Day 0: Hello, World!

by AvmnuSng

JavaScript Basics

Lexical Structure

The lexical structure of a programming language is the set of elementary rules that tells you *how* to write programs in that language. It's essentially the lowest-level syntax of a

Character Set

- JavaScript programs are written using the *Unicode* character set. Unicode is a superset of *ASCII* and *Latin-1*.
- JavaScript is a case-sensitive language.
- JavaScript ignores spaces that appear between tokens in programs. For the most part, JavaScript also ignores line breaks.

Printing Output

We use the *console.log* method to write data to standard output in JavaScript.

- EXAMPLE

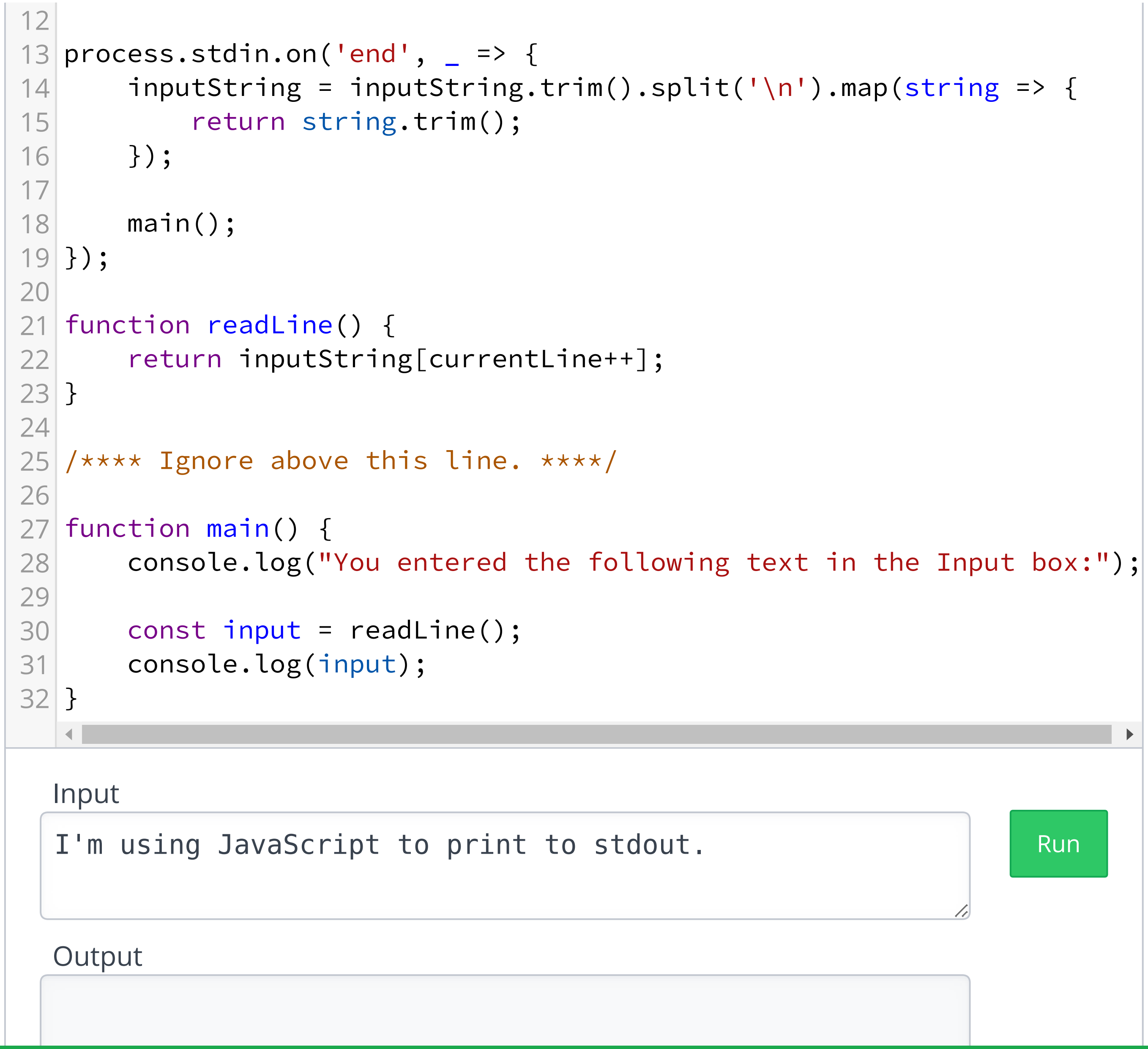
Run the code below to demonstrate printing with *console.log*.

```
'use strict';

process.stdin.resume();
process.stdin.setEncoding('utf-8');

let inputString = '';
let currentLine = 0;

process.stdin.on('data', inputStdin => {
```



```

process.stdin.on('end', _ => {
    inputString = inputString.trim().split('\n').map(string => {
        return string.trim();
    });

    main();
});

function readLine() {
    return inputString[currentLine++];
}

/**** Ignore above this line. ****/

function main() {
    console.log("You entered the following text in the Input box:");

    const input = readLine();
    console.log(input);
}
```

Input

```
I'm using JavaScript to print to stdout.
```

Output

Comments

JavaScript supports two styles of comments, as demonstrated below.

Inline Comments

Any text between a `//` and the end of a line is ignored by JavaScript and treated as a comment:

```
console.log("This is an instruction that won't be ignored.");
// This is an inline comment and will be ignored
```

Block Comments

Any text between `/*` and `*/` is also treated as a comment:

```
console.log("This is an instruction that won't be ignored.");
/*
 * This is a block comment and will be ignored
 */

console.log("This is an instruction that won't be ignored.");

/*
 * This is part of our block comment and will be ignored
 * This is part of the same block comment and will be ignored
 */
```

A literal is a data value that appears directly in a program. For example:

```
// The integer number twelve:
12

// The floating-point number one-point-two:
1.2

// A string of text:
"Hello, World."

// Another string:
'Hi!'

// A boolean value:
true

// The absence of an object:
null
```

More complex expressions can serve as array and object literals.

```
// An object initializer:
{x: 1, y: 2}

// An array initializer:
[1, 2, 3, 4, 5]
```

An identifier is simply a name that you can specify and use as a means of referring back to a specific value or other piece of code. In JavaScript, identifiers are used to name variables and functions, as well as to provide labels for certain code loops.

A JavaScript identifier must begin with a letter, an underscore (`_`), or a dollar sign (`$`). Subsequent characters can be letters, underscores, dollar signs, or *digits* (i.e., the numbers 0 through 9). Like many other languages, JavaScript doesn't allow digits as the first character of an identifier because it makes them more easily distinguishable from numbers.

```
// Some valid identifiers are:
x
variable_name
sum13
_variable
$variable
```

A number of identifiers are *reserved words* or *keywords*, meaning they are part of a set of predefined words that have special meaning in the language itself. You cannot use these words as identifiers in your programs. For example, `for` and `function` are reserved words in JavaScript. In addition, there are a number of predefined global variables and functions; it's important to avoid using these predefined names for your own variables and functions.

Optional Semicolon

without a separator, the end of one statement might appear to be the beginning of the next (and vice versa). In JavaScript, you can usually omit the semicolon between two statements as long as those statements are written on separate lines.