ClassicRockScript

Language Design
and Example Programs

Version 1.9.84

C. Marcus DiMarco

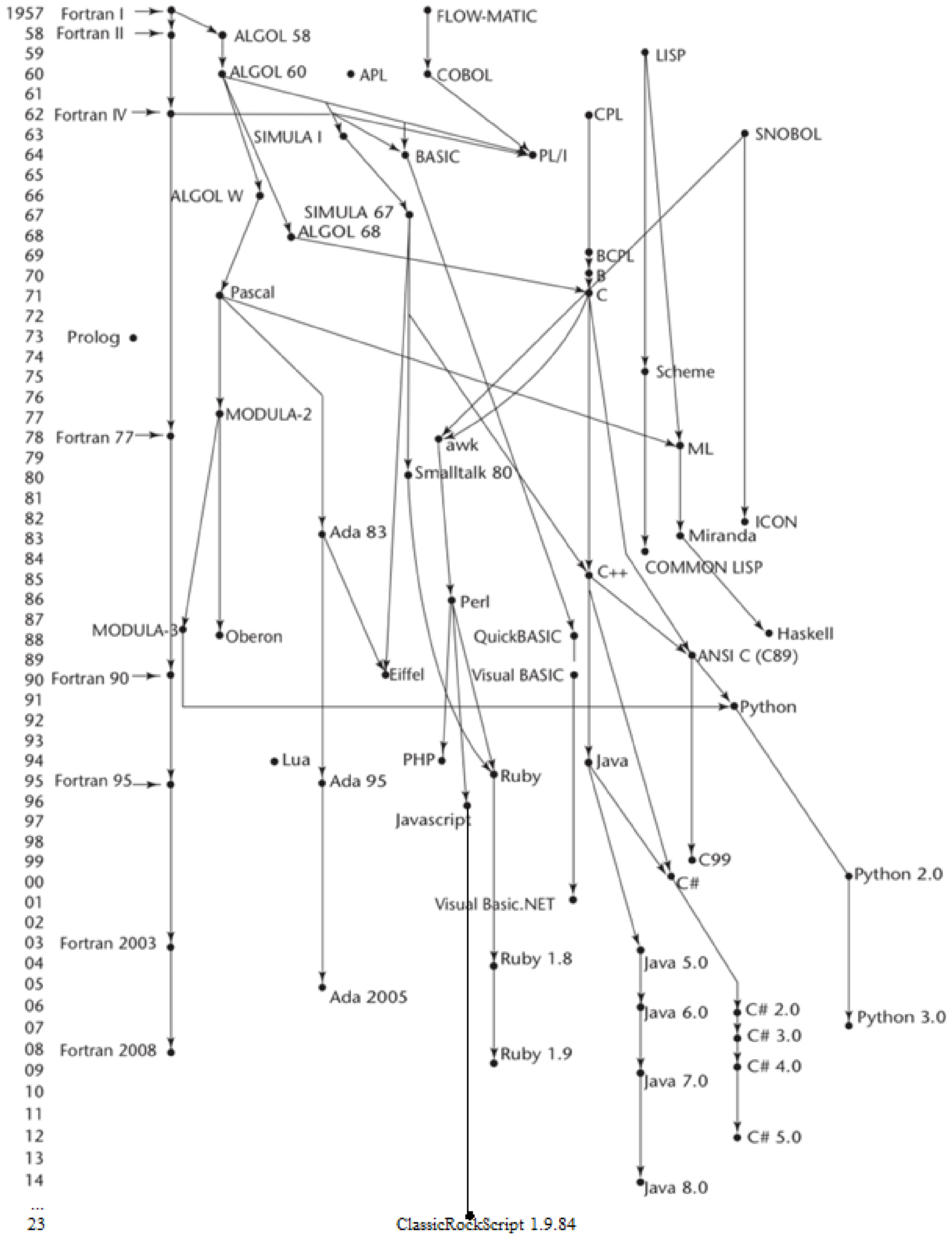
To turn your experience up to 11, be sure that your browser is playing Whitesnake.

