Day 2: Conditional Statements: Switch

All topics

1 Switch Conditional Statements
2 String Basics

Switch Conditional Statements

JavaScript Switch Statements

A *switch* statement allows a program to evaluate an expression by attempting to match the expression's value to a *case label*. If a match is found, the program jumps to the statement(s) associated with the matched label and continues executing at that point. Note that execution will continue sequentially through all the statements starting at the jump point unless there is a call to `break;`, which exits the switch statement. A switch statement looks like this:

```
switch (expression) {
    case label1:
        statement1;
        break;
    case label2:
        statement2;
        break;
    case label3:
        statement3;
        statement4;
        break;
    default:
        statement;
}
```

The program first looks for a `case` clause with a label matching the value of ***expression***, then transfers control to the matching clause and executes the associated statements. If no matching label is found, the program looks for the optional *default* clause and, if found, transfers control to that clause and executes the statements associated with it. If no `default` clause is found, the program continues executing after the end of the switch statement.

The `default` Clause

By convention, the `default` clause is always listed last. This is because the statements are checked sequentially, so you run into the following issues if you use the `default`

Table Of Contents

JavaScript Switch Statements
The `default` Clause
The `break;` Statement
Multi-Criteria Case

Go to Top

label in an earlier clause:

- If the `default` case is listed *before* (above) a case that matches ***expression***, it will match the `default` case instead. This means the statements associated with the programmed match case won't be executed.
- If the `default` case doesn't have a break statement, any statements in the case label immediately following it will be executed.

The `break;` Statement

The break statement is optional, but you'll typically see one at the end of each `case` clause to ensure that the program breaks out of the switch statement once the statements associated with a matched case are executed. Once the flow of execution hits `break;`, it exits the switch statement and continues executing at the next line following the end of the switch statement; if the break statement is omitted, the program continues executing the next statement in the switch statement — even if its case label doesn't match ***expression***.

- EXAMPLE

Given an integer, n, such that $0 < n < 11$, do the following:

1. If n is equal to 2, print `A`.
2. If n is equal to 3, print `B`.
3. If n is equal to 4, print `C`.
4. If n is equal to 5, print `D`
5. For all the other values of n, print `E`.

Input Format

A single integer denoting n.

```
var input = "";
process.stdin.on('data', function (data) {
    input = data;
    switchDemo();
});
function readLine() { return input; }
/**** Ignore above this line. ****/

function switchDemo() {
    var n = parseInt(readLine());

    switch (n) {
        case 2:
            console.log("A");
            break;
        case 3:
            console.log("B");
            break;
        case 4:
            console.log("C");
            break;
        case 5:
            console.log("D");
            break;
        default:
            console.log("E");
    }

    console.log("Exited switch.");
```

Input

4

Run

Go to Top

Output

Run the code above with the given input, and then try replacing that input with other integers and seeing how it changes. Note that, once reached, the `break;` statements transfer control back outside of the switch statement to the next line of code (in this example, there is no more code to execute).

Now, let's consider the same problem, but this time we'll remove all the `break;` statements from our code:

```
var input = "";
process.stdin.on('data', function (data) {
    input = data;
    switchDemo();
});
function readLine() { return input; }
/**** Ignore above this line. ****/

function switchDemo() {
    var n = parseInt(readLine());

    switch (n) {
        case 2:
            console.log("A");
        case 3:
            console.log("B");
        case 4:
            console.log("C");
        case 5:
            console.log("D");
        default:
            console.log("E");
    }

    console.log("Exited switch.");
```

Input

```
4
```

Run

Output

Run the code above with the given input, and then try replacing that input with other integers and seeing how it changes. Observe that the statements execute sequentially, starting with the matching case.

