JavaScript Review

types

Types in JavaScript

String	`"hello"`
Number	`2`
Boolean	`false`
Undefined	`undefined`
Null	`null`

} Primitive

Object {

Object	`{property: "value"}`
Array	`[1,2,3]`
Date	`new Date()`
RegExp	`/.*/g,`
Function	`function(){}`

Numbers

- Simply typed numeric digits.
- How many cookies do we have?
- Includes integers, positive/ negative, and decimal (floating point numbers)

```
// Integers are numbers
var x = 5;

// Fractional values are numbers
var y = 5.5;

// Negative values are numbers
var z = -3;
```

Basic Arithmetic Operators

- **Used to perform operations between numerical pieces of data.**
- **Common Arithmetic Operators**
  - Addition: +
  - Subtraction: -
  - Multiplication: *
  - Division: /
  - Remainder(Modulus): %
  - Can anyone think of a use case for this?

```
// Addition
var sum = 5 + 5;

// Subtraction
var dif = 5 - 5;

// Multiplication
var prod = 5 * 5;

// Division
var quot = 5 / 5;

// Modulus
var remainder = 5 % 5;
```

Immutability & Shorthand

- Operations on numbers DO NOT modify the numbers they operate on.
- This is true of all primitives in JavaScript. Modification requires *reassignment*.
- This syntax works the same for subtraction, multiplication, and division.

```
1  var num = 12;
2  num / 2;
3  // does nothing. num still 12
4
5  num += 2; // num is now 14
6  myNum ++; // num is now 15
7
8  console.log(myNum) // 15
```

Booleans

- For use when there are ONLY two possible states: true or false.
- Do I have any cookies left?
- true and false are RESERVED words.
- Attempting to use a reserved word as a variable name will cause the interpreter to throw an error.

```
var false = "hello";
// SyntaxError: Unexpected token false.
```

Null

- Used often as a placeholder to represent data that could be present but is currently 'null' in value.
  - Not to be confused with 0 or undefined.

```
var middleName = null;
```

Undefined

- A variable that has been DECLARED but a value has not been defined.
  - This is different then null because the value has NOT been set.

```
var middleName;
```

Strings

- A way of keeping a word/ phrase/character as a data type.
  - Who's eating all of my cookies?
- Any information that is wrapped in quotes
- (single ' or ")

```
var name = 'Corey';

var monster = "Say 'Hello' to me.";

var vegMonster = 'This is not Corey';
```

Concatenation

- The process of putting two things together.
  - NOT addition.
- But like addition, uses the + operator

```
var firstName = 'Amy';
var lastName = 'Smith';


var fullName = firstName + lastName;

console.log(fullName)  // Amy Smith
```

Accessing a String

- trings are made up of characters
  - Each character has an ‘index’ representing it’s location within the string.
  - These indices begin at 0.
- So to access the first letter we would write str[0]

```
var myDog = 'Fluffy';

console.log(myDog[0])  // F
console.log(myDog[1])  // l

console.log(myDog.length)  // 6
```

Built-in String Methods

- [] or .charAt() // Reference single string character
- .slice() or .substring() // returns a copy of a piece of a string
- .toUpperCase() // returns an uppercase version of the string
- .toLowercase() // returns a lowercase version of the string
- .split() // Splits string into array using arg as delimiter
- .length // the length of the string
- Number() // converts string to number
- Check it: https://repl.it/C83G/2
- And more: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/prototype

The `typeof` operator

- The `typeof` operator checks the type of the value it precedes, and returns a string value that (mostly) indicates its type.
- Note that `typeof [1, 2, 3]` returns `"object"` because arrays are of type object.

```
typeof undefined     === "undefined"; // true
typeof true          === "boolean";   // true
typeof 42            === "number";    // true
typeof "42"          === "string";    // true
typeof [1, 2, 3]     === "object";    // true

/* Some notes:

1.typeof returns a string
2.the first letter of the string is LOWERCASE
3.using typeof on an array returns "object"
  AS IT SHOULD

*/

// ODDLY:
typeof null          === "object";    // true
```