Regular Expressions, External JS

ITP 301
Spring 2021

Strings

Strings are series of characters.

- Sometimes treated like arrays with letters in each slot.

0	1	2	3	4	5	6	7	8	9	10	11
H	e	l	l	o		W	o	r	l	d	!

does not change the original string,
but just return a copy of the replaced version

`.replace(`*find*`,` *replace*`)`	Find first occurrence of *find* and replace it with *replace*.
`.substr(`*start*`,` *[length]*`)`	Return substring starting at index *start*. Length of substring is optional.

```
var myString = "Hello World!";

console.log( myString.replace('Hello', 'Hi') );
// Returns "Hi World!"

console.log( myString.substr(6) );
// Returns "World!"

console.log( myString.substr(0, 5) );
// Returns "Hello"
```

Regular Expressions

Regular Expressions (RegEx or RegExp) are **patterns** used to search through strings.

Two ways to create RegEx:

1. Literal Syntax
2. Constructor Function Syntax

```
var literal = /search_pattern/;

var constructor = new RegExp('search_pattern');
```

Use Constructor Function Syntax when pattern is stored in a variable.

```
var pattern = 'search_pattern';

var constructor = new RegExp(pattern);
```

Related Functions

String Functions:

`.search( regEx)`	Return index of first *regEx* occurrence. If substring is not found, return `-1`.
`.replace( regEx, replace)`	Find substring(s) matching *regEx* and replace it / them with *replace*.

RegExp Functions:

`.test( string)`	Test if RegEx matches *string*. Returns `true` if match is found, `false` otherwise.

```
var myString = "Hello World!";

console.log( myString.search(/o/) );     // Outputs '4'.
console.log( myString.search(/llo/) );   // Outputs '2'
console.log( myString.search(/Hi/) );    // Outputs '-1'

console.log( myString.replace(/Hello/, 'Hi') );
// Outputs 'Hi World!'



var regExp = /Hello/;
console.log( regExp.test('Hello World!') );
// Outputs 'true'.

console.log( /hello/.test('Hello World!') );
// Outputs 'false'.
```

RegEx Options

Flags / Modifiers:

`i`	Case-insensitive search.
`g`	Global search (find **all** matches).

Special Characters:

`^`	Beginning of string.
`$`	End of string.
`.`	Any character.
`\d`	Digit.
`\s`	Space character.
`\w`	Word character.

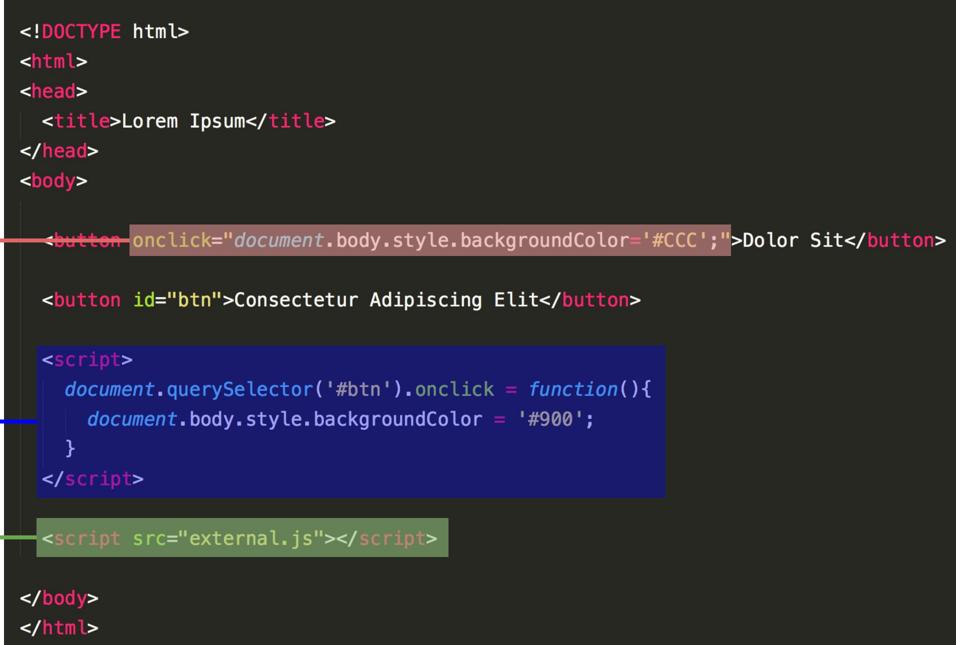
`[abc]`	Any character between brackets.
`[a-z]`	Any character between the range.
`(a|b)`	*a* or *b*.
`a{n}`	*n* occurrences of *a*.
`a{n,}`	At least *n* occurrences of *a*.

```
// Case-insensitive pattern for 'hello'.
var regExp = new RegExp('hello', 'i');

// Case-insensitive pattern for ALL instances of 'hello'.
/hello/ig;


// Pattern for empty string.
/^$/;

// Pattern for ALL digits.
/\d/g;

// Pattern for lower-case characters.
/[a-z]/;

// Case-insensitive pattern for 'Trojan' and 'Trojans'.
/trojans{0,1}/i;
```