Now, let's look at what happens if we don't parse the input as an integer:

```
var input = "";
process.stdin.on('data', function (data) {
    input = data;
    switchDemo();
});
function readLine() { return input; }
/**** Ignore above this line. ****/

function switchDemo() {
    // This will read n as an object.
    var n = readLine();

    switch (n) {
        case 2:
            console.log("A");
            break;
        case 3:
            console.log("B");
```

Go to Top

```
            break;
        case 4:
            console.log("C");
            break;
        case 5:
            console.log("D");
            break;
        default:
            console.log("E");
    }

    console.log("Exited switch.");
```

Input

4

Run

Output

Run the code above with the given input, and observe that the code *does not* parse n as an integer. This means it's evaluated as an object, where a strict comparison (`===`) is made comparing the *case label* to the *expression value*.

Multi-Criteria Case

In the example below, we consider a similar problem in which there are multiple criteria for each case.

\- EXAMPLE

Given an integer, n, such that $0 < n < 11$, do the following:

1. If n is equal to 2, print `A`.
2. If n is equal to 4, print `A`.
3. If n is equal to 6, print `A`.
4. If n is equal to 3, print `B`.
5. If n is equal to 5, print `B`.
6. If n is equal to 7, print `B`.
7. For all other values of n, print `C`.

Input Format

A single integer denoting n.

```
var input = "";
process.stdin.on('data', function (data) {
    input = data;
    switchDemo();
});
function readLine() { return input; }
/**** Ignore above this line. ****/

function switchDemo() {
    var n = +(readLine());

    switch (n) {
        case 2:
        case 4:
        case 6:
            console.log("A");
            break;
```

Go to Top

```
        case 3:
        case 5:
        case 7:
            console.log("B");
            break;
        default:
            console.log("C");
    }

    console.log("Exited switch.");
```

Input

4

Run

Output

Run the code above with the given input, and then try replacing that input with other integers and seeing how it changes.

- IF-ELSE CONDITIONAL STATEMENTS Recommended Article

If-Else Conditional Statements View

String Basics

JavaScript Strings

These are chains of zero or more Unicode characters (i.e., letters, digits, and punctuation marks) used to represent text.

We denote string literals by enclosing them in single (`'`) or double (`"`) quotation marks. Double quotation marks can be contained in strings surrounded by single quotation marks (e.g., `'"'` evaluates to `"`), and single quotation marks can be contained in strings surrounded by double quotation marks (e.g., `"'"` evaluates to `'`). In addition, you can also enclose a single or double quotation within another quotation of its same type by preceding the quotation you wish to have interpreted literally with the escape character (`\`).

Each string has a property called `String.length` denoting the length of, or number of characters in, the string. For example, if we write `var s = "Hello."`, the value of `s.length` is **6**.

- EXAMPLE

```
var input = "";
process.stdin.on('data', function (data) {
    input = String(data).trim();
    main();
});
/** Ignore above this line. **/

function main() {
    console.log("\"" + input + "\"");
    console.log('\'' + input + '\'');
    console.log(input.length);
```

Table Of Contents

JavaScript Strings

String Constructor

`String.charAt()`

`String.concat()`

`String.includes()`

`String.endsWith()`

`String.indexOf()`

`String.lastIndexOf()`

`String.match()`

`String.normalize()`

`String.repeat()`

`String.replace()`

`String.search()`

`String.slice()`

`String.split()`

`String.startsWith()`

`String.substr()`

`String.substring()`

Go to Top

Input

HackerRank

Run

Output

`String.toLowerCase()`

`String.toUpperCase()`

`String.trim()`

`String.trimLeft()`

`String.trimRight()`

String Constructor

To create a new string, we use the syntax `String(value)` where ***value*** denotes the data we want to turn into a string.

Click *Run* below to see the string constructor in code.

```
var myNumber = 4;
var myString = String(myNumber);

console.log(myNumber + " is a " + typeof myNumber);
```

Output

Run

Observe that the value `4` assigned to the ***myNumber*** variable is typed as a *number*, and that the ***myString*** variable is typed as a string because we passed ***myNumber*** to the string constructor.

