CORSO DI LAUREA IN INFORMATICA

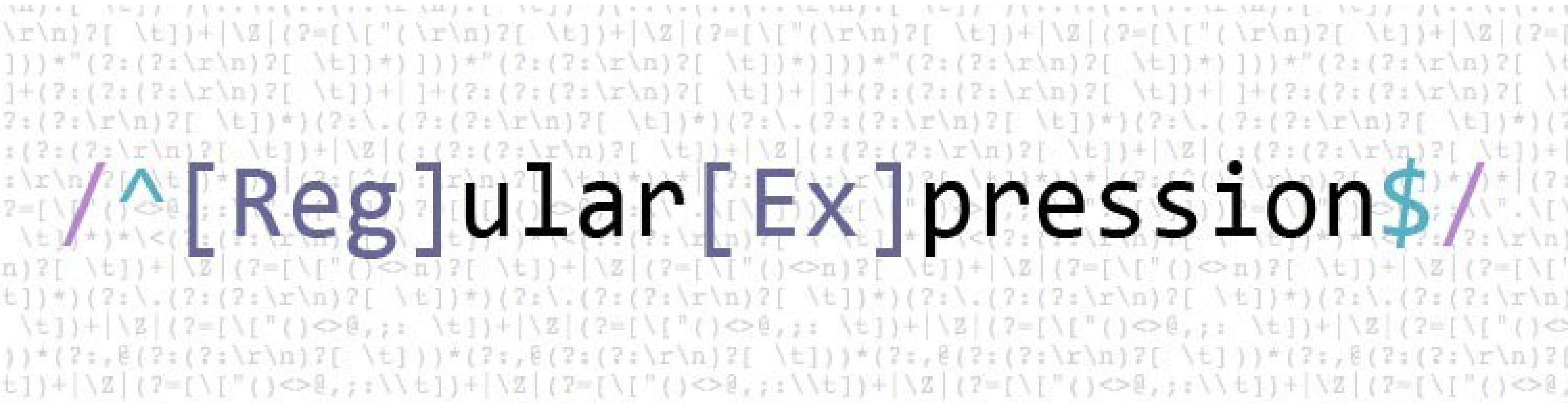
# Tecnologie Software per il Web

## REGULAR EXPRESSIONS

Docente: prof. Romano Simone
a.a. 2024-2025

## Regular expressions

- A **regular expression** is an object that describes a pattern of characters
- *Regular expressions are used to perform "pattern-matching" and "search-and-replace" functions on text*



