JS Notes

Sunday, February 11, 2018 8:41 AM

JS Understanding:

**** Object is collection of key value pair in Js**

Address : - Object

Name: Ritesh Kumar,
Phone: 9035625024,
Location: - Object
Street : Ranka Colony Road,
Building: No2

**** Global Objects And Global Environment**

Global Object:
Window - if its browser
this - global variable

**** Creating and Hosting**

Creation Phase: this phase will set this things

global objects
this
outer environment

Hosting : Setup memory space for variables and functions

Initially all js variable will set to undefined

function body will resides in memory

undefined is not a string its a keyword in js

**** JS is single threaded and synchronous in its behavior.**

**** Function Invocation and The Execution stack :**
Execution creation and execution

**** Let - Block Variables Declaration**

Local Variable Declaration

**** . Asynchronous js**

event queue

Event queue will be looked after execution stack will be empty

**** Types and JavaScript**

Dynamic Typing :- You won't tell the engine what type of data or variable holds, it figures it out while your code is running.

```
var a = "true";
        a = 10;
        a = "!yup";
```

variables can hold different types of values because it's all figured out during execution.

**** Primitive Types**

A type of data that represents a single value
this is not an object

6 type of primitive type:

1. undefined - lack of existence
2. null - lack of existence for setting empty value
3. BOOLEAN - true/f
4. NUMBER - with decimal value
5. STRING - value with ' ' and " "
6. SYMBOL - included in ES6

**** Operators**

A special function that is syntactically (written) differently
Generally operators takes two parameters and returns one result.
Operators are functions.

**** Operator Precedence and Associativity**

Associativity : -What order operator functions get called in left to right or right to left
When functions have same precedence

**** Coercion**

Converting a value from one type to another this happened frequent in JS because of its dynamic nature

```
var a = 1 + '2"
console.log(a); a = 12
```

**** Operator Comparison**

console.log(3 < 2 < 1) Return true

3 < 2 - false

console.log(false < 1) 0 < 1 - true

Number(undefined) -- NAN

Number(null) -- 0

false == 0 true

null == 0 false

"3" == 3 true "3" === 3 false strict equality

equality uses coercion and change the value like "3" to 3 so "3" == 3 NUMBER("3") = 3

What is the Object.is - Need to Explore this

// Need to check diff between ==, ===, Object.is in Mozilla developer network

a = "", null, undefined - if(a) - false

**** Object and function**

Object we can create by different way

Object Contains primitive properties, Object properties, Function properties

Type of way to create Object:

1. Through New Keyword
```
var person = new Object();
person["firstName"] = "Ritesh";
```

Accessing this object
```
person["firstName"] - Ritesh
person.firstName    - Ritesh
```

2. Through Object Literls
```
var personObject = {}; - Object Literals
var person = { firstName: 'Ritesh',
                lastName: 'Kumar',
                address: {
                        street: 'Ranka',
                        city: 'Bangalore',
                        state: 'KA'
                }
               }
```

**** Difference Between Object Literals and JSON**

Json will should have double or single quotes around its properties.
'{"FirstName": "Gudun", "LastName": "Singh"}'

So JSON is subset of object literals. all JSON are objects literals but not all object literals are json.
var ObjectVal = {firstName: 'Ritesh', lastName: 'Kumar'};

```
console.log(ObjectVal);
console.log(JSON.stringify(ObjectVal)); // Object to string
var jsonVal = '{"FirstName": "Gudun", "LastName": "Singh"}';
console.log(JSON.parse(jsonVal)); // string to Object
```

**** Functions are Object**

First Class Function:
Everything you can do with other types you can do with Java Script functions
Assign them to variables, pass them around, create them on fly

```
// First Class function
log(function() { // passed anynomous function and assigned to a and it printed
console.log("Hello");
});

var anonymousVal = function() {
console.log("Hello");
}
```

**** Function - a special type of Object**

It can contains
Primitives
Object
Function
Name - optional, can be anonymous
code - Invocable ()

**** Function Statement & Expressions**

Expression is a unit of code that result in a value.
It doesn't have to save to variable.