JavaScript Types

Just like CSS, there are 3 types of JavaScript:

1. Inline
2. Internal
3. External



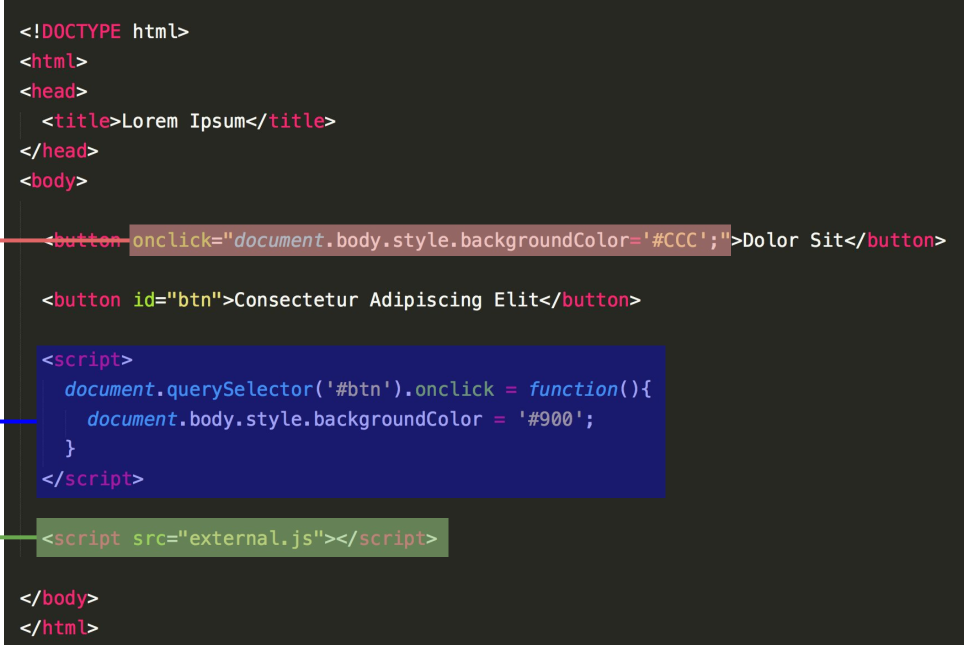
```html
<!DOCTYPE html>
<html>
<head>
  <title>Lorem Ipsum</title>
</head>
<body>

  <button onclick="document.body.style.backgroundColor='#CCC';">Dolor Sit</button>

  <button id="btn">Consectetur Adipiscing Elit</button>

  <script>
    document.querySelector('#btn').onclick = function(){
      document.body.style.backgroundColor = '#900';
    }
  </script>

  <script src="external.js"></script>

</body>
</html>
```

JavaScript Types

Just like CSS, there are 3 types of JavaScript:

1. ~~Inline~~
2. Internal
3. External

Just like inline CSS, avoid using inline JS.



```html
<!DOCTYPE html>
<html>
<head>
  <title>Lorem Ipsum</title>
</head>
<body>

  <button onclick="document.body.style.backgroundColor='#CCC';">Dolor Sit</button>

  <button id="btn">Consectetur Adipiscing Elit</button>

  <script>
    document.querySelector('#btn').onclick = function(){
      document.body.style.backgroundColor = '#900';
    }
  </script>

  <script src="external.js"></script>

</body>
</html>
```