "spinach";  
< "spinach"
```

```
> userObject  
< {name: "John", age: 34, hobby: "Crossfit", isMarried: false, favoriteFood: "spinach"}  
  age: 34  
  favoriteFood: "spinach"  
  hobby: "Crossfit"  
  isMarried: false  
  name: "John"  
  __proto__: Object
```

Change value of a property the same way

```
> userObject.hobby = "soccer"  
< "soccer"
```

```
> userObject  
< {name: "John", age: 34, hobby: "soccer", isMarried: false, favoriteFood: "spinach"}  
  age: 34  
  favoriteFood: "spinach"  
  hobby: "soccer"  
  isMarried: false  
  name: "John"  
  __proto__: Object
```

How come is “object” a javascript type, but “array” isn’t?

An array is a kind of object (uses indices as properties)

Example: **ARRAY []** within **OBJECT { }**

```
> userObject.quotes = ["its lit", "sup bro", "damn girl"]
```

```
> userObject
< {name: "John", age: 34, hobby: "soccer", isMarried: false, favoriteFood: "spinach", ...} ⓘ
  age: 34
  favoriteFood: "spinach"
  hobby: "soccer"
  isMarried: false
  name: "John"
  quotes: Array(3)
    0: "its lit"
    1: "sup bro"
    2: "damn girl"
    length: 3
    __proto__: Array(0)
  __proto__: Object
```

To access array:

```
> userObject.quotes
< (3) ["its lit", "sup bro", "damn girl"] ⓘ
  0: "its lit"
  1: "sup bro"
  2: "damn girl"
  length: 3
  __proto__: Array(0)
```

To access value:

```
> userObject.quotes[1]
< "sup bro"
```

Example: **OBJECT { }** within **ARRAY []**

```
> var listArray = [  
  {  
    username: "andy",  
    password: "secret",  
  },  
  {  
    username: "jess",  
    password: "123",  
  }  
];
```

```
> listArray  
< ▼ (2) [{...}, {...}] ⓘ  
  ▼ 0:  
    password: "secret"  
    username: "andy"  
    ► __proto__: Object  
  ▼ 1:  
    password: "123"  
    username: "jess"  
    ► __proto__: Object  
    length: 2  
    ► __proto__: Array(0)
```

To access object:

```
> listArray[0]  
< ▼ {username: "andy", password: "secret"} ⓘ  
  password: "secret"  
  username: "andy"  
  ► __proto__: Object
```

To access value:

```
> listArray[0].password  
< "secret"
```

METHOD: FUNCTION within OBJECT (or array)

```
> userObject.shout = function() {  
    console.log("AHHHHH!");  
}  
< f () {  
    console.log("AHHHHH!");  
}
```

```
> userObject  
< {name: "John", age: 34, hobby: "soccer", isMarried: false,  
  favoriteFood: "spinach", ...} ⓘ  
  age: 34  
  favoriteFood: "spinach"  
  hobby: "soccer"  
  isMarried: false  
  name: "John"  
  ▶ quotes: (3) ["its lit", "sup bro", "damn girl"]  
  ▶ shout: f ()  
  ▶ __proto__: Object
```

To see function code:

```
> userObject.shout  
< f () {  
    console.log("AHHHHH!");  
}
```

Use () to call function

We would say shout is a method of userObject

```
> userObject.shout()
```

AHHHHH!

pathturbo.js:1

We would say that for those, concat, sort, etc are methods of listArray or any array we create

Also, look at `console.log()`... `console` is an object and `log` is one of many methods

```
> console
< ▾ console {debug: f, error: f, info: f, log: f, warn: f, ...} ⓘ
  ▶ assert: f assert()
  ▶ clear: f clear()
  ▶ context: f context()
  ▶ count: f count()
  ▶ countReset: f countReset()
  ▶ debug: f ()
  ▶ dir: f dir()
  ▶ dirxml: f dirxml()
  ▶ error: f ()
  ▶ exception: f ()
  ▶ group: f group()
  ▶ groupCollapsed: f groupCollapsed()
  ▶ groupEnd: f groupEnd()
  ▶ info: f ()
  ▶ log: f ()
  ▶ markTimeline: f {}
  ▶ read: f ()
  ▶ time: f ()
  ▶ timeLog: f timeLog()
  ▶ timeEnd: f timeEnd()
  ▶ timeLogEnd: f timeLogEnd()
  ▶ timeLogStart: f timeLogStart()
  ▶ timeLogEnd: f timeLogEnd()
  ▶ timeLogStart: f timeLogStart()
```

Lets try.. `console.info()`