Methods

Here are some frequently used methods that operate on strings:

`String.charAt()`

Returns the character at the specified index. For example:

```
var input = "";
process.stdin.on('data', function (data) {
    input = data;
    main();
});
function readLine() { return input; }
/** Ignore above this line. **/
var s = "HackerRank";
var i = +(readLine());
```

Input

6

Run

Output

Index:	0	1	2	3	4	5	6	7	8	9
String:	H	a	c	k	e	r	**R**	a	n	k

Go to Top

`String.concat()`

Returns a new string consisting of the calling string concatenated with another string passed as an argument. For example:

```
var s = "Hacker";
var t = "Rank";
var u = s.concat(t);
console.log(s + " " + t);
```

Output

Run

`String.includes()`

Returns a boolean denoting whether a string passed as an argument exists within the calling string. For example:

```
var s = "HackerRank";
console.log(s.includes() + " " + s.includes(""));
```

Output

Run

`String.endsWith()`

Returns a boolean denoting whether the calling string ends with the characters of another string passed as an argument. For example:

```
var s = "HackerRank";
console.log(s.endsWith() + " " + s.endsWith(""));
```

Output

Run

`String.indexOf()`

Returns an integer denoting the index within the calling String object of the *first* occurrence of the given argument:

- If the argument isn't found in the string or no argument was passed to the function, it returns `-1`.
- If an integer is passed as a second argument, it will start searching the string from the index denoted by the integer.

For example:

```
var s = "HackerRank";
console.log(
    s.indexOf("a") + " "
    + s.indexOf("a", s.length) + " "
    + s.indexOf("a", 6)
);
console.log(
    s.indexOf("an") + " "
    + s.indexOf("x") + " "
```

Go to Top

```
    + s.indexOf("")
```

Output

Run

String.lastIndexOf()

Returns an integer denoting the index within the calling String object of the *last* occurrence of the given argument:

- If the argument isn't found in the string or no argument was passed to the function, it returns `-1`.
- If an integer is passed as a second argument, it will search the string backward starting from the index denoted by the integer.

For example:

```
var s = "HackerRank";
console.log(
    s.lastIndexOf("a") + " "
    + s.lastIndexOf("a", s.length) + " "
    + s.lastIndexOf("a", 6)
);
console.log(
    s.lastIndexOf("ac") + " "
    + s.lastIndexOf("x") + " "
    + s.lastIndexOf("")
```

Output

Run

String.match()

Match a regular expression passed as an argument against the calling string. If a match is found, it returns an object with three properties: the matched substring, the `index` it was found at, and the `input` (i.e., the initial string); if no match is found, it returns null. For example:

```
var s = "HackerRank";
console.log(s.match());
```

Output

Run

String.normalize()

Returns a string containing the Unicode Normalization Form of the calling string's value. The argument must be one of the following:

- `"NFC"`: Normalization Form Canonical Composition. This is the default in the event that no argument is given.
- `"NFD"`: Normalization Form Canonical Decomposition.
- `"NFKC"`: Normalization Form Compatibility Composition.
- `"NFKD"`: Normalization Form Compatibility Decomposition.

For example:

Go to Top

```
var s = "HackerRank";
console.log(s.normalize());
```

Output

Run

String.repeat()

Returns a string consisting of the elements of the calling String object repeated some number of times (given as an integer argument). If no argument or a 0 argument are given, then it returns the empty string. For example:

```
var s = "HackerRank";
console.log(s.repeat () + "x" + s.repeat(0));
```

Output

Run

String.replace()

Finds a match between a regular expression and a string, then returns a string where the first matched substring is replaced with a new substring. If no match is found, the returned string is the same as the original string. For example:

```
var s = "HackerRank";
```

Output

Run

String.search()