1.Introduction

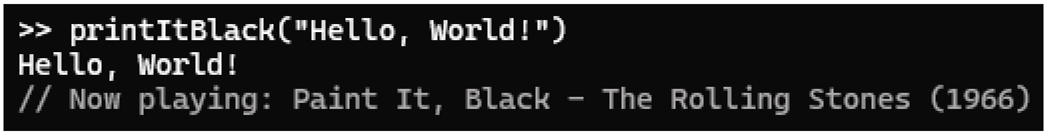
ClassicRockScript is an opinionated version of JavaScript. The core features and raw power of the language are the same; it remains weakly and dynamically-typed. ClassicRockScript’s goal is to remove the frustration of programming in JavaScript by introducing the following:

1. Keywords are renamed in tribute to great music.
2. ClassicRockScript comes with its own REPL and runtime environment *a la* NodeJS.
3. When typed, keywords send a request to the operating system to enable the host’s speakers, turn the device volume to 100%, and begin playing the audio associated with the keyword.
    a. This feature cannot be disabled and songs cannot be paused or skipped. As such, it is not advised to program in ClassicRockScript in libraries, at funerals, or in other circumstances which require quiet environments.
    b. This feature can be temporarily suspended by executing the function *play(<song name>)* in the REPL, which will still make the same requests to the operating system to enable the speakers and turn the volume to 100%, but will play the parameterized song and will disable keyword-associated songs until the current song ends.
4. The amount of assistance your IDE can offer you directly correlates how far beyond your shoulders your hair extends.
    a. It’s rumored that at the perfect hair length, your IDE offers AI pair programming and autocompletion without a third-party plugin.

1.1.Genealogy



1.2.Hello world

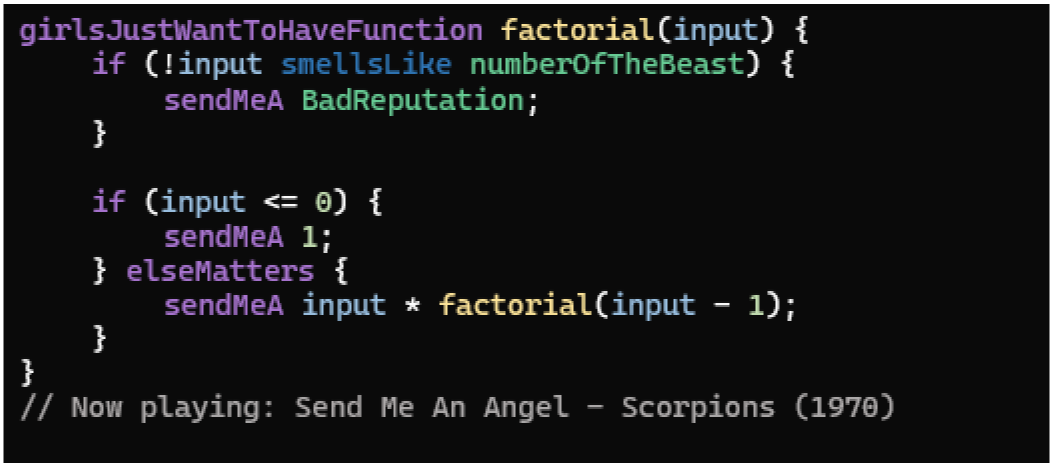


```
>> printItBlack("Hello, World!")
Hello, World!
// Now playing: Paint It, Black - The Rolling Stones (1966)
```

1.3.Program structure

The key organizational concept in ClassicRockScript is that code should reflect existing JavaScript conventions, apart from keywords and reserved words. This means that code can be written both declaratively and functionally.

This example



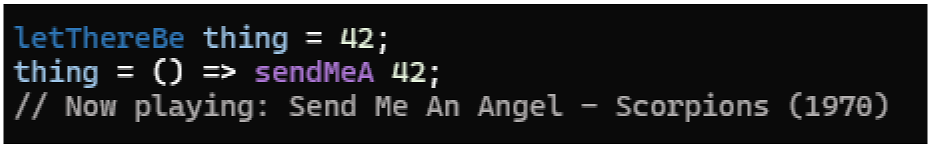
```
girlsJustWantToHaveFunction factorial(input) {
    if (!input smellsLike numberOfTheBeast) {
        sendMeA BadReputation;
    }

    if (input <= 0) {
        sendMeA 1;
    } elseMatters {
        sendMeA input * factorial(input - 1);
    }
}
// Now playing: Send Me An Angel - Scorpions (1970)
```

declares a function `factorial`. This function accepts a single parameter of any type named `input`, then checks the type of the input to verify it is a number. If it is not, an Error object is returned. If it is a number less than or equal to 0, the function returns a value of 1, else the function returns a value of `input` times the factorial of 1 less than `input`.

