Web Advanced: Javascript APIs

"We will learn JavaScript properly. Then, we will learn useful design patterns. Then we will pick up useful tools for making cool things better."

FALL 2018

SESSION #3

SYNTAX: OBJECTS

jaink@newschool.edu

https://canvas.newschool.edu/courses/1407281

https://classroom.github.com/classrooms/42809645-parsons-web-advanced-javascript-fall-2018

RECAP

OBJECTS

- ➔ An object in JavaScript is a self-contained set of related values and functions.
- ➔ Javascript comprises of primitives (string, number, boolean, undefined, null)...and Objects.
- ➔ Arrays are types of objects
- ➔ Functions are types of objects
- ➔ Creating custom objects: literals
- ➔ Defining object methods
- ➔ JSON object

USING OBJECTS

Create an Object using the Constructor Method:

```
const my_date = new Date();
const new_string = new String('Hello');
```

Properties:

```
new_string.length;
new_array.length;
```

Methods:

```
new_string.toUpperCase();
new_array.sort();
```

BUILT-IN OBJECT TYPES

String Object:

```
let string1 = new String("Hello");
let string2 = new String(123);
let string3 = "This is me. "; //preferred
```

String Property:

```
string2.length;
```

String Methods:

```
string3.indexOf(pattern, start_position);
   string3.charAt(4);
string3.substring(start_position,
end_position);   string3.trim();
```

BUILT-IN OBJECT TYPES

Number Object:

```
let first_number = new Number(123);
let second_number = new Number(123.567);
let third_number = 3.14157; //preferred
```

Number Methods:

```
second_number.toFixed();
second_number.toFixed();
Number.isNaN(second_number);
Number.isInteger(second_number);
Number·parseInt(second_number);
Number.parseFloat(second_number)
```

BUILT-IN OBJECT TYPES

Math Object: already exists and is ready to be called:

```
let big_number = Math.PI; //constant
let pos_number = Math.abs(-23);
let sm_number = Math.floor(4.2);
let int_number = Math.round(4.5);
let pow_number = Math.pow(3,2);
let rand_number = Math.random();
```

BUILT-IN OBJECT TYPES

Array Object:

```
const my_array = new Array('test', 2, 4);
const my_array =['test', 2, 4];
```

Array Property:

```
my_array.length;
```

Array Methods:

```
my_array.push(9);
my_array.sort();
my_array.concat(second_array);
my_array.forEach(function() {} );
my_array.map(function() {} );
```

BUILT-IN OBJECT TYPES

Date Object: handling all date/time calculations

```
const the_date = new Date(); // actual
stored value: 1518110329605
```

```
const the_date1 = new Date("31 January
2014");
```

Date Methods:

```
the_date.getDate()
```

```
the_date.getDay()
```

```
getFullYear()
```

```
the_date.toDateString()
```

CUSTOM OBJECTS

Custom Object: like arrays but more flexible as it can encapsulate all of an entity's values and properties and methods to use in it.

```
// Constructor method:
const person = new Object();
// Literal method:
const person = {}; //preferred
```

Add Properties:

```
person.firstName = "John";
person.lastName = "Parsons";
```

Add Methods:

```
person.sayHello = function() {
    console.log("My name is " +
this.firstName + " " + this.lastName;
}
person.sayHello();
```

CUSTOM OBJECTS

```
const person = {
            firstName: "John",
            lastName: "Parsons",
            citiesLived: ["New York", "Boston",
"Vien"],
            sayHello: function() {
                        console.log("My name is "
                  + this.firstName + " " +
                  this.lastName);
                  }
        };
```

Alt approach with ES6:

```
let firstName: "John",
let lastName: "Parsons
// old way
const person = { firstName: firstName, lastName:
lastName };
// short ES6 way
const person = { firstName, lastName };
```

CUSTOM OBJECTS

Accessing:

```
person.firstName;
//OR
person["firstName"];
person["Alias Name"];
```

Updating:

```
person.firstName = "Abraham";
```

CUSTOM OBJECTS

Dynamic property names

```
const person = { firstName: "John" ,
lastName:"Parsons", ["Alias" + "Name"]:
"Johnny" };

console.log(person);
=> {   firstName: "John" ,
       lastName:"Parsons",
       AliasName: "Johnny"
   }
```

