DANIEL GRAHAM PHD

JAVASCRIPT OBJECTS

JAVASCRIPT SYNTAX OBJECT

- Objects are containers for properties
  - Where properties have name and value
  - Values can be any javascript type except undefined.
    - This means that values can even be other objects.

JAVASCRIPT OBJECT SYNTAX

```
var empty_object = {}
```

```
var student ={
    "first name": "Daniel",
    "last name": "Graham",
    title: "PHD",
}
```

IT IS OK TO INCLUDE
OR EXCLUDE TRAILING COMMAS

JAVASCRIPT SYNTAX OBJECTS

- Object can even contain other objects

```
var teacher ={
    name: "Daniel Graham",
    student: {
        age: 21,
        name: "John Stewart",
        gade: 86.2,
    }
}
```

JSON (JAVASCRIPT OBJECT NOTATION)

JAVASCRIPT OBJECT RETRIEVAL

```
console.log(student["first name"])

console.log(student.title)

console.log(teacher.student.name)
```

JAVASCRIPT SYNTAX OBJECTS

- What happens if you reference a property that is apart of the object

```
console.log(teacher.salary)
```

THROWS AN UNDEFINED ERROR

```
console.log(teacher.salary.raise)
```

```
console.log(teacher.salary.raise)
                           ^

TypeError: Cannot read property 'raise' of undefined
    at Object.&lt;anonymous&gt; (/Users/dgg6b/Documents/Classes/MobileApplicationDevel
opment/00-Lecture/Code/objects.js:26:28)
    at Module._compile (internal/modules/cjs/loader.js:721:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:732:10)
    at Module.load (internal/modules/cjs/loader.js:620:32)
```

JAVASCRIPT SYNTAX OBJECTS

- Updating properties: Properties can be updated by assignment.

```
teacher['name'] = 'Daniel Graham PhD'
```

SINGLE QUOTES INTERCHANGEABLE WITH DOUBLE QUOTES

- New properties can also be added.

```
teacher['purpose'] = 'Thinking &  Creating'

teacher.affiliation = 'UVA'
```

BRACKET AND DOT NOTION ARE EQUIVALENT

JAVASCRIPT SYNTAX OBJECTS

- It is also possible to updated nested objects

```
teacher.student = {
    age: 22,
    grade: 97.2,
    year: 2,
    name: "John Stewart"
}
```

```
teacher.student.rank = 1
```

JAVASCRIPT SYNTAX OBJECTS

- Objects are passed by reference not value. Objects are never copied.

```
person1 = {
    name: 'John',
    age: 32
}
```

WHAT GETS PRINTED?

```
person2 = person1
person2.name = 'Tom'

console.log(person1.name)
```

TOM

JOHN

JAVASCRIPT OBJECT SYNTAX

```
/** Copies all the the properties from person 1 into person3
**/
person3 = Object.assign({}, person1)

person3.name = 'Jill'
```

DOES A SWALLOW COPY

JAVASCRIPT OBJECT DEEPCOPY

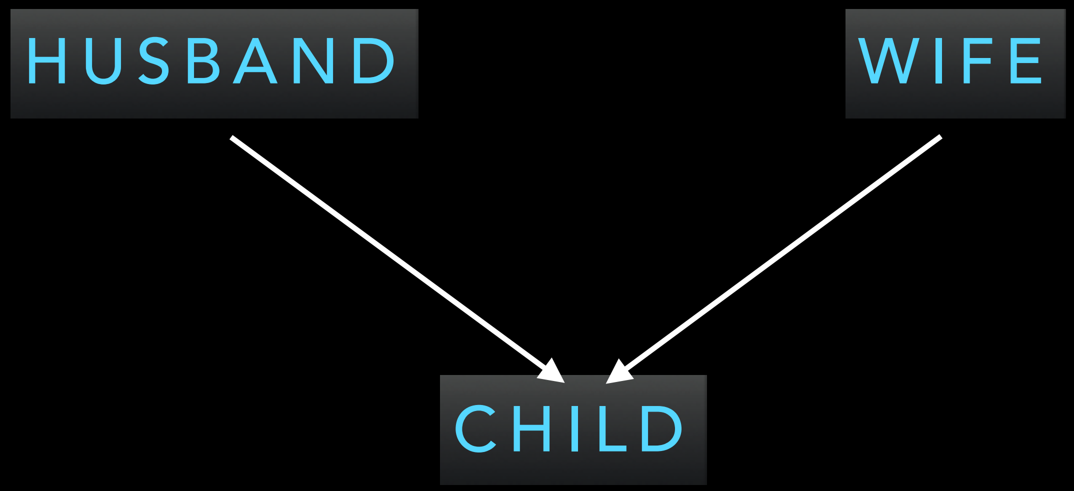
WHAT IS THE RESULT OF THIS

```
husband = {
    name: 'Daniel',
    child: {
        name: 'Ruth',
        sex: 'Female',
        age: 1,
    }
}

//Sallow Copy
wife = Object.assign({}, husband)
wife.name = 'Shea'
wife.child.name = 'George'

console.log(husband.child.name)
```