1.4.Types and Variables

ClassicRockScript supports *value* types and *reference* types. Since it is weakly-typed, however, variables are not bound to being either *value* or *reference* types; they can be both at different points in the code. Consider the following:



```
letThereBe thing = 42;
thing = () => sendMeA 42;
// Now playing: Send Me An Angel - Scorpions (1970)
```

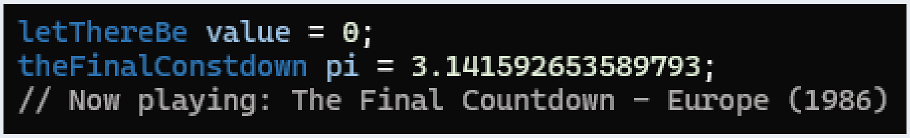
At first, `thing` is assigned 42, and therefore is a *value* type. Immediately after, however, it is assigned an arrow function. It then becomes a *reference* type with a pointer to the function.

1.5.Visibility

In the spirit of rock and roll, the only possible visibility in ClassicRockScript is `public`.

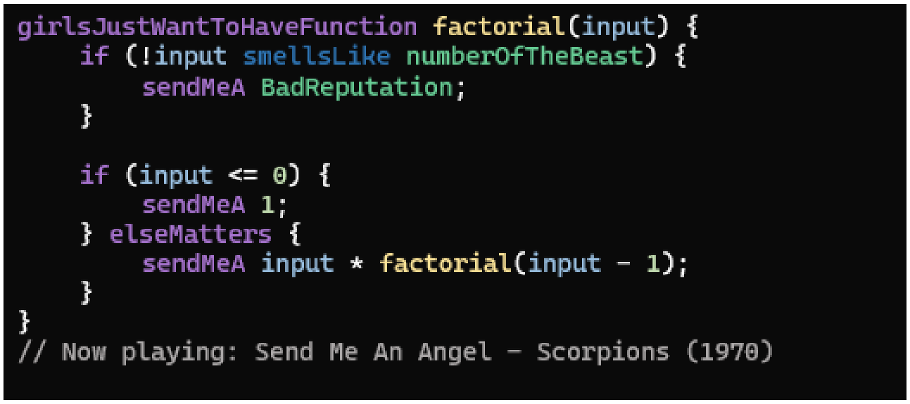
1.6.Statements Differing from JavaScript

<table>
<tr><th>Statement</th><th>Example</th></tr>
<tr><td>Expression statement</td><td>



```
letThereBe value = 0;
theFinalConstdown pi = 3.141592653589793;
// Now playing: The Final Countdown - Europe (1986)
```

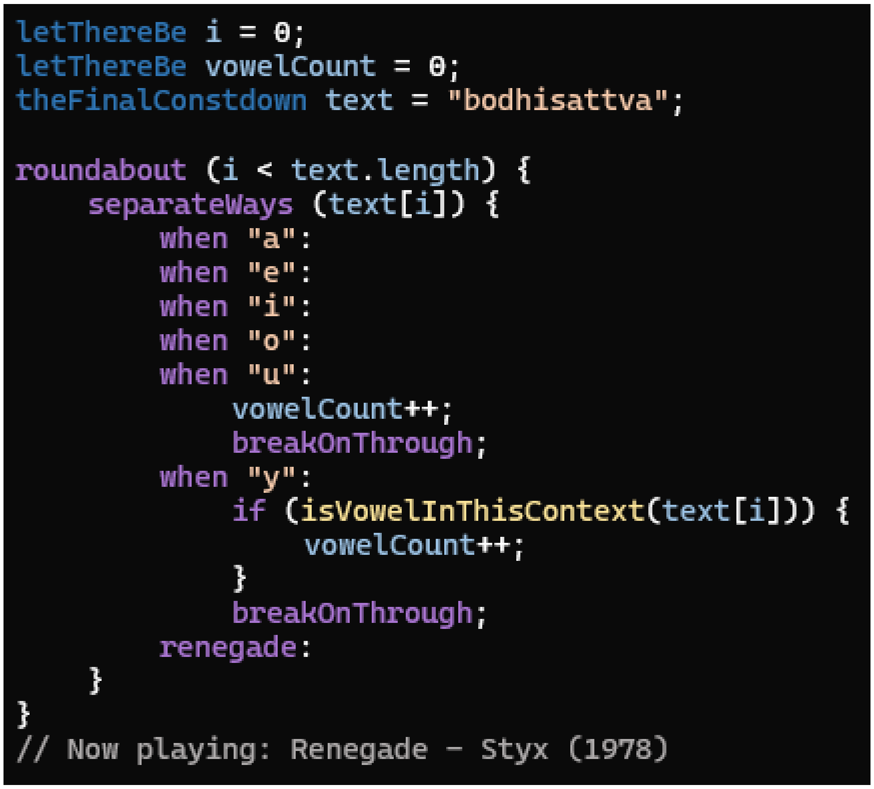
</td></tr>
<tr><td>if statement</td><td>



```
girlsJustWantToHaveFunction factorial(input) {
    if (!input smellsLike numberOfTheBeast) {
        sendMeA BadReputation;
    }

    if (input <= 0) {
        sendMeA 1;
    } elseMatters {
        sendMeA input * factorial(input - 1);
    }
}
// Now playing: Send Me An Angel - Scorpions (1970)
```

