STRINGS IN JAVASCRIPT

Sai Vardhan T

STRINGS

- JavaScript strings are sequences of characters, used to represent text.
- They are a primitive data type in JavaScript.

CREATING STRINGS

- You can create strings in JavaScript using either single quotes(‘ ‘) or double quotes(“ “).

```
let singleQuotedString = 'Hello, World!';
let doubleQuotedString = "Hello, World!";
```

STRING LENGTH

- You can get the length of a string using the ‘length’ property.

```
let greeting = 'Hello, World!';
console.log(greeting.length); // Output: 13
```

ACCESSING CHARATERS

- You can access individual characters of a string using bracket notation with the index.

```
let greeting = 'Hello, World!';
console.log(greeting[0]); // Output: 'H'
console.log(greeting[7]); // Output: 'W'
```

CONCATENATIONG STRINGS

- You can concatenate (combine) strings using the '+' operator or the 'concat()' method.

```
let firstName = 'John';
let lastName = 'Doe';
let fullName = firstName + ' ' + lastName;
console.log(fullName); // Output: 'John Doe'

let greeting = 'Hello';
let name = 'Alice';
let message = greeting.concat(', ', name, '!');
console.log(message); // Output: 'Hello, Alice!'
```

STRING METHODS

- JavaScript provides a variety of built-in methods for working with strings
  - charAt()
  - charCodeAt()
  - Concat()
  - Startswith()
  - Endswith()
  - Includes()
  - Search()
  - toLowerCase()
  - toUpperCase()
  - indexOf()
  - lastIndexOf()
  - Match()
  - Repeat()
  - Replace()
  - Slice()
  - split()
  - Trim()

charAt()

- Returns the character at the specified index in a string

```
let str = "Hello";
console.log(str.charAt(0)); // Output: 'H'
console.log(str.charAt(4)); // Output: 'o'
```

charCodeAt()

- Returns the Unicode value of the character at the specified index in a string.

```
let str = "Hello";
console.log(str.charCodeAt(0)); // Output: 72
console.log(str.charCodeAt(1)); // Output: 101
```

Concat()

- Concatenates one or more strings to the end of another string and returns the combined string.

```
let str1 = "Hello";
let str2 = "World";
console.log(str1.concat(" ", str2)); // Output: 'Hello World'
```

endsWith()

- Checks if a string ends with the specified characters and returns true or false

```
let str = "Hello, World!";
console.log(str.endsWith("World!")); // Output: true
```

Startswith()

- Check if a string starts with the specified characters and returns true or false.

```
let str = "Hello, World!";
console.log(str.startsWith("Hello")); // Output: true
```

Includes()

- Checks if a string contains the specified characters and returns true or false.

```
let str = "Hello, World!";
console.log(str.includes("World")); // Output: true
```

indexOf()

- Returns the index within the calling string object of the first occurrence of the specified value.

```
let str = "Hello, World!";
console.log(str.indexOf("o")); // Output: 4
```

lastIndexOf()

- Returns the index within the calling string object of the last occurrence of the specified value

```
let str = "Hello, World!";
console.log(str.lastIndexOf("o")); // Output: 8
```

Match()

- Searches a string for a match against a regular expression and returns the matches an array.

```
let str = "The rain in Spain falls mainly on the plain";
console.log(str.match(/ain/g)); // Output: ['ain', 'ain', 'ain']
```

Repeat()

- Returns a new string with a specified number of copies of the string it was called on.

```
let str = "Hello";
console.log(str.repeat(3)); // Output: 'HelloHelloHello'
```

Replace()

- Searches a string for a specified value or regular expression and replaces it with another value

```
let str = "Hello, World!";
console.log(str.replace("World", "Universe")); // Output: 'Hello, Universe!'
```

Search()

- Searches a string for a specified value or regular expression and returns the position of the match.

```
let str = "The rain in Spain falls mainly on the plain";
console.log(str.search("Spain")); // Output: 12
```

Slice()

- Extracts a section of a string and returns it as a new string.

```
let str = "Hello, World!";
console.log(str.slice(7, 12)); // Output: 'World'
```

Split()

- Splits a string into an array of substrings based on the specified separator.

```
let str = "Hello, World!";
console.log(str.split(", ")); // Output: ['Hello', 'World!']
```