RUTH

GEORGE

JAVASCRIPT OBJECT DEEPCOPY



Sallow copies only copy references to nested objects

JAVASCRIPT OBJECT DEEPCOPY

- Correct way to deep copy in javascript

```
wife = JSON.parse(JSON.stringify(husband));
```

Write your own

Part of Project 0 homework

JAVASCRIPT SYNTAX OBJECTS

- Javascript is a prototype based language (aka Classless) while Java or C++ are class based languages

- In Class-based languages new methods cannot be added at run-time or when the object is declared

- Each object in Javascript is created from a prototypical object (Object.prototype). This means that methods can be added or removed a run-time.

"Objects inherit from objects. What could be more object oriented than that?"

–DOUGLAS CROCKFORD

JAVASCRIPT SYNTAX OBJECTS

- Why are there no constructors?
- It is like the objects are created *ex nihilo* ("from nothing")
- It javascript there is a root object (Object.prototype) which comes standard with javascript. Objects are recreated by inheriting from Object.prototype

JAVASCRIPT SYNTAX OBJECTS

- The {...} notation creates a new object by inheriting from Object.prototype standard Javascript object.

```
superClass ={
    language: "Javascript",
    version: 2.1,
}

subClass = {
    fork: "UVAscript",
}

Object.setPrototypeOf(subClass, superClass)

console.log(subClass.fork + ' ' + subClass.language)
```

LANGUAGE IS NOT A PROPERTY OF THE SUBCLASS

PRINTS: UVASCRIPT JAVASCRIPT

JAVASCRIPT SYNTAX OBJECTS

```
/**
 * Demonstrate the class hierarchy
 */
oneUp = Object.getPrototypeOf(subClass)
oneUpList = Object.getOwnPropertyNames(oneUp)
console.log('One Up List\n’ + oneUpList)

 //Get the prototype of superClass
 twoUp = Object.getPrototypeOf(oneUp)
 twoUpList = Object.getOwnPropertyNames(twoUp)
 console.log('Two Up List\n’+ twoUpList)
```

ROOT CLASS
OBJECT.PROTOTYPE

SUPER CLASS

SUB CLASS

NOTE THE ROOT CLASS
CONTAINS THE CONSTRUCTOR

JAVASCRIPT SYNTAX OBJECTS

- Enumerating of the properties

```
/** Enumerating all the properties of the subClass **/

for (prop in subClass)
{
    console.log(prop)
}
```

Prints:

fork

language

version

INCLUDES THE PROPERTIES OF SUPERCLASS

There are no guarantees on the order

JAVASCRIPT DELETING PROPERTIES

- The delete operator can be used to delete properties from an object.

```
delete subClass.fork
```

DELETES FORK PROPERTY FROM SUBCLASS

```
delete subClass.language
```

DOES NOTHING BECAUSE LANGUAGE IS A PROPERTY OF SUPERCLASS

JAVASCRIPT DELETING PROPERTIES

```
superClass ={
    language: "Javascript",
    version: 2.1,
}

subClass = {
    fork: "UVAscript",
}
```

```
subSubClass = {
    fork: "UVAsubscript"
}

Object.setPrototypeOf(subSubClass, subClass)
console.log(subSubClass.fork)
delete subSubClass.fork
// Allows the superclass property to shine through
console.log(subSubClass.fork)
```

PRINTS UVASUBSCRIPT

PRINTS UVASCRIPT

GLOBAL ABATEMENT

- Recall that variables declared with the var key word are available outside of the blocks in which it was defined.
- Nesting object inside of global object is one way to reduce unwanted interacts between variables. (Libraries etc)

GLOBAL ABATEMENT

```
APP = {}

APP.nestedClass = {
    prop1: 'nestedClass'
}

APP.nestedClass2 = {
    prop1: 'nestedClass'
}
```

NODE JS HAS A GLOBAL A OBJECT CALL GLOBAL THAT HOLDS ALL OFF THE VAR VARIABLES

JAVASCRIPT CLASSES

- The class keyword was introduced in ECMAScript 2015.
- However, they have not induced a new inheritance model the prototype inheritance model still holds

JAVASCRIPT CLASSES

```
class wahoo{
    constructor(leadership, grit){
        this.leadership = leadership
        this.grit = grit
    }
}
```

OVERRIDES THE CONSTRUCTOR IN OBJECT.PROTOTYPE

```
john = new wahoo(0.7,0.9)
console.log(john.grit)
```

USES THE NEW KEYWORD TO CALL THE CONSTRUCTOR

```
console.log(john.toString())
```

THE TOSTRING METHOD EXIST BECAUSE THE CLASS STILL INHERITS FROM OBJECT.PROTOTYPE