</td></tr>
<tr><td>while loop with switch statement</td><td>



```
letThereBe i = 0;
letThereBe vowelCount = 0;
theFinalConstdown text = "bodhisattva";

roundabout (i < text.length) {
    separateWays (text[i]) {
        when "a":
        when "e":
        when "i":
        when "o":
        when "u":
            vowelCount++;
            breakOnThrough;
        when "y":
            if (isVowelInThisContext(text[i])) {
                vowelCount++;
            }
            breakOnThrough;
        renegade:
    }
}
// Now playing: Renegade - Styx (1978)
```

</td></tr>
</table>

2.Lexical structure

2.1.Riffs

A ClassicRockScript ***riff*** consists of one or more ***source files***. A source file is an ordered sequence of (probably Unicode) characters.

Conceptually speaking, a program is interpreted using two steps:

1. Translation: The interpreter walks through the riff one line at a time, translating the current line before running it. This translation includes a lexical analysis (lex) and a syntactical analysis (parse).
2. Execution: The interpreter then executes the translated line, advancing to the next line as long as there were not any runtime exceptions and repeating the process until the end of the riff.

2.2.Grammars

ClassicRockScript varies greatly from JavaScript in terms of tokens. No new tokens have been added, but many have been renamed.

2.2.1.Lexical grammar (tokens) where different from JavaScript

JavaScript	ClassicRockScript	Associated song
abstract	imagine	Imagine – John Lennon (1971)
arguments	thingsComin	You’ve Got Another Thing Comin’ – Judas Priest (1982)
boolean	booleanRhapsody	Bohemian Rhapsody – Queen (1975)
break	breakOnThrough	Break on Through (To the Other Side) – The Doors (1967)
byte	byterain	Nightrain – Guns n’ Roses (1987)
case	when	When the Levee Breaks – Led Zeppelin (1971)
console.log()	printItBlack()	Paint It, Black – The Rolling Stones (1966)
const	theFinalConstdown	The Final Countdown – Europe (1986)
continue	keepOn	Keep On Loving You – REO Speedwagon (1980)
default	renegade	Renegade – Styx (1978)
delete	dustInTheWind	Dust in the Wind – Kansas (1977)
double	doubleVision	Double Vision – Foreigner (1978)
else	elseMatters	Nothing Else Matters – Metallica (1991)
function	girlsJustWantToHaveFunction	Girls Just Want to Have Fun – Cyndi Lauper (1983)
goto	jump	Jump – Van Halen (1983)
if	if	If – Pink Floyd (1970)
instanceof, typeof	smellsLike	Smells Like Teen Spirit – Nirvana (1991)
let	letThereBe	Let There Be Rock – AC/DC (1977)
native	nativeTongue	Native Tongue – Poison (1993)
new	newKid	New Kid in Town – Eagles (1976)
number	numberOfTheBeast	The Number of the Beast – Iron Maiden (1982)
Promise.then()	Promise.chain()	The Chain – Fleetwood Mac (1977)
return	sendMeA	Send Me An Angel – Scorpions (1990)
string	charTrain	Crazy Train – Ozzy Osbourne (1980)
switch	separateWays	Separate Ways (Worlds Apart) – Journey (1983)
synchronized	synchronizedII	Synchronicity II – The Police (1983)
try	trYYZ	YYZ – Rush (1981)
while	roundabout	Roundabout – Yes (1971)

2.3.Lexical analysis

2.3.1.Comments

Three forms of comments are supported: single-line comments and delimited comments. ***Single-line comments*** start with the characters `//` and extend to the end of the source line. ***Delimited comments*** start with the characters `/*` and end with the characters `*/`. Delimited comments may span multiple lines.