Substring()

- Extracts the characters from a string between two specified indices and returns the new string.

```
let str = "Hello, World!";
console.log(str.substring(7, 12)); // Output: 'World'
```

toLowerCase()

- Converts all characters in a string to lowercase.

```
let str = "Hello, World!";
console.log(str.toLowerCase()); // Output: 'hello, world!'
```

toUpperCase()

- Converts all characters in a string to uppercase.

```
let str = "Hello, World!";
console.log(str.toUpperCase()); // Output: 'HELLO, WORLD!'
```

Trim()

- Removes whitespace from both ends of a string

```
let str = "   Hello, World!   ";
console.log(str.trim()); // Output: 'Hello, World!'
```

Regular Expressions in JS

Sai Vardhan T

Regular Expressions in JS

- Regular expressions provide a powerful way to search and manipulate text.
- Instead of saying "regular expression", people often shorten it to "regex" or "regexp".
- A regular expression consists of:
  - A pattern you use to match text
  - Zero or more modifiers (also called flags) that provide more instructions on how the pattern should be applied
- JavaScript provides the RegExp() constructor which allows you to create regular expression objects.
- **Syntax:**
  - variable variable_name = new RegExp(pattern)

EXAMPLE

- **Example:**

```
var re = new RegExp("j.*t");
```

- There is also the more convenient regexp literal:

```
var re = /j.*t/;
```

- In the example above, **j.*t** is the regular expression pattern. It means, "Match any string that starts with j, ends with t, and has zero or more characters in between".
- The asterisk * means "zero or more of the preceding";
- the dot (.) means "any character". The pattern needs to be placed in quotation marks when used in a RegExp() constructor.

Properties of the RegExp Objects

- **global:**
  - If this property is false, which is the default, the search stops when the first match is found. Set this to true if you want all matches.
- **ignoreCase:**
  - Case sensitive match or not, defaults to false.
- **multiline:**
  - Search matches that may span over more than one line, defaults to false.
- **lastIndex:**
  - The position at which to start the search, defaults to 0.
- **source:**
  - Contains the regexp pattern.
- **None of these properties, except for lastIndex, can be changed once the object has created.**

Properties of the RegExp Objects

- The first three parameters represent the regex modifiers. If you create a regex object using the constructor, you can pass any combination of the following characters as a second parameter:
  - "g" for global
  - "i" for ignoreCase
  - "m" for multiline
- These letters can be in any order. If a letter is passed, the corresponding modifier is set to true. In the following example, all modifiers are set to true:

```
var re = new RegExp('j.*t', 'gmi');
```

Properties of the RegExp Objects

- Let's Verify once:

```
re.global;
//true
```

- Once set, the modifier cannot be changed:

```
re.global = false;
re.global
//true
```

- To set any modifiers using the regex literal, you add them after the closing slash.

```
var re = /j.*t/ig;
re.global
//true
```

Methods of the RegExp Objects

- The regex objects provide two methods you can use to find matches:
  - test()
  - exec().
- They both accept a string parameter.
- **test()**
  - returns a boolean (true when there's a match, false otherwise)
- **exec()**
  - returns an array of matched strings.

Methods of the RegExp Objects

- People often use regular expressions for validation purposes, in this case test() would probably be enough.

```
//No match, because of the capital J:
var re = /j.*t/;
re.test("Javascript");
//false

//Case insensitive test gives a positive result:var re = /j.*t/i;
re.test("Javascript");
//true
```

- The same test using exec() returns an array and you can access the first element as shown below:

```
var re = /j.*t/i;
re.exec("Javascript")[0]
//"Javascript"
```

String Methods that Accept Regular Expressions as Parameters

- Previously in this chapter we talked about the String object and how you can use the methods indexOf() and lastIndexOf() to search within text.
- Using these methods you can only specify literal string patterns to search. A more powerful solution would be to use regular expressions to find text.
- String objects offer you this ability.
- The string objects provide the following methods that accept regular expression objects as parameters:
  - **match()** returns an array of matches
  - **search()** returns the position of the first match
  - **replace()** allows you to substitute matched text with another string

Search()

- Let's see some examples of using the method search()
- First, we will create a string object.

```
var s = new String('HelloJavaScriptWorld');
```

- The search() method gives you the position of the matching string:

```
s.search(/j.*a/i);
//5
```