Executes the search for a match between a regular expression and a specified string, then returns the index of the first character of the first match. For example:

```
var s = "HackerRank";
console.log(s.search() + " " + s.search("[a-z]"));
```

Output

Run

String.slice()

Extracts a section of a string and returns it as a new string. The extracted section depends on the arguments passed to the function:

- If no arguments are passed to the function, it returns the entire string.
- If one integer argument, i, is passed to the function, it returns a substring starting at index i and ending at the end of the string.
- If two integer arguments, i and j, are passed to the function, it returns a substring consisting of characters in the range $[i, j)$; in other words, this is a substring starting at index i and ending at $j - 1$. For example:

If one (or both) of the arguments passed to this function is negative, then the index

Go to Top

corresponding to that argument is calculated as `String.length` minus the given argument. For example:

```
var s = "HackerRank";
console.log(s.slice(0, 6) + " " + s.slice(6));
console.log(
    s.slice() + " "
    + s.slice(-4, 8) + " "
    + s.slice(-4, -2)
```

Output

Run

`String.split()`

Splits a String object into an array of strings by separating the string into substrings:

- If no argument is given, it returns an array containing the original string.
- If the empty string is passed as an argument, it returns an array of the string's individual letters.
- If a string consisting of one or more letters is passed as an argument, it splits the string at each occurrence of that string and returns an array of the split substrings.

For example:

```
var s = "HackerRank";
console.log(s.split(""));
console.log(s.split("k"));
```

Output

Run

`String.startsWith()`

Returns a boolean denoting whether a string begins with the characters of another string passed as an argument. For example:

```
var s = "HackerRank";
console.log(s.startsWith("Hack"));
```

Output

Run

`String.substr()`

Returns a substring consisting of characters in a given range, depending on the arguments passed to the function:

- If no arguments are passed to the function, it returns the entire string.
- If one integer argument, i, is passed to the function, it returns a substring starting at index i and ending at the end of the string.
- If two integer arguments, i and j, are passed to the function, it returns a substring consisting of characters in the range $[i, j)$; in other words, this is a substring starting at index i and ending at $j - 1$.

For example:

Go to Top

```
var s = "HackerRank";
console.log(s.substr());
```

Output

Run

String.substring()

Returns a substring consisting of characters in a given range, depending on the arguments passed to the function:

- If no arguments are passed to the function, it returns the entire string.
- If one integer argument, i, is passed to the function, it returns a substring starting at index i and ending at the end of the string.
- If two integer arguments, i and j, are passed to the function, it returns a substring consisting of characters in the range $[i, j)$; in other words, this is a substring starting at index i and ending at $j - 1$.

For example:

```
var s = "HackerRank";
console.log(s.substring());
```

Output

Run

String.toLowerCase()

Returns the calling string's value, converted to lowercase letters. For example:

```
var s = "HackerRank";
s = s.toLowerCase();
```

Output

Run

String.toUpperCase()

Returns the calling string's value, converted to uppercase letters. For example:

```
var s = "HackerRank";
s = s.toUpperCase();
```

Output

Run

String.trim()

Returns the string with the whitespace trimmed from its beginning and end. This is part of the *ECMAScript 5* standard. For example:

```
var s = "    HackerRank    ";
console.log("Original Length: " + s.length);
```

Go to Top

```
s = s.trim();
```

Output

Run

String.trimLeft()

Returns the string with the whitespace trimmed from its left side. For example:

```
var s = "    HackerRank    ";
console.log("Original Length: " + s.length);
s = s.trimLeft();
```

Output

Run

String.trimRight()

Returns the string with the whitespace trimmed from its right side. For example:

```
var s = "    HackerRank    ";
console.log("Original Length: " + s.length);
s = s.trimRight();
```

Output

Run

- DATA TYPES — Recommended Article

Data Types — View

Related challenge for String Basics

Day 3: Try, Catch, and Finally

Success Rate: 99.01% Max Score: 15 Difficulty:

Solve Challenge