

# Strings

## Quick overview

Shadi Lahham - Programmazione web - Frontend - Javascript

# Strings

# What are strings

- Strings in JavaScript are used to manipulate texts and characters
- Can be used to process:
  - names, addresses, phone numbers, ID, company names, product codes, serial numbers etc.
- Can contain:
  - Alphanumeric characters (letters, numbers)
  - Special character such as #,@,\$,!,&\*,\,+, - etc.
- Strings are zero-indexed:
  - The index of the first character is 0, the second character 1 and so on

# Quick example

You can use single or double quotes  
Pick a style and stick with it!

```
// this is a string  
let client = "James";
```

```
// this is also a string  
let bestFriend = 'Robbie';
```

There are cases when it's useful to mix quotes:

```
let status = "It's raining";  
let answer = "The password is 'Bigfoot'";  
let alternative = 'The password is "Bigfoot"';
```

# Useful functions

Strings have many useful properties and functions:

```
// length
const alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let alphabetLength = alphabet.length;

// charAt()
let greeting = "HELLO WORLD";
let result = greeting.charAt(0);

// indexOf()
let statement = "Hello world, welcome to the universe.";
let wordPosition = statement.indexOf("welcome");
```

# String access

There are two ways to access characters in a string

```
// property access
const alphabetLowercase = "abcdefghijklmnopqrstuvwxyz";
console.log(alphabetLowercase [0]); // 'a'
```

```
// charAt()
let greeting = "HELLO WORLD";
let result = greeting.charAt(0);
```

Property access is unpredictable:

- does not work in old browsers
- makes strings look like arrays (confusing)
- if no character is found, [ ] returns undefined, charAt() returns an empty string
- is read only. alphabet[0] = "X" does not work and gives no errors
- don't use property access

# String reference

[JavaScript Strings](#)

[JavaScript String Reference](#)

Read carefully. You will need some string methods for the exercises.

Check browser compatibility before using since not all String methods are supported on all browsers

Note: You can check on [CanIUse](#) or [MDN](#)

# Regular expressions



# Regular expressions

## JavaScript RegExp Object

Regular expressions are also very useful for string manipulation

# Template Strings

# Template strings

```
const title = `Template strings are syntactic sugar`;
```

```
const message = `Can be  
on multiple  
lines`;
```

```
console.log(`Used almost anywhere strings are used, more or less`);
```

# Template strings

```
const name = 'james';  
const age = 25;
```

```
// Interpolate variable bindings  
console.log(`My name is ${name} I am ${age + 10}  
years old (lie)`);
```

```
let name = 'james';  
let age = 25;
```

```
// Interpolate variable bindings  
console.log('My name is '.concat(name, ' I am  
' ).concat(age + 10, ' years old (lie)'));
```

Your turn

# 1.Reverse

- Write a JavaScript function called `printReverse` which has one parameter, a string, and which prints that string in reverse.
- For example, the call `printReverse("foobar")` should result in "raboof" being displayed.
- **Note:** If you used Array methods in your solution, try to write the same function without using the array methods (submit separate files for each solution)

## 2.Reverse 2

- Write a JavaScript function called reverse which has one parameter, a string, and which returns that string in reverse.
- For example, the call reverse("foobar") should return the string "raboof".
- **Note:** If you used Array methods in your solution, try to write the same function without using the array methods (submit separate files for each solution)

## 3. Palindrome

- Using your `reverse()` function from the previous exercise, write a simple function to check if a string is a palindrome.
- A palindrome is a word that reads the same backwards as forwards. For example, the word "madam" is a palindrome.
- Write a JavaScript function called `isPalindrome` which has one parameter, a string, and which returns `true` if that string is a palindrome, else `false`.
- For example, the call `isPalindrome("madam")` should return `true`, while `isPalindrome("madame")` should return `false`.
- **Bonus:** Try to write the same function without using the `reverse()` function



## 4.Capital

- Write a JavaScript function called `capital` which has one parameter, a string, and which returns that string with the first letter capitalized.
- For example, the call `capital("hello world")` should return the string "Hello world".
- **Bonus:** modify the function so that it capitalizes each word.
- `capital2("my name is john")` should return the string "My Name Is John".

# 5.Money

- Create a function called Money
- It should take a single argument, an amount, and return '<amount> dollars'
- Add a smiley at the end if the amount is 1 million. Deal with edge cases
- For example:

```
Money(1): 10 dollar  
Money(10): 10 dollars  
Money(1000000): 1000000 dollars ;)
```

- **Bonus:** add to the function the ability to convert dollars to euros.

```
Money(10): 10 dollars are 9.31 euros
```

## 6.MixUp

- Create a function called mixUp.
- It should take in two strings, and return the concatenation of the two strings (separated by a space) slicing out and swapping the first 2 characters of each.
- You can assume that the strings are at least 2 characters long.
- For example:

```
mixUp('mix', 'pod'): 'pox mid'  
mixUp('dog', 'dinner'): 'dig donner'
```

## 7.FixStart

- Create a function called fixStart.
- It should take a single argument, a string, and return a version where all occurrences of its first character have been replaced with '\*', except for the first character itself.
- You can assume that the string is at least one character long.
- For example:

```
fixStart('babble'): 'ba**le'
```

Bonus

## 8. Verbing

- Create a function called verbing.
- It should take a single argument, a string. If its length is at least 3, it should add 'ing' to its end, unless it already ends in 'ing', in which case it should add 'ly' instead.
- If the string length is less than 3, it should leave it unchanged.
- For example:

```
verbing('swim'): 'swimming'  
verbing('swimming'): 'swimmingly'  
verbing('go'): 'go'
```

## 9. Not Bad

- Create a function called `notBad` that takes a single argument, a string.
- It should find the first appearance of the substring 'not' and 'bad'.
- If the 'bad' follows the 'not', then it should replace the whole 'not'...'bad' substring with 'good' and return the result.
- If it doesn't find 'not' and 'bad' in the right sequence (or at all), just return the original sentence.
- For example:

```
notBad('This dinner is not that bad!'): 'This dinner is good!'
notBad('This movie is not so bad!'): 'This movie is good!'
notBad('This dinner is bad!'): 'This dinner is bad!'
```

# 10.Contains

- Create a function called aContainsb.
- It should take in two strings, and return true if the first string contains the second, otherwise it should return false.
- For example:

```
aContainsB ("Another hello world", "hell");
```



# 11.The group

- Use the previous function to write another function called group that checks whether a string is part of another longer string that is a list of names of a group.
- The function should output the results to the console.

```
let group = "Mary, James, and John";
```

```
let oldGuy = "James";
```

```
// Outputs: "James IS part of the group"
```

```
let newGuy = "Philip";
```

```
// Outputs: "Philip is NOT part of the group"
```

## 12.Cut me up

- In the exercise folder include a .doc or an .md file in which you explain the difference between the following string methods:
  - `slice()`, `substring()` and `substr()`
  - Explain the differences in terms of parameters and behavior
  - Provide code examples to prove your point

# References

[JavaScript Strings](#)

[JavaScript String Reference](#)

[JavaScript RegExp Object](#)