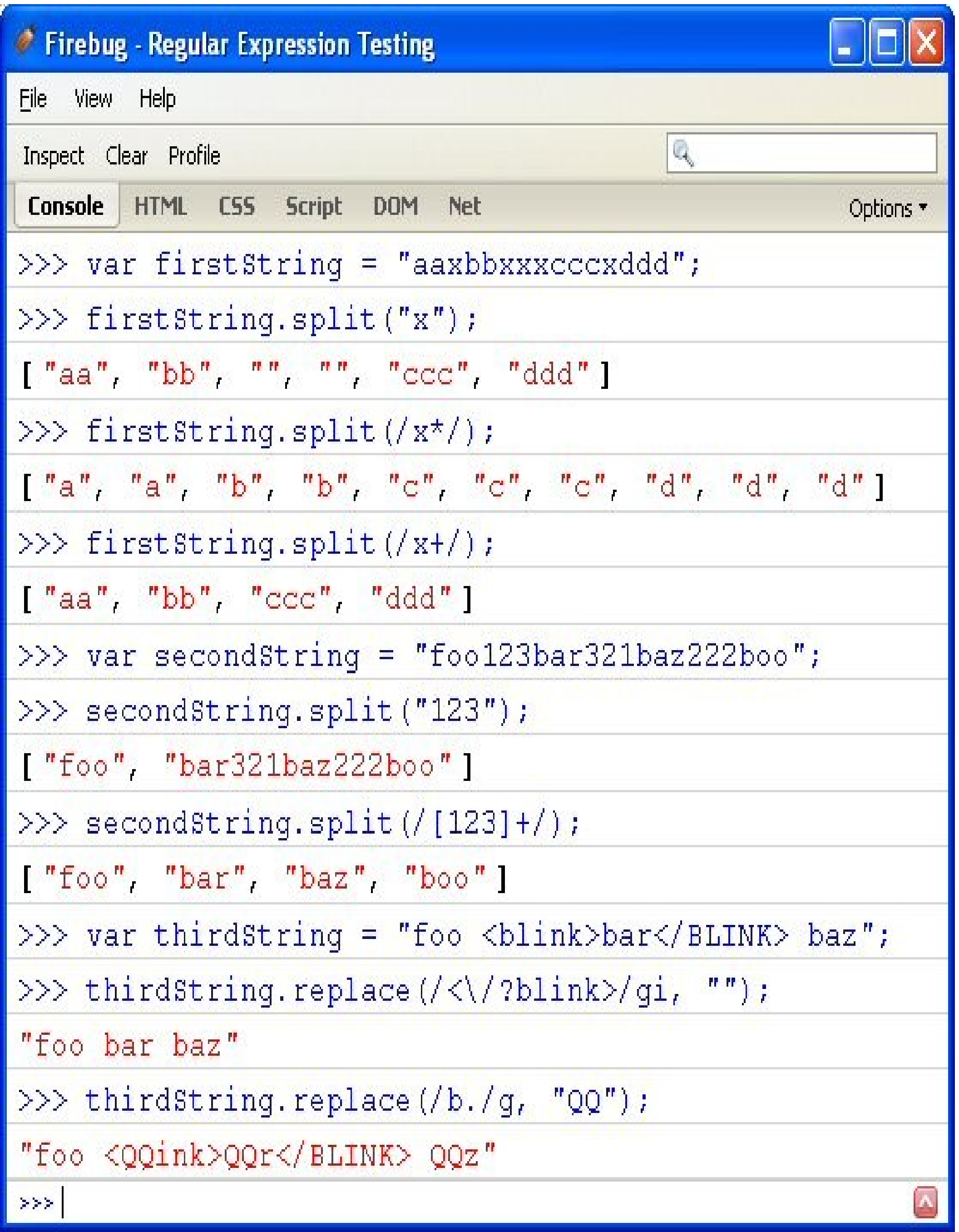
# Regular expressions: overview

- You specify a regex with /**pattern**/
  - *Not* with a string as in java and many other languages
- Most special characters same as in java/unix/perl
  - `^, $,` – beginning, end of string
  - `.` – any one char (except newline and line terminator)
  - `\` – escape what would otherwise be a special character
  - `*, +, ?` – 0 or more, 1 or more, 0 or 1 occurrences
  - `{n}, {n,}, {n, m}` – exactly n, n or more occurrences, from n to m occurrences
  - `[ ]` – grouping
  - `[^]` – not in the group
  - `\s, \S` – whitespace, non-whitespace
  - `\w, \W` – word char (letter or number), non-word char
  - `\d, \D` – number, a non-digit character
  - `(x | y)` – any of the alternatives specified
  - `?=n, ?!n` – any string followed by n, any string not followed by n
- Modifiers
  - `/Pattern/g` – do global matching (find all matches, not just first one)
  - `/Pattern/i` – do case-insensitive matching
  - `/Pattern/m` – do multiline matching

## String methods that use regular expressions

- Match
  - Returns array of parts of the string that match the regular expression
  - "A**x**b**xx**c**xxx**d".match(/x+/g) → ["x", "xx", "xxx"]
- Replace
  - Replaces all places that match the regular expression with a replacement string
  - "A**x**b**xx**c**xxx**d".replace(/x+/g, "q") → ”A**q**b**q**c**q**d"
- Split
  - Returns array of all parts of the string that are in between the regular expressions
  - "**A**x**b**xx**c**xxx**d**".split(/x+/) → ["A", "b", "c", "d"]
- Search
  - Returns the position of the first place that matches the regular expression
  - "A**x**bxxcxxxd".search(/x+/) → 1

## Example: regular expressions


Firebug - Regular Expression Testing
File View Help
Inspect Clear Profile
Console HTML CSS Script DOM Net Options

```
>>> var firstString = "aaxbbxxxcccxddd";
>>> firstString.split("x");
["aa", "bb", "", "", "ccc", "ddd"]
>>> firstString.split(/x*/);
["a", "a", "b", "b", "c", "c", "c", "d", "d", "d"]
>>> firstString.split(/x+/);
["aa", "bb", "ccc", "ddd"]
>>> var secondString = "foo123bar321baz222boo";
>>> secondString.split("123");
["foo", "bar321baz222boo"]
>>> secondString.split(/[123]+/);
["foo", "bar", "baz", "boo"]
>>> var thirdString = "foo <blink>bar</BLINK> baz";
>>> thirdString.replace(/<\/?blink>/gi, "");
"foo bar baz"
>>> thirdString.replace(/b./g, "QQ");
"foo <QQink>QQr</BLINK> QQz"
>>>
```

