Chapter 6: Datatypes in JavaScript

Section 6.1: typeof

`typeof` is the 'official' function that one uses to get the `type` in JavaScript, however in certain cases it might yield some unexpected results ...

1. Strings

```
typeof "String" or
typeof Date(2011,01,01)
```

> "string"

2. Numbers

```
typeof 42
```

> "number"

3. Bool

`typeof true` (valid values `true` and `false`)

> "boolean"

4. Object

```
typeof {} or
typeof [] or
typeof null or
typeof /aaa/ or
typeof Error()
```

> "object"

5. Function

```
typeof function(){}
```

> "function"

6. Undefined

```
var var1; typeof var1
```

> "undefined"

Section 6.2: Finding an object's class

To find whether an object was constructed by a certain constructor or one inheriting from it, you can use the `instanceof` command:

```
//We want this function to take the sum of the numbers passed to it
//It can be called as sum(1, 2, 3) or sum([1, 2, 3]) and should give 6
function sum(...arguments) {
    if (arguments.length === 1) {
        const [firstArg] = arguments
        if (firstArg instanceof Array) { //firstArg is something like [1, 2, 3]
            return sum(...firstArg) //calls sum(1, 2, 3)
        }
    }
    return arguments.reduce((a, b) => a + b)
}

console.log(sum(1, 2, 3))   //6
console.log(sum([1, 2, 3])) //6
console.log(sum(4))         //4
```

Note that primitive values are not considered instances of any class:

```
console.log(2 instanceof Number)        //false
console.log('abc' instanceof String)    //false
console.log(true instanceof Boolean)    //false
console.log(Symbol() instanceof Symbol) //false
```

Every value in JavaScript besides `null` and `undefined` also has a `constructor` property storing the function that was used to construct it. This even works with primitives.

```
//Whereas instanceof also catches instances of subclasses,
//using obj.constructor does not
console.log([] instanceof Object, [] instanceof Array)           //true true
console.log([].constructor === Object, [].constructor === Array) //false true

function isNumber(value) {
    //null.constructor and undefined.constructor throw an error when accessed
    if (value === null || value === undefined) return false
    return value.constructor === Number
}
console.log(isNumber(null), isNumber(undefined))                  //false false
console.log(isNumber('abc'), isNumber([]), isNumber(() => 1))     //false false false
console.log(isNumber(0), isNumber(Number('10.1')), isNumber(NaN)) //true true true
```

Section 6.3: Getting object type by constructor name

When one with `typeof` operator one gets type `object` it falls into somewhat wast category...

In practice you might need to narrow it down to what sort of 'object' it actually is and one way to do it is to use object constructor name to get what flavour of object it actually is: `Object.prototype.toString.call(yourObject)`

1. String

```
Object.prototype.toString.call("String")
```

> "[object String]"

2. Number

```
Object.prototype.toString.call(42)
```

> "[object Number]"

3. Bool

```
Object.prototype.toString.call(true)
```

> "[object Boolean]"

4. Object

```
Object.prototype.toString.call(Object()) or
Object.prototype.toString.call({})
```

> "[object Object]"

5. Function

```
Object.prototype.toString.call(function(){})
```

> "[object Function]"

6. Date

```
Object.prototype.toString.call(new Date(2015,10,21))
```

> "[object Date]"

7. Regex

```
Object.prototype.toString.call(new RegExp()) or
Object.prototype.toString.call(/foo/);
```

> "[object RegExp]"

8. Array

```
Object.prototype.toString.call([]);
```

> "[object Array]"

9. Null

```
Object.prototype.toString.call(null);
```

> "[object Null]"

10. Undefined

```
Object.prototype.toString.call(undefined);
```

> "[object Undefined]"

11. Error

```
Object.prototype.toString.call(Error());
```

> "[object Error]"