```
> console.info("hello");  
hello
```

pathturbo.js:1

```
console.error()
```

```
> console.error("welcome to fail");
```

❌ ► welcome to fail

pathturbo.js:1

When we declare a variable and it's empty, it's *undefined*

When we create an empty object or array, it's empty but NOT undefined, although its properties/indices are undefined.

Create empty object

```
> userObject2 = {};  
< ▶ {}
```

```
> userObject2.fakeproperty  
< undefined
```

Create empty array

```
> listArray2 = [];  
< ▶ []
```

```
> listArray2[0]  
< undefined
```

Null is the 5th Javascript Data Type

Null and Undefined are 2 different data types!
Properties are NOT undefined, they return an error!

Create NULL object

```
> nullObject = null;  
< null
```

```
> nullObject.fakeproperty  
✖ ▶ Uncaught TypeError: Cannot read property 'fakeproperty' of null  
   at <anonymous>:1:12 VM273576:1
```

We can add properties to an empty object/array...

```
> userObject2.fakeproperty = "andy"
< "andy"
> userObject2
< ▶ {fakeproperty: "andy"}
```

but cannot add properties to a null object...

```
> nullObject.fakeproperty = "andy"
✖ ▶ Uncaught TypeError: Cannot set property 'fakeproperty' of null
  at <anonymous>:1:25 VM276207:1
```

// Exercise 7

// Create an object and an array which we will use in our facebook exercise.

// 1. Create an object that has properties "username" and "password". Fill those values in with strings.

```
> var objectFB = {  
  username: "stephy",  
  password: "abc1"  
}
```

// 2. Create an array which contains the object you have made above and name the array "database".

```
> var databaseArray = [objectFB];
```

// 3. Create an array called "newsfeed" which contains 3 objects with properties "username" and "timeline".

```
> var newsfeedArray = [  
  {  
    username: "stephma",  
    timeline: "qwerty"  
  },  
  {  
    username: "andylool",  
    timeline: "ijk"  
  },  
  {  
    username: "sallyki",  
    timeline: "chai latte time!"  
  }  
];
```

Exercise: Build Facebook

```
> var databaseArray = [
  {
    username: "andrei",
    password: "supersecret"
  }
];

var newsfeedArray = [
  {
    username: "stephMa",
    timeline: "Qwerty is the best."
  },
  {
    username: "andyLol",
    timeline: "ijk vs xyz?"
  },
  {
    username: "Sally Ki",
    timeline: "Chai latte time!"
  }
];

var usernamePrompt = prompt("What's your username?");
var passwordPrompt = prompt("What's your password?");

function signInFB(user, pw) {
  if (user === databaseArray[0].username
    && pw === databaseArray[0].password) {
    console.log(newsfeedArray);
  } else {
    alert ("Wrong username/password");
  }
}

signInFB(usernamePrompt, passwordPrompt);
```

If you enter correct username and password...

www.litter-robot.com says

What's your username?

andrei

www.litter-robot.com says

What's your password?

supersecret

Cancel OK

....you get the timeline in the console:

```
pathturbo.js:1
▼ (3) [{...}, {...}, {...}] ⓘ
  ► 0: {username: "stephMa", timeline: "Qwerty is the best."}
  ► 1: {username: "andyLol", timeline: "ijk vs xyz?"}
  ► 2: {username: "Sally Ki", timeline: "Chai latte time!"}
    length: 3
  ► __proto__: Array(0)
```

If incorrect, get message:

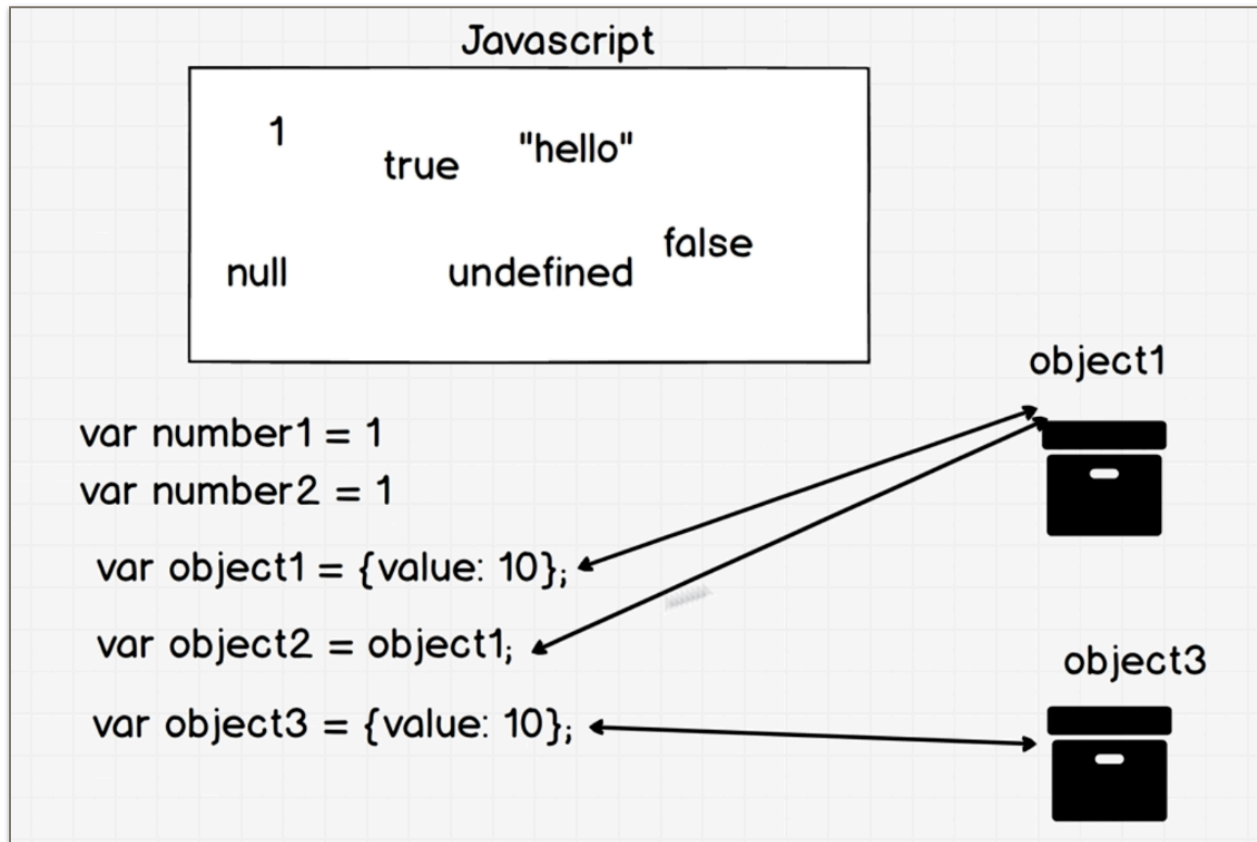
www.litter-robot.com says

Wrong username/password

OK

Advanced JavaScript Objects

Reference Type vs. Primitive Type



Primitive type - defined by the programming language, pass by value
Ex: number, null, undefined, boolean, string, symbol, etc.

Reference type - defined by the programmer, pass by reference
Ex: object, array

To clone an object (break reference): `let obj2 = Object.assign({}, obj1);`

To clone an array (break reference): `var array2 = [].concat(array1);`

Array does not equal array (because it's a reference type)

```
[] === [];
```

```
< false
```

Let's define some objects...

```
var object1 = { value: 10 };  
var object2 = object1;  
var object3 = { value: 10 };
```

Check it out:

```
object1 === object2; // reference same object
```

```
< true
```

```
object1 === object3; // reference different objs
```

```
< false
```

```
object2.value;
```

```
< 10
```

```
object1.value = 15;
```

```
object2.value; // changes automatically!
```

```
< 15
```

```
object3.value; // unaffected by change
```

```
< 10
```

Context

Context - tells you where you are within the object

this - tells us what object environment we are in right now

By default, we are in the window object...

```
console.log(this)
```

```
VM4655:1  
▶ Window {postMessage: f, blur: f, focus: f, close: f, parent: Windo  
w, ...}
```

```
this === window;
```

```
< true
```

```
this.alert("hello")
```

github.com says

hello

OK

```
function a() {  
  console.log(this);  
}
```

```
> a()
```

```
VM4682:2  
▶ Window {postMessage: f, blur: f, focus: f, close: f, parent: Windo  
w, ...}
```

```
> a() === window.a()
```

```
< true
```


What if we are NOT in the window object? (function in object4)

```
const object4 = {  
  newFunxn: function() {  
    console.log(this);  
  },  
  sample: "sample text"  
}
```

Let's run the function:

```
object4.newFunxn();
```

```
► {newFunxn: f, sample: "sample text"}
```

VM4793:3

It shows us that we are in object4.

Instantiation

OOP - class inheritance, instances, etc.

Classical Inheritance

```
var Player = function(name, type) {
  console.log('player', this);
  this.name = name;
  this.type = type;
}