## Practical approach

- JavaScript code for validating user input

```
function validateInput(obj){
    var pattern = /.../;
    if(obj.value.match(pattern))  {
        //Do something: set class
        return true;
    }  else  {
        //Do something: set error class, focus, show error
            message, show suggestion,
        return false;
    }
}
```

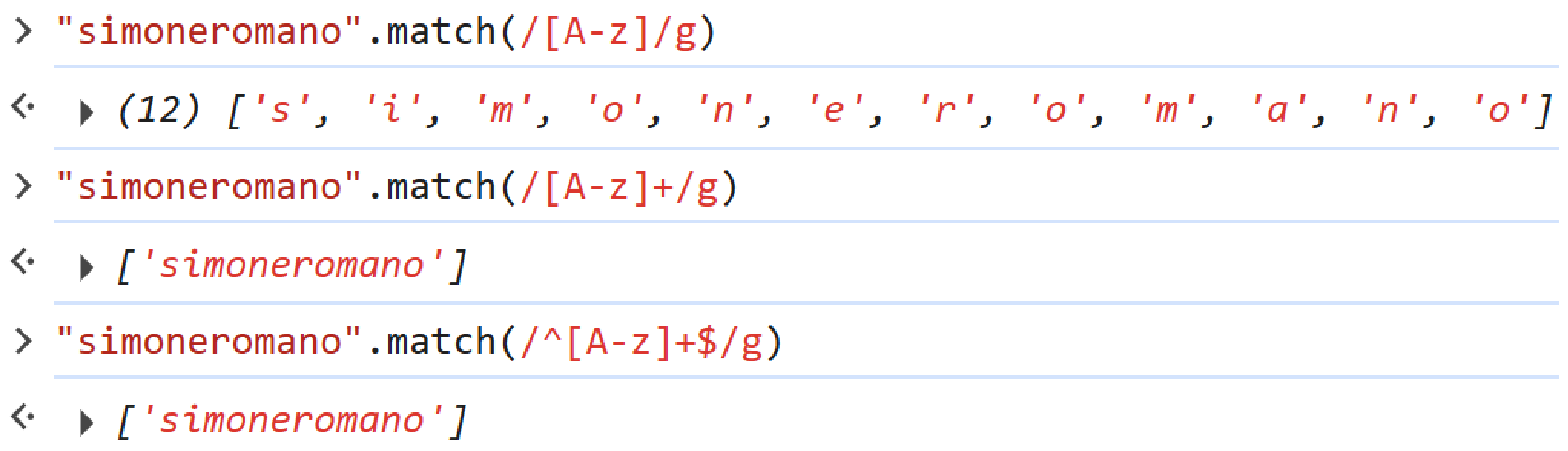
## Example 1: username

- JavaScript code for validating user name

```
function allLetter(uname){
        var letters = /^[A-z]+$/g;
        if(uname.value.match(letters)) {
                return true;
        } else {
                alert("Username must have alphabet
                        characters only");
                return false;
        }
}
```

[A-z] indica un qualsiasi lettera maiuscola o minuscola, mentre [a-z] indica una qualsiasi lettera minuscola e [A-Z] una qualsiasi lettera maiuscola