2+3 = 5 is expression because it returns value but not sits in memory

**** By Value & By Reference**

All Primitive types are by value means each variable will has its own copy in memory.
Example:

```
//By Value(All Primitives supports by Value)
// Means a and b will have there own memory address
var a = 3;
b = a;
b = 5;
console.log(a); // 3
console.log(b); // 5
```

All Object are by reference means each object refer to each other will point to same address in memory
Even functions also
Example:

```
//By Reference(All Objects including functions is by reference)
// Means two refenced objects will have same memory address (muttable)
```

```
var greet = {greeting: 'Hello'};
newGreet = greet;
newGreet.greeting = "Hola";
console.log(greet); //Hola
console.log(newGreet); //Hola

//By Referenece for functions
function greetings(obj) {
obj.greeting = "hi";
}
greetings(greet);
console.log(greet); //hi
console.log(newGreet); //hi

// Memory address will change if you use = sign so it will create new object
var greet = {greeting:'hello jee'}; // point new memory address
console.log(greet); //hello jee
console.log(newGreet); //hi
```

**** Object Function & This**

This is a object which will be available as windows object when use as global variable

**** Arrays**

Array will have different types of data in collection.

Example:

```
var arr = [
1, // Integer
true, // Boolean
{  // Object
name: "Ritesh",
address: "Bangalore"
},
function(name) { // function
console.log("hello "+name);
},
"Kumar"
]
console.log(arr);
arr[3](arr[2].name); - hello Ritesh
```

**** Arguments & Spread**

Execution Context is created (Functions)
variable Environment
Outer Environment
This
Arguments

Spread : like var args in java

...other - contains arrays of different arguments

```
function greet(firstName, middleName, lastName="Chauhan", ...others) { //
available in newer version of javascript
```

```
if(arguments.length === 0) {
console.log("No data available");
}
console.log(firstName);
console.log(middleName);
console.log(lastName);
console.log(others); // spread one case - "1", "2", "3" only 3 dot allowed
console.log("--------------------");
}
greet();
greet("Ritesh");
greet("Ritesh", "Kumar");
greet("Ritesh", "Kumar", "Singh");
greet("rit", "kum", "sing", "1", "2", "3");
```

**** Function Overloading is not available in JavaScript**

**** Dangerous Aside**

```
function getPerson() {
return // Return undefined because syntax parser added semi colon and returned
undefined from there
{
name: 'Tony'
}
}
console.log("Hola : "+getPerson());
```

**** Immediately Invoked Function Expression**

Invoked function immediately after function declaration
Invoke at run time
Example:

```
//Function Statement
function greet(name) {
console.log(name);
}
// Immediately Invoked Function Expression
var greetings = function(name) { // function Expression
return "Hola " + name;
}('Kumar');
greet('Ritesh');
console.log(greetings);
// Immediately Invoked Function Expression
(function(name) {
console.log("Hello "+name);
}('Chauhan'));
```

****Closures**

Closures need to understand better way

```
//demo 1 closures
function buildFunction() {
var arr = [];
for(var i=0; i<3; i++) {
// console.log("begin")
// console.log(i);
arr.push(function(){ // refernce will be persists due to closures
```

```
console.log(i);
});
// console.log("begin")
// console.log(arr);
}
return arr;
}
var f2 = buildFunction();
f2[0](); //3
f2[1](); //3
f2[2](); //3

// demo 2 of closures
function buildFunction2() {
var arr = [];
for(var i=0; i<3; i++) {
// console.log("begin")
// console.log(i);
arr.push(function(j){
return function() { // this will have there own execution context
console.log(j);
}
}(i))
// console.log("begin")
// console.log(arr);
}
return arr;
}
var f3 = buildFunction2();
f3[0](); //0
f3[1](); //1
f3[2](); //2
```

**** Closures and Callback Function**

A Function you give to another function, to be run when the other function is finished.
So the function you call(i.e invoke), 'calls back' by calling the function you gave it when it finishes.

Example:

```
function sayHiLater() {
var greeting = "Hi!";
setTimeout(function() {
console.log("greetings"); // greetings will be available through closures
}, 3000)
}
sayHiLater();
function tellMewhenDone(callback) {
var a = 1000;
var b = 2000;
callback();
}
tellMewhenDone(function() {
console.log('I am done!');
});
tellMewhenDone(function() {
```

```
alert('I am done! alert');
});
```

**** call(), apply() and bind()**