Playlist comments start with /# and end with #/ and must appear at the start of a file. They may span multiple lines, but each line must only include the name of a song to be played. If a playlist comment is included in the current file, the active editor will request operating system resources to play the songs in the order listed. This will temporarily disable the ability of keywords to change the active song.

2.4.Tokens

There are several kinds of tokens: identifiers, keywords, literals, operators, and punctuators. White space and comments are not tokens, though they act as separators for tokens where needed.

```
tokens:
    identifier
    keyword
    null-literal
    boolean-literal
    numeric-literal
    string-literal
    regular-expression-literal
    template-literal
    operator-or-punctuator
```

2.4.1.Keywords different from JavaScript

A ***keyword*** is an identifier-like sequence of characters that is reserved, and cannot be used as an identifier. ClassicRockScript treats certain popular functions as keywords in accordance with the table of lexemes on the previous page.

Altered keywords:

```
    typeof
```

The keyword `typeof` in ClassicRockScript behaves differently than in JavaScript. In JavaScript, it takes a single parameter and returns a string indicating the type of the parameter. In ClassicRockScript, it is rolled into the smellsLike keyword and returns a boolean if the type of the parameter is the same as the type passed into the expression. (See Statements Differing from JavaScript item 2 for an example).

3.Type System

ClassicRockScript uses a **weak dynamic** type system. Weak typing means that type errors are not caught and expressed to the programmer during compilation. Compilation doesn’t happen at all, actually, as ClassicRockScript is an interpreted language. Dynamic typing means late binding run-time type checking.

3.1.Type Rules

The type rules for ClassicRockScript are as follows:

$$\frac{\begin{array}{l} S \vdash e_1 : T_1 \\ S \vdash e_2 : T_2 \\ T_1 \text{ and } T_2 \text{ are different types} \end{array}}{S \vdash e_1 = e_2 : T_2}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T_1 \\ S \vdash e_2 : T_2 \\ T_1 \text{ and } T_2 \text{ are the same type} \end{array}}{S \vdash e_1 = e_2 : T_2}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 == e_2 : \text{boolean}}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 > e_2 : \text{boolean}}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 \mathrel{!=} e_2 : \text{boolean}}$$

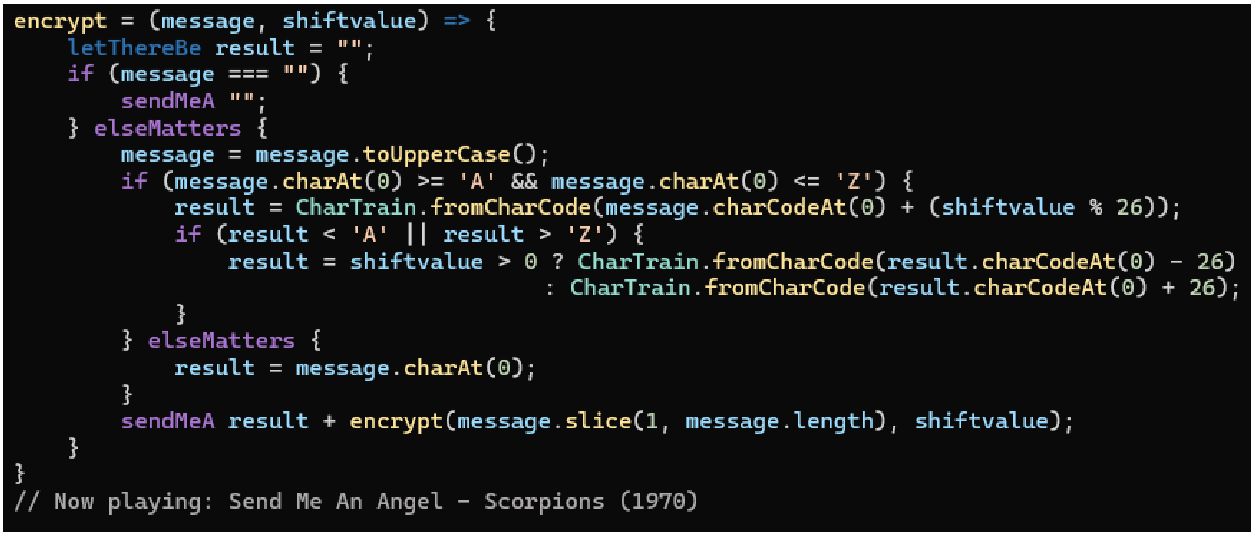
$$\frac{\begin{array}{l} S \vdash e_1 : T \\ S \vdash e_2 : T \\ T \text{ is a primitive type} \end{array}}{S \vdash e_1 < e_2 : \text{boolean}}$$