Match()

- Let's see some examples of using the method match().
- First, you create a string object.

```
var s = new String('HelloJavaScriptWorld');
```

- Using match() you get an array containing only the first match:

```
s.match(/a/);
//["a"]
```

- Using the g modifier, you perform a global search, so the result array contains two elements:

```
s.match(/a/g);
/*['a', 'a']*/
```

- Case insensitive match:

```
s.match(/j.*a/i);
//["Java"]
```

replace()

- Let's see some examples of using the method match().
- First, you create a string object.

```
var s = new String('HelloJavaScriptWorld');
```

- replace() allows you to replace the matched text with some other string.
- The following example removes all capital letters (it replaces them with blank strings):

```
s.replace(/[A-Z]/, '');
//"elloJavaScriptWorld"
```

- If you omit the g modifier, you're only going to replace the first match:

```
s.replace(/[A-Z]/g, '');
//"elloavacriptorld"
```

\d: Matches any digit (0-9).

- Example:
  - \d will match any single digit in a string.

```
const pattern = /\d/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\D: Matches any non-digit.

- Example:
  - \D will match any character that is not a digit.

```
const pattern = /\D/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\s: Matches any whitespace character

- Matches any whitespace character like spaces, tabs, newlines.
- Example:
  - \s will match any space character.

```
const pattern = /\s/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\S: Matches any non-whitespace character.

- Example:
  - \S will match any character that is not a space.

```
const pattern = /\S/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\w: Matches any word character (alphanumeric + underscore).

- Example:
  - \w+ will match one or more word characters.

```
const pattern = /\w/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\W: Matches any non-word character.

- Example:
- \W will match any character that is not a word character.

```
const pattern = /\W/g;
const text = 'The price is 25 Rupees';
const matches = text.match(pattern);
console.log(matches)
```

\b: Represents a word boundary.

- Example:
  - \bword\b will match the word "word" as a whole word.

```
const pattern = /\bprice/g;
const text = 'The price is 25 Rupees price';
const matches = text.match(pattern);
console.log(matches)
```

Meta Characters

- . (Dot): Matches any character except the newline character.

```
const re = /h.t/g;
const text = 'hat, pot, hit, haat';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- ^ (Caret): Matches the pattern only at the start of the string, indicating a "Starts With" condition.

```
const re = /^The/g;
const text = 'The price is 25 rupees';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- $ (Dollar): Matches the pattern only at the end of the string before the newline character, indicating an "Ends With" condition.

```
const re = /rupees$/g;
const text = 'The price is 25 rupees';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- (Asterisk): Matches zero or more occurrences of the preceding pattern.

```
const re = /ab*c/g;
const text = 'ac, abc, abbc, abd';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- (Plus): Matches one or more occurrences of the preceding pattern.

```
const re = /ab+c/g;
const text = 'ac, abc, abbc, abd';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- ? (Question mark): Matches zero or one occurrence of the preceding pattern.

```
const re = /colors?/g;
const text = 'colors, color';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- {} (Curly Braces): Matches the exactly specified number of occurrences of the preceding pattern.

```
const re = /\d{2}:\d{2}/g;
const text = 'the time is 10:20';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- [] (Bracket): Defines a set of characters, and the pattern matches any one of the characters within the set.

```
const re = /[ch]at/g;
const text = 'The cat wear hat';
const matches = text.match(re);
console.log(matches);
```

Meta Characters

- | (Pipe): Acts as an OR operator, allowing the pattern to match either of the defined patterns.

```
const re = /cat|hat/g;
const text = 'The cat wear hat';
const matches = text.match(re);
console.log(matches);
```

Program to Validate only Numerical input:

- By using regular expressions we have checked whether the user is entering only numbers in the given input field or not
- If the user doesn't enter the number, an alert message will be activated.

```
<html>
    <head>
        <script>
            function valid(){
                var val = document.getElementById('val').value;
                var x = /\d/g;
                if(x.test(val)){
                    alert("Thank you for your valid input");
                }
                else{
                    alert("Please eneter numbers only");
                }
            }
        </script>
</head>
<body>
    <form>
        Enter Age : <input type='text' id='val'/>
        <input type='button' onclick="valid()" value='Submit' />
    </form>
</body>
</html>
```