Dynamic values:

```
const person = { firstName: "John" ,
lastName:"Parsons", "Wealth":
(return_amount > 10**6) ? true : false };
```

CUSTOM OBJECTS

Looping through all properties: use for-in loop

```
for(let key in person) {
  console.log(key + ": " + person[key]);
}

Check if property exists:
if ("firstName" in person) {...};

person.hasOwnProperty("firstName");

for(const key of Object.keys(person)) {
    console.log(key);
}
//ES2017 only
for(const value of Object.values(person)) {
    console.log(value);
}
```

CUSTOM OBJECTS

Looping through all properties and values:

```
for(const [key,value] of
Object.entries(person)) {
    console.log(`${key}: ${value}`);
}
```

Removing a property:

```
delete person.alias
```

CUSTOM OBJECTS

Nested Objects:

```
const family = {
    dad: { realName: "John Denver" },
    mom: { realName: "Katie McGraw" },
    son: { realName: 'Lawrence Holly" }
}

family.dad.realName;
family.dad["realName"];
```

CUSTOM OBJECTS

Object vs. Primitives:

- ➔ **objects are assigned by reference**
- ➔ **variable is assigned to an object will simply refer to the exact same memory space**
- ➔ **any changes made using either reference will affect the same object**

```
const person = {
    name: 'Thor',
    weapon: "Axe"
};
const clonePerson = person;
clonePerson.name = Sif;
person.name; // returns Sif
```

FUNCTION PARAMS - OBJECTS

- ➔ **Useful when handling a large number of parameters to a function**
- ➔ **Order of params doesn't matter since properties in an object are not in a particular order**

```
const person = {
    name: 'Thor',
    weapon: "Axe"
};
function greet(person_object) {
    return `My name is ${person_object.name} and I wield the ${person_object.weapon}`;
}
// or with defaults:
function greet(name = "Hulk, weapon = "fist") {
    return `My name is ${name} and I wield the ${weapon}`;
}
```

BUILT-IN OBJECT TYPES

JSON Object: a cross-platform format used for exchanging information between web services

```
const json_person = '{"firstName": "John",
"lastName": "Parsons"}';
```

JSON Methods:

```
JSON.parse(json_person);
```

```
JSON.stringify(person);
```

THE CONCEPT OF THIS

- ➔ **keyword *this* refers to the object from within.**
- ➔ **used inside methods to gain access to the object's own properties**

```
const person = {
    firstName: "John",
    lastName: "Adams",
    fullName: function () {
         // using this to access own property
        console.log(this.firstName + " " +
this.lastName);
        // We could have also written this but
that has issues with namespace:
        console.log(person.firstName + " " +
person.lastName);
    }
}
```

EXAMPLES

```
// obj vars are NON-PRIMITIVES are references.
// so change one and other changes too eg.

const my_array = [0, 1, 2];
const my_secondarray = my_array;
my_array[0] = 100;
console.log(my_array);
console.log(my_secondarray);

// when working with primitives
let my_string = "test";
let my_secondstring = my_string;
my_string = "now";
console.log(my_string);
console.log(my_secondstring);



//exercise - get the current day
const days = ['sun', 'mon', 'tue', 'wed', 'thurs', 'fri',
'sat'];
let the_day = the_date.getDay();
console.log(days[the_day]);

//alt using localestring
const options = {  weekday: 'long', year: 'numeric',
month: 'long', day: 'numeric' };
let long_day = the_date.toLocaleDateString('en-US',
options);
long_day.split(',');
```

EXAMPLES

```
// custom objects
const person = {}; //preferred
person.firstName = "John";
person.lastName = "Parsons";
person.sayHello = function() {
  console.log("My name is " + this.firstName + " " +
this.lastName);
}

//looping through props only
for(let key in person) {
  if (typeof(person[key]) !== 'function') {
    console.log(key + ": " + person[key]);
  }
}

//a simple object project:
const dice = {
  sides: 6,
  roll: function(){
     return Math.floor(this.sides * Math.random()) + 1;
  }
}
// try changing sides: dice.sides = 12;
dice.roll();
```

Assignment:

Next Steps

➔ Manipulating with Document Object Model (DOM)