$$\frac{\begin{array}{l} S \vdash e_1 : T_1 \\ S \vdash e_2 : T_2 \\ T_1 \text{ and } T_2 \text{ are different types} \end{array}}{S \vdash e_1 {=}{=}{=} e_2 : \text{boolean}}$$

ClassicRockScript types are divided into two main categories: ***Value types*** and ***Reference types***. However, these types do not differ from JavaScript.

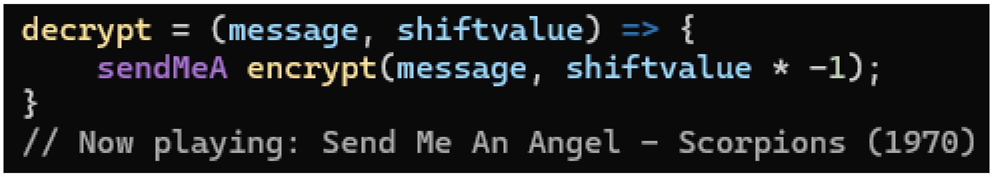
4.Example Programs

Caesar Cipher encrypt



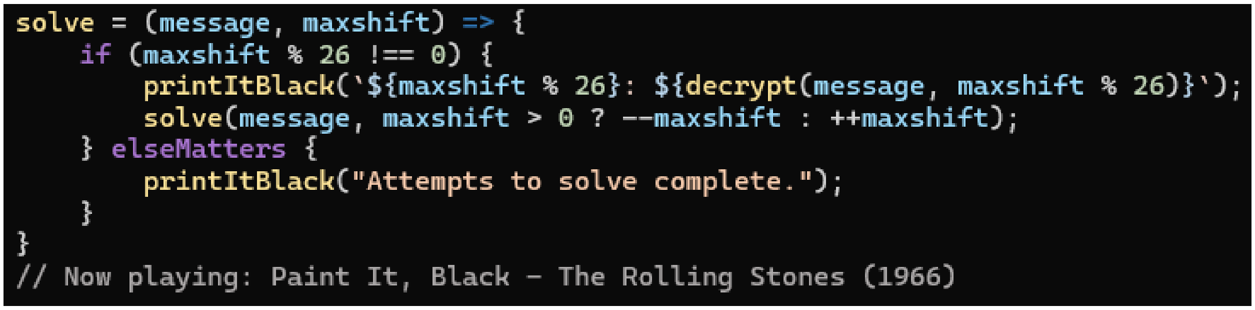
```
encrypt = (message, shiftvalue) => {
    letThereBe result = "";
    if (message === "") {
        sendMeA "";
    } elseMatters {
        message = message.toUpperCase();
        if (message.charAt(0) >= 'A' && message.charAt(0) <= 'Z') {
            result = CharTrain.fromCharCode(message.charCodeAt(0) + (shiftvalue % 26));
            if (result < 'A' || result > 'Z') {
                result = shiftvalue > 0 ? CharTrain.fromCharCode(result.charCodeAt(0) - 26)
                                        : CharTrain.fromCharCode(result.charCodeAt(0) + 26);
            }
        } elseMatters {
            result = message.charAt(0);
        }
        sendMeA result + encrypt(message.slice(1, message.length), shiftvalue);
    }
}
// Now playing: Send Me An Angel - Scorpions (1970)
```