Player.prototype.introduce = function() {
  console.log(`Hi, I am ${this.name}, I'm a ${this.type}`);
}

var wizard1 = new Player('Shelly', 'healer');
var wizard2 = new Player('Shawn', 'magic');

wizard1.play = function() {
  console.log(`WELL I'm a ${this.type}`);
}

wizard2.play = function() {
  console.log(`WELL I'm a ${this.type}`);
}

wizard1.play();
wizard1.introduce();

wizard2.play();
wizard2.introduce();
```

Works fine in all browsers except the template strings. From BABEL:

```
15 wizard1.play = function () {
16   console.log("WELL I'm a ".concat(this.type));
17 };
18
19 wizard2.play = function () {
20   console.log("WELL I'm a ".concat(this.type));
21 };
```

Inheritance in ES5/ES6 ONLY:

```
class Player {
  constructor(name, type) {
    console.log('player', this);
    this.name = name; // creates these properties when
constructing instance of Player object
    this.type = type;
  }
  introduce() { // create new method
    console.log(`Hi, I am ${this.name}, I'm a ${this.type}`)
  }
}

class Wizard extends Player { // adds on top of Player class
  constructor(name, type) {
    super(name, type) // accesses name/type from Player
  }
  console.log('wizard', this);
}
play() {
  console.log(`WELL I'm a ${this.type}`);
}

const wizard1 = new Wizard ('Shelly', 'Healer');
const wizard2 = new Wizard ('Shawn', 'Magic');

wizard1.play();
wizard1.introduce();

wizard2.play();
wizard2.introduce();
```

| | |
|--|---------------------------|
| player ▶ Wizard {} | VM4866:3 |
| wizard ▶ Wizard {name: "Shelly", type: "Healer"} | VM4866:15 |
| player ▶ Wizard {} | VM4866:3 |
| wizard ▶ Wizard {name: "Shawn", type: "Magic"} | VM4866:15 |
| WELL I'm a Healer | VM4866:18 |
| Hi, I am Shelly, I'm a Healer | VM4866:8 |
| WELL I'm a Magic | VM4866:18 |
| Hi, I am Shawn, I'm a Magic | VM4866:8 |

Snippet from BABEL for previous code... whole thing is 78 lines:

```
3 function _typeof(obj) { if (typeof Symbol === "function" && typeof
  Symbol.iterator === "symbol") { _typeof = function _typeof(obj) {
    return typeof obj; }; } else { _typeof = function _typeof(obj) {
    return obj && typeof Symbol === "function" && obj.constructor ===
    Symbol && obj !== Symbol.prototype ? "symbol" : typeof obj; }; }
  return _typeof(obj); }
4
5 function _possibleConstructorReturn(self, call) { if (call &&
  (_typeof(call) === "object" || typeof call === "function")) { return
  call; } return _assertThisInitialized(self); }
6
7 function _getPrototypeOf(o) { _getPrototypeOf = Object.setPrototypeOf
  ? Object.getPrototypeOf : function _getPrototypeOf(o) { return
  o.__proto__ || Object.getPrototypeOf(o); }; return _getPrototypeOf(o);
  }
8
9 function _assertThisInitialized(self) { if (self === void 0) { throw
  new ReferenceError("this hasn't been initialized - super() hasn't been
```

Exercise: Advanced Objects

Section 13, Lecture 147

It's time to code some javascript! Get your sublime text ready for this exercise, and use Google Chrome javascript console to test your code. You can find the exercise file and the solution file attached. Good luck!

```
//Evaluate these:  
//#1  
[2] === [2] // false  
{ } === { } // false
```

```
//#2 what is the value of property a for each object.  
const object1 = { a: 5 };  
const object2 = object1;  
const object3 = object2;  
const object4 = { a: 5};  
object1.a = 4;  
  
//object1.a = 4;  
//object2.a = 4;  
//object3.a = 4;  
//object4.a = 5;
```

```
//#3 create two classes: an Animal class and a Mammal class.  
// create a cow that accepts a name, type and color and has  
// a sound method that moo's her name, type and color.
```

```
class Animal {  
  constructor(name, type, color) {  
    this.name = name;  
    this.type = type;  
    this.color = color;  
  }  
}  
  
class Mammal extends Animal {  
  constructor(name, type, color) {  
    super(name, type, color)  
  }  
  introduce() {  
    console.log(`Hi, I'm a ${this.color} ${this.type}, I'm  
    ${this.name}`)  
  }  
}  
  
var cow1 = new Mammal ('Bea', 'cow', 'brown');  
  
cow1.introduce();
```