## Try step by step



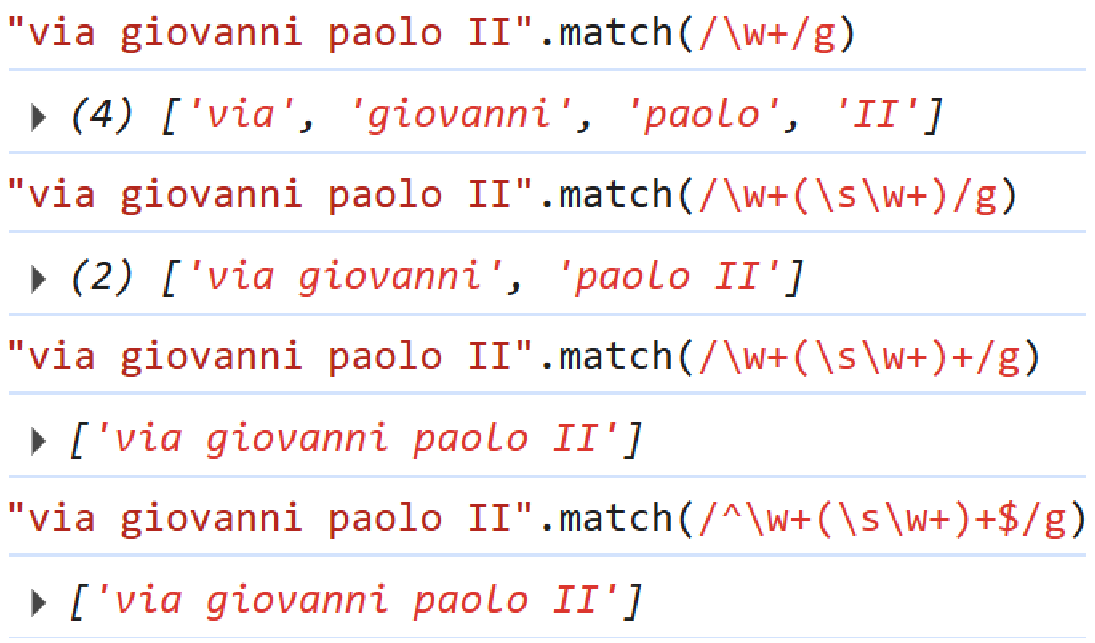
```
> "simoneromano".match(/[A-z]/g)
<· ▸ (12) ['s', 'i', 'm', 'o', 'n', 'e', 'r', 'o', 'm', 'a', 'n', 'o']
> "simoneromano".match(/[A-z]+/g)
<· ▸ ['simoneromano']
> "simoneromano".match(/^[A-z]+$/g)
<· ▸ ['simoneromano']
```

## Example 2: user address

JavaScript code for validating an address

```
function alphanumericAndSpaces(uadd){
        var letters =  /^\w+(\s\w+)+$/g;
        if(uadd.value.match(letters)){
                return true;
        } else {
                alert("User address must be made of
                        sequences of words or numbers
                        separated by whitespaces");
                return false;
        }
}
```

# Try step by step



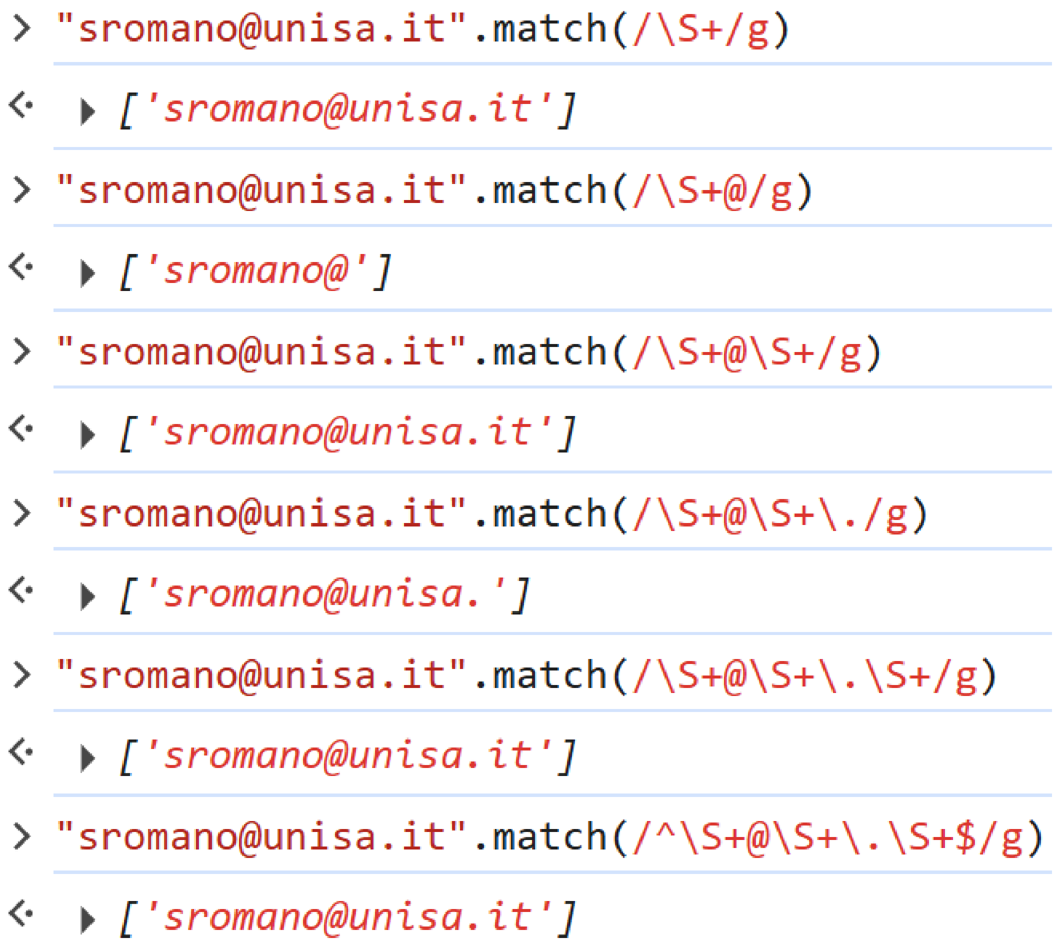
```
"via giovanni paolo II".match(/\w+/g)
▸ (4) ['via', 'giovanni', 'paolo', 'II']
"via giovanni paolo II".match(/\w+(\s\w+)/g)
▸ (2) ['via giovanni', 'paolo II']
"via giovanni paolo II".match(/\w+(\s\w+)+/g)
▸ ['via giovanni paolo II']
"via giovanni paolo II".match(/^\w+(\s\w+)+$/g)
▸ ['via giovanni paolo II']
```

