KINGSLAND UNIVERSITY

Strings and RegExp

String Operations and Regular Expressions

Table of Contents

- Strings
  - Definition
  - Comparing strings
  - Methods
- RegExp
  - Definition
  - Patterns
  - Methods

Operations, Comparison and Methods

Strings

What is a String?

- Strings are used for **storing** and **manipulating** text

```
let str = "Hello, World!";
```

- The **+** operator can be used to **append multiple strings** together

```
let longString = "This is a very long string" +
                 "to wrap across multiple lines" +
                 "otherwise my code is unreadable."
```

Quotes in Strings

- There is **no distinction** between **single-quoted** strings and **double-quoted** strings in JavaScript

```
let carName = "Volvo XC60"; // Double quotes
let carName = 'Volvo XC60'; // Single quotes
```

- Quotes can be used inside a string, as long as they **don't match** the quotes surrounding the string

```
let str1 = "It's alright";
let str2 = "He is called 'Johnny'";
let str3 = 'He is called "Johnny"';
```

Length and Special Characters

- The length of a string is found in the built-in property **length**

```
let myStr = "Find my length.";
let length = myStr.length; // 15
```

- **Special characters** can be encoded using **escape notation**

Code	Result	Description
¥'	'	Single quote
¥''	"	Double quote
¥¥	¥	Backslash

Escape Sequences

Code	Result
¥b	Backspace
¥f	Form feed
¥n	New Line

Code	Result
¥r	Carriage Return
¥t	Horizontal Tabulator
¥v	Vertical Tabulator

```
let example = "This is an example ¥nfor a new line.";
// This is an example
// for a new line.
```

Comparing Strings

- Equality ("**==**") - True if **operands** are the same, otherwise false

```
let str = "example";
if (str == "example") // true
```

- Strict equality ("**===**") **-** True if **operands** and **data type** are the same, otherwise false

```
let str2 = new String("example");
if (str === str2) // not true
```

Comparing Strings (2)

- Inequality ("**!=**") - True if **operands** are **not the same**, otherwise false

```
let string = "9900";
let number = 9900
if (string != number) // false
```

- Strict inequality ("**!==**") - True if **operands** and **data types** are **not the same**, otherwise false

```
if (string !== number) // true
```

Comparing Strings (3)

- Greater than - "**>**" (Greater than or equal - "**>=**")
  - True if first operand is greater than (or equal to) the second one

```
if (9 > 5) // true
```

- Less than - "**<**" (Less than or equal - "**<=**")
  - True if second operand is greater than (or equal to) the first one

```
if ('Example of a long string' <= 'A short one') // false
```

String Methods

- **indexOf()** - returns the position of the first found occurrence of a specified value in a string

```
let str = "JavaScript is fun!";
console.log(str.indexOf("JavaScript")); // 0
console.log(str.indexOf("java")); // -1
```

- **slice()** - extracts a part of a string and returns a new one

```
let str = "Hello world!";
let res = str.slice(0, 5);  // Hello
```

String Methods

- **substring()** - extracts the characters from a string between two specified **indices**

```
let str = "I am JavaScript developer";
let sub = str.substring(5, 9); // Java
```

- **substr()** - extracts the characters from a string from a start position and through specified **length**

```
let str = "I am JavaScript developer";
let sub = str.substr(5);  // JavaScript developer
```

String Methods

- Accessing elements like an array

```
let str = "JavaScript is fun!";
let letter = str[0];
console.log(letter); // J
```

```
let str = "JavaScript is fun!";
let letter = str.charAt(0);
console.log(letter); // J
```

- Converting string to an array with the split method

```
let str = "I like JS";
let tokens = str.split(' ');
console.log(tokens); // ["I", "like", "", "", "", "JS"]
tokens = tokens.filter(s => s != '');
console.log(tokens.join(' ')); // I like JS
```

The Beauty of Modern String Processing

Regular Expressions

What Are Regular Expressions?

- Patterns used to **match** character **combinations** in **strings**
- RegExp in **string methods**
  - **/i** - makes the regex match **case insensitive**

```
let str = "RegExp Example";
let search = str.search(/RegExp/i) // 0
```

  - **/g** - replaces **all** matches

```
let str = "Java Regex Example Java";
let search = str.replace(/Java/g, "JavaScript");
// JavaScript RegExp Example JavaScript
```

Patterns

- **Patterns** are defined by special syntax
  - **`[0-9]+`** - matches non-empty sequence of digits
  - **`[A-Z][a-z]*`** - matches a capital + small letters
  - **`¥s+`** - matches whitespace (non-empty)
  - **`¥S+`** - matches non-whitespace
  - **`[0-9]{3,6}`** - matches 3-6 digits
  - **`¥d+`** - matches digits
  - **`¥D+`** - matches non-digits
  - **`¥w+`** - matches letters
  - **`¥W+`** - matches non-letters

RegEx Brackets

- Very useful for grouping words and ranges of letters and numbers

[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any non-digit between the brackets
(x|y)	Find any of the alternatives specified

Quantifiers

- **n+** - matches any string that contains **at least one** n
- **n*** - matches any string that contains **zero or more** occurrences of n
- **n?** - matches any string that contains **zero or one** occurrences of n
- **n{X}** - matches any string that contains **a sequence of X** n's

Quantifiers (2)

- **n{X,Y}** - matches any string that contains **a sequence of X to Y** n's
- **n{X,}** - matches any string that contains a **sequence of at least X** n's
- **n$** - matches any string with n **at the end** of it
- **^n** - matches any string with n **at the beginning** of it

RegEx Methods

- **exec()** - used to execute the search for a match in a specified string

```
let namePattern = (/[A-Z][a-z]+/g);

let names = "Jack Mason, example, Example";

let match;

while(match = namePattern.exec(names)) {
   console.log(match[0]);
}

// Jack

// Mason

// Example
```

RegEx Methods

- **match()** - retrieves the result of matching a string against a regular expression

```
let namePattern = (/[A-Z][a-z]+/g);
let names = "Jack Mason, example, Example";
let match = names.match(namePattern);
console.log(match) // ["Jack","Manson","Example"]
```

- **test()** - returns **true** or **false**

```
let pattern = (/[0-9]+/g);
let str = "Jack Mason";
console.log(pattern.test(str)); // false
```

Live Exercise

Summary

- Strings are used for **storing** and **manipulating** text
- Special characters can be encoded using **escape notation**
- Regular expressions are **patterns** used to **match** character combinations in **strings**
- Patterns are defined by **special syntax**

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com

KINGSLAND UNIVERSITY

THANK YOU