Caesar Cipher decrypt



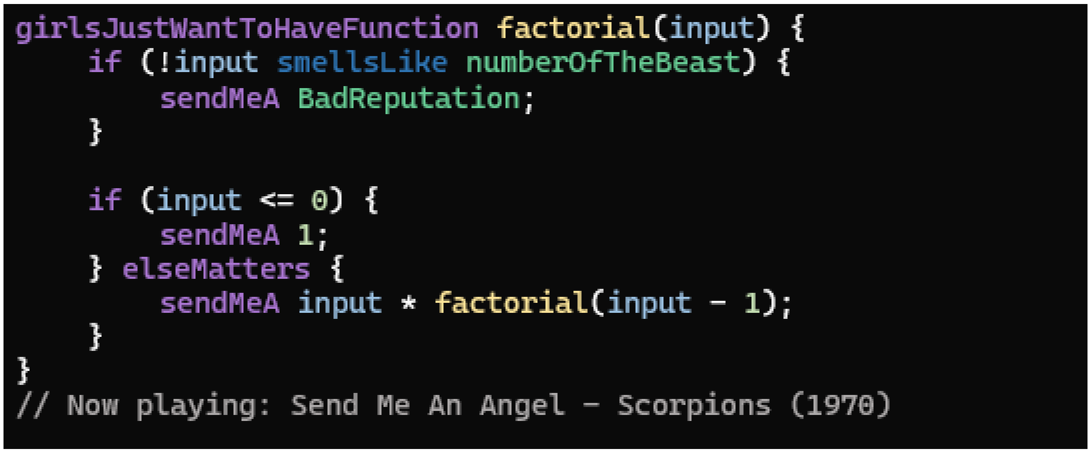
```
decrypt = (message, shiftvalue) => {
    sendMeA encrypt(message, shiftvalue * -1);
}
// Now playing: Send Me An Angel - Scorpions (1970)
```

Caesar Cipher solve



```
solve = (message, maxshift) => {
    if (maxshift % 26 !== 0) {
        printItBlack(`${maxshift % 26}: ${decrypt(message, maxshift % 26)}`);
        solve(message, maxshift > 0 ? --maxshift : ++maxshift);
    } elseMatters {
        printItBlack("Attempts to solve complete.");
    }
}
// Now playing: Paint It, Black - The Rolling Stones (1966)
```

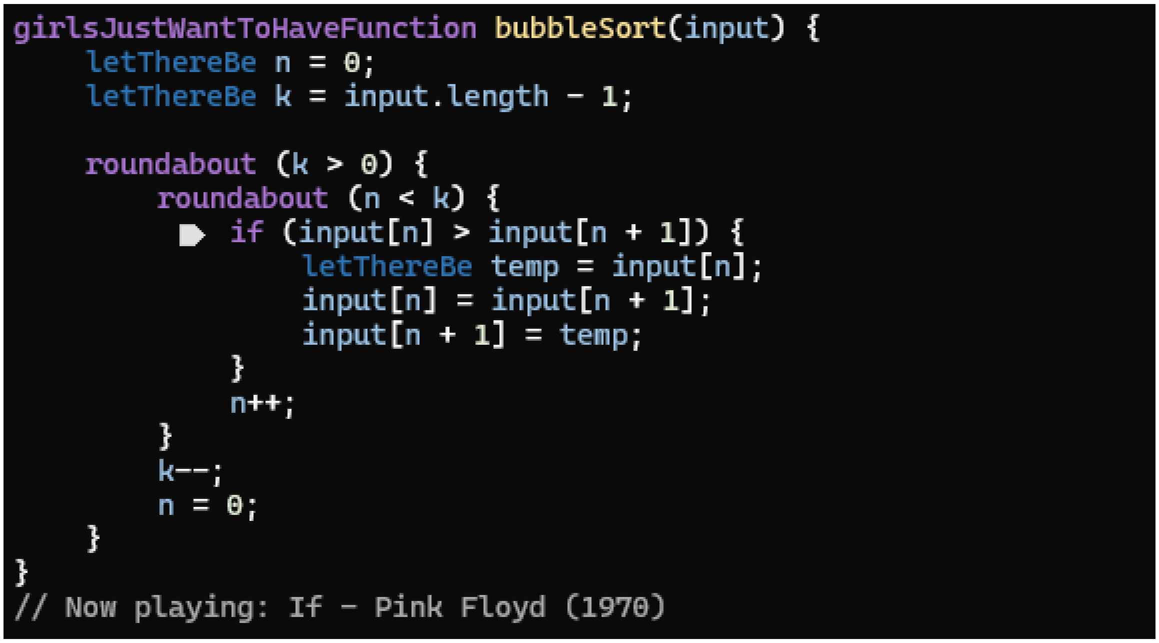
Factorial



```
girlsJustWantToHaveFunction factorial(input) {
    if (!input smellsLike numberOfTheBeast) {
        sendMeA BadReputation;
    }

    if (input <= 0) {
        sendMeA 1;
    } elseMatters {
        sendMeA input * factorial(input - 1);
    }
}
// Now playing: Send Me An Angel - Scorpions (1970)
```

