Cheat Sheet: **Basic** JavaScript utilizing ES6 © Howard Bates

Comments

```
// Inline comment (single-line)
// Great for URLs or disabling small code portions

/*
Block comment (multi-line)
For large chunks or disabling large code portions
*/

/**
* @desc JSDoc comment
* @desc Used for creating HTML code documents
*/
```

Selection Logic

```
if (value <= otherValue) {
    doStuff();
} else if (value >= differentValue) {
    doOtherStuff();
} else {
    doSuperOtherStuff();
}

switch (someValue) {
    case 'value1': doStuff();
        break;
    case 'value2': doOtherStuff();
        break;
    default: doSuperOtherStuff();
}
```

Operators

+	Addition operator (also overloaded concat opr)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulo operator
**	Exponent operator
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
===	Equal to value & data type (strict)
!==	Not equal to value & data type (strict)
!	Boolean NOT
||	Boolean OR
&&	Boolean AND
++	Auto increment (decrement is --)
+=	Adds value on right to left. REFERENCE + more!

condition ? expr1 : expr2 Ternary operator

Functions/Methods/Subroutines

```
function addNumbers(num1, num2) {
    return num1 + num2;
}

let sum = addNumbers(3, 4); // sum value is 7
```

Functions/methods/subroutines are all synonyms for a mini-program that runs inside your global program. This is useful for modularizing your code. The **return** keyword is used to pass a value back to a function caller. Variables & constants initialized within functions are locally scoped to that function only. Locally-scoped data may be passed to other functions as parameters. Everything in between the opening & closing curly braces is the function code block. The parenthesis after the function name is called the parameter list. A **function signature** is the line that contains the keyword function, the function name, & a receiving parameter list. A **function call** is simply the function name & sending parameter list.

Arrays

```
let people = []; //Simple SD array declaration

for (let i = 0; i < 1; i++) {
    people[i] = new Person(); //Assigns object to array
}

const COLUMNS = 3;
for (let i = 0; i < 1; i++) {
    people[i] = []; //Makes array MD
    for (let j = 0; j < COLUMNS; j++) {
        people[i][j] = someValue;
    }
}

people.push(value); //append to SD array right side
people.pop(); //remove from SD array right side
people.shift(); //remove from SD array left side
people.unshift(value); //append to SD array left side
people.sort(); //sorts SD array
people.reverse(); //reverse sorts SD array
```

Arrays are essentially variables that can hold multiple values in the same namespace. The values are accessed by an index position in square braces starting with 0. Arrays can either be single-dimensional (SD) or multi-dimensional (MD). MD Arrays look like spreadsheet with rows & columns. Use nested C-style for loops to iterate over MD array columns first then rows. Prefer SD arrays with object population over MD arrays.

Variables, Constants, & Data Types

```
let age; //Globally declared/initialized variable
const PI = 3.14; //Declared & assigned constant
let hitPoints = 14; //Local NUMBER variable
let animalType = 'Dog'; //Local STRING variable
let isValid = false; //local BOOLEAN variable
```

Variables are *declared/initialized* with the **let** keyword & are camel cased. Constants are *declared/initialized* with the **const** keyword and are ALL_CAPS with underscores. Variables & constants are both *assigned* with the = character (assignment operator). Global variables should only be *declared* initially, then *assigned* in a mutator method. Local variables should be preferred over global & declared + assigned on the same line. The primitive data types in JavaScript are: Number, String, Boolean, Undefined, Symbol, & Null (which is broken & technically an Object type).

Looping Logic

```
//while loop
while (value < otherValue) {
    doStuff();
    value++;
}

//C-style for loop
for (let i = 0; i < something; i++) {
    doStuff();
}

//for of loop
for (let value of array) {
    doStuff();
}
```

Use break to exit loop early. Use continue to skip current truth and continue on to next iteration of loop.

Recursion - !SEVERLY LIMIT!

```
function doStuff(value) {
    if (value < 1) {
        return value;
    } else {
        return doStuff(value - 1); //Recursive call
    }
}
```

Cheat Sheet: **Basic** JavaScript utilizing ES6 © Howard Bates

File I/O

```
const IO = require('fs');  // Library for file I/O

//Read data.csv data into SD array
function populatePeople() {
    let data = IO.readFileSync(`data.csv`, 'utf8');
    let lines = data.toString().split(/\r?\n/);
    for (let i = 0; i < lines.length; i++) {
        people.push(lines[i].toString().split(/,/));
    }
}

//Write SD Array back to dataX.csv file
function writePeople() {
    const COLUMNS = 6;
    for (let i = 0; i < people.length; i++) {
        for (let j = 0; j < COLUMNS; j++) {
            if (j < COLUMNS - 1) {
                IO.appendFileSync(`dataX.csv`, `${people[i][j]},`, 'utf8');
            } else {
                IO.appendFileSync(`dataX.csv`, people[i][j], 'utf8');
            }
        }
        IO.appendFileSync(`dataX.csv`, "\n", 'utf8');
    }
}
```

Programming 10 Commandments

01. Self-documenting code > commenting
02. Consistent, proper style
03. Prefer numeric over String
04. Explicit over implicit
05. NO magic numbers
06. Prefer local over global
07. Validate, distill, & sanitize input
08. Limit recursion
09. Loose coupling & high cohesion
10. Practice algorithms

Node.js Utilities & files

readline-sync :: Reading user input
fs :: File I/O
eslint :: Code quality parser (global)
.eslintrc :: Config file for ESLint
.gitignore :: Specifying git ignorable

Strings

```
`This is a text string inside template
literal. Note that newlines are handled
properly and there is no need for
concatenation as variable data can be
interprolated like this ${varName}.`

The \ character is used to escape special
characters like this: \$ \"
Escape sequences do special things.
\n is a new line.
\t is a tab.
```

toString() :: Converts object to string
trim() :: Trims whitespace
toUpperCase() & **toLowerCase()** :: self-expl.
split() :: Splits string into array
slice() :: Extract section of string
substr() :: Similar to **slice()**

Object Oriented Programming (OOP)

```
class MakeObject {
    constructor() {
        MakeObject.doStuff(); //Calling static method
        this.doOtherStuff(); //Calling instance method
    }

    //Static method
    static doStuff() {
        console.log(`Doing stuff.`);
    }

    //Instance method
    doOtherStuff() {
        console.log(`Doing other stuff.`);
    }
}

//Instantiate a new MakeObject object
{
    new MakeObject();
}
```

Basic Four-Section Code file Layout (non-OOP)

Section 1:
Comment-header Block, pragmas, & library imports

Section 2:
Global variables & constants

Section 3:
Dispatch section using main method and call to main. e.g.:

```
function main() {
    doStuff1();
    doStuff2();
}

main();
```

Section 4:
All mutator methods for global variables & general utility worker methods.

Try/Catch, Throw, Code Debugging

```
try {
    doStuff(1);
} catch (error) {
    logErrors(error);
} finally {
    sayGoodbye();
}

function doStuff(someNum) {
    if (someNum > 0) {
        throw `Error: High Num`;
    }
    console.log(someNum);
}
```

The try...catch statement marks a block of statements to try, and specifies a response, should an exception be thrown. Use the throw statement to throw an exception.

Use **console.log()** prodigiously to debug problems in your code. Surround actions inside functions, etc.