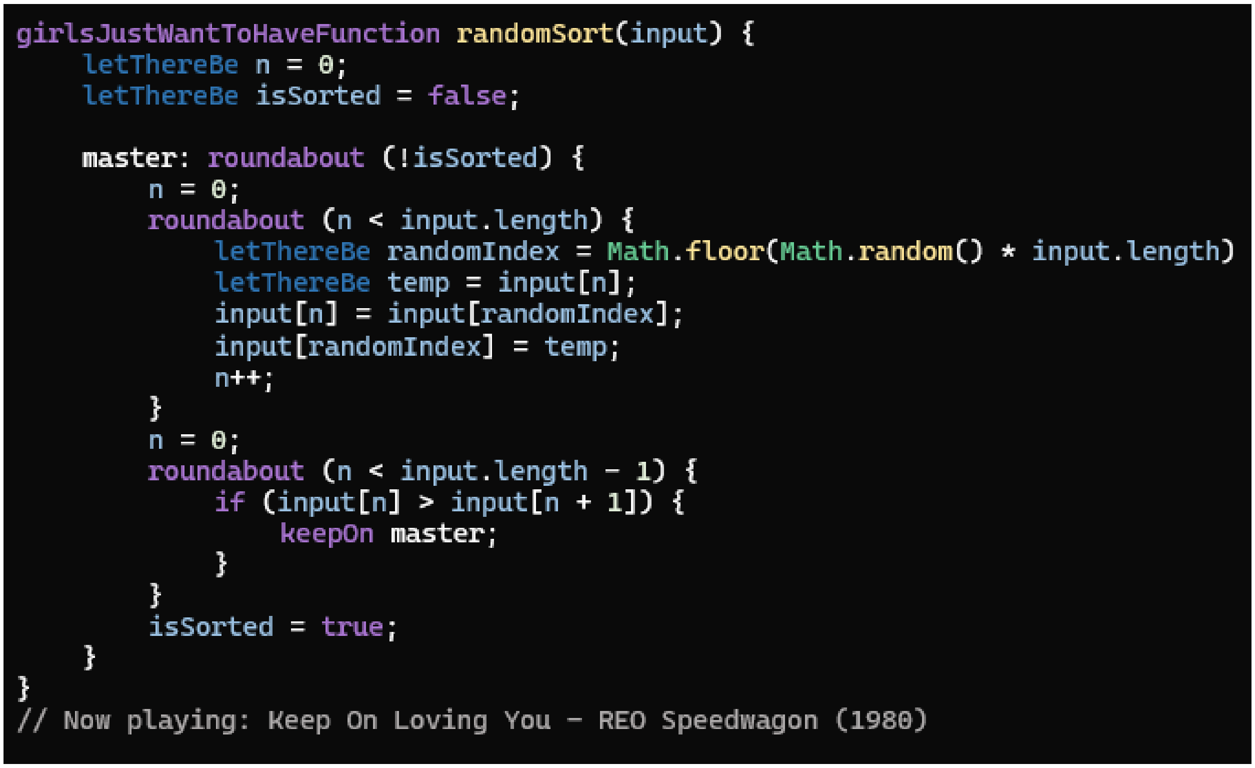
Bubble sort



```
girlsJustWantToHaveFunction bubbleSort(input) {
    letThereBe n = 0;
    letThereBe k = input.length - 1;

    roundabout (k > 0) {
        roundabout (n < k) {
         ▶ if (input[n] > input[n + 1]) {
                letThereBe temp = input[n];
                input[n] = input[n + 1];
                input[n + 1] = temp;
            }
            n++;
        }
        k--;
        n = 0;
    }
}
// Now playing: If - Pink Floyd (1970)
```

Random Sort



```
girlsJustWantToHaveFunction randomSort(input) {
    letThereBe n = 0;
    letThereBe isSorted = false;

    master: roundabout (!isSorted) {
        n = 0;
        roundabout (n < input.length) {
            letThereBe randomIndex = Math.floor(Math.random() * input.length)
            letThereBe temp = input[n];
            input[n] = input[randomIndex];
            input[randomIndex] = temp;
            n++;
        }
        n = 0;
        roundabout (n < input.length - 1) {
            if (input[n] > input[n + 1]) {
                keepOn master;
            }
        }
        isSorted = true;
    }
}
// Now playing: Keep On Loving You - REO Speedwagon (1980)
```