## Example 3: email

- JavaScript code for validating an email (actually, the regexp for emails is more complex!!!)

```
function validateEmail(uemail){
    var mailformat = /^\S+@\S+\.\S+$/g;
    if(uemail.value.match(mailformat))  {
       return true;
    } else {
       alert("You have entered an invalid
              email address");
       return false;
    }
}
```

## Try step by step



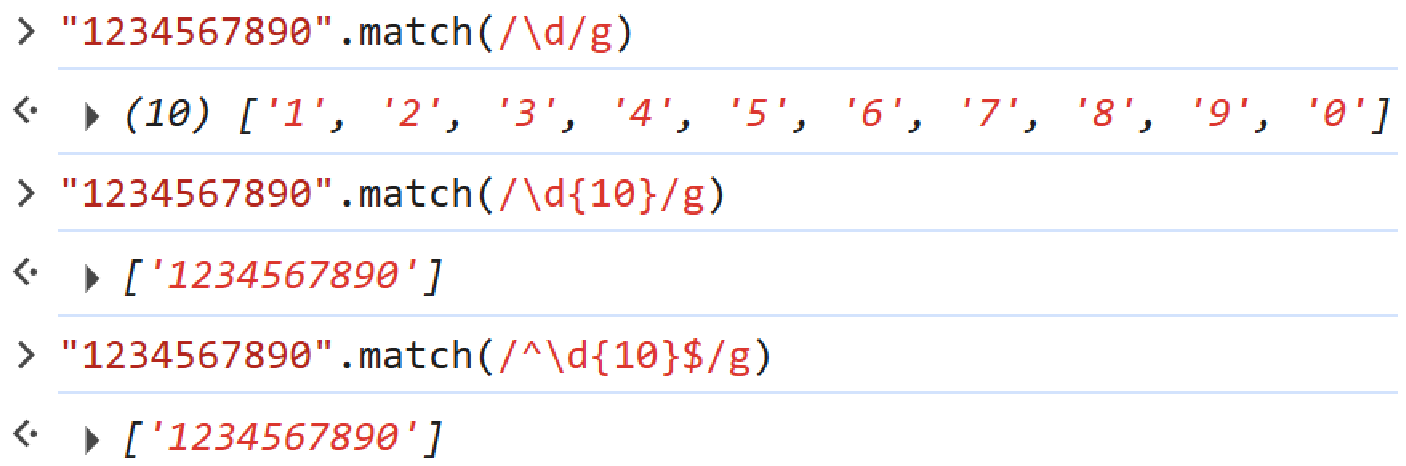
```
> "sromano@unisa.it".match(/\S+/g)
<· ▸ ['sromano@unisa.it']
> "sromano@unisa.it".match(/\S+@/g)
<· ▸ ['sromano@']
> "sromano@unisa.it".match(/\S+@\S+/g)
<· ▸ ['sromano@unisa.it']
> "sromano@unisa.it".match(/\S+@\S+\./g)
<· ▸ ['sromano@unisa.']
> "sromano@unisa.it".match(/\S+@\S+\.\S+/g)
<· ▸ ['sromano@unisa.it']
> "sromano@unisa.it".match(/^\S+@\S+\.\S+$/g)
<· ▸ ['sromano@unisa.it']
```

## Example 4: phone number (1)

- JavaScript code for validating a phone number made of 10 digits

```
function phoneNumber(inputtxt) {
  var phoneno = /^\d{10}$/;
  if((inputtxt.value.match(phoneno)) {
      return true;
  } else {
      alert("The numeric input is not valid");
      return false;
  }
}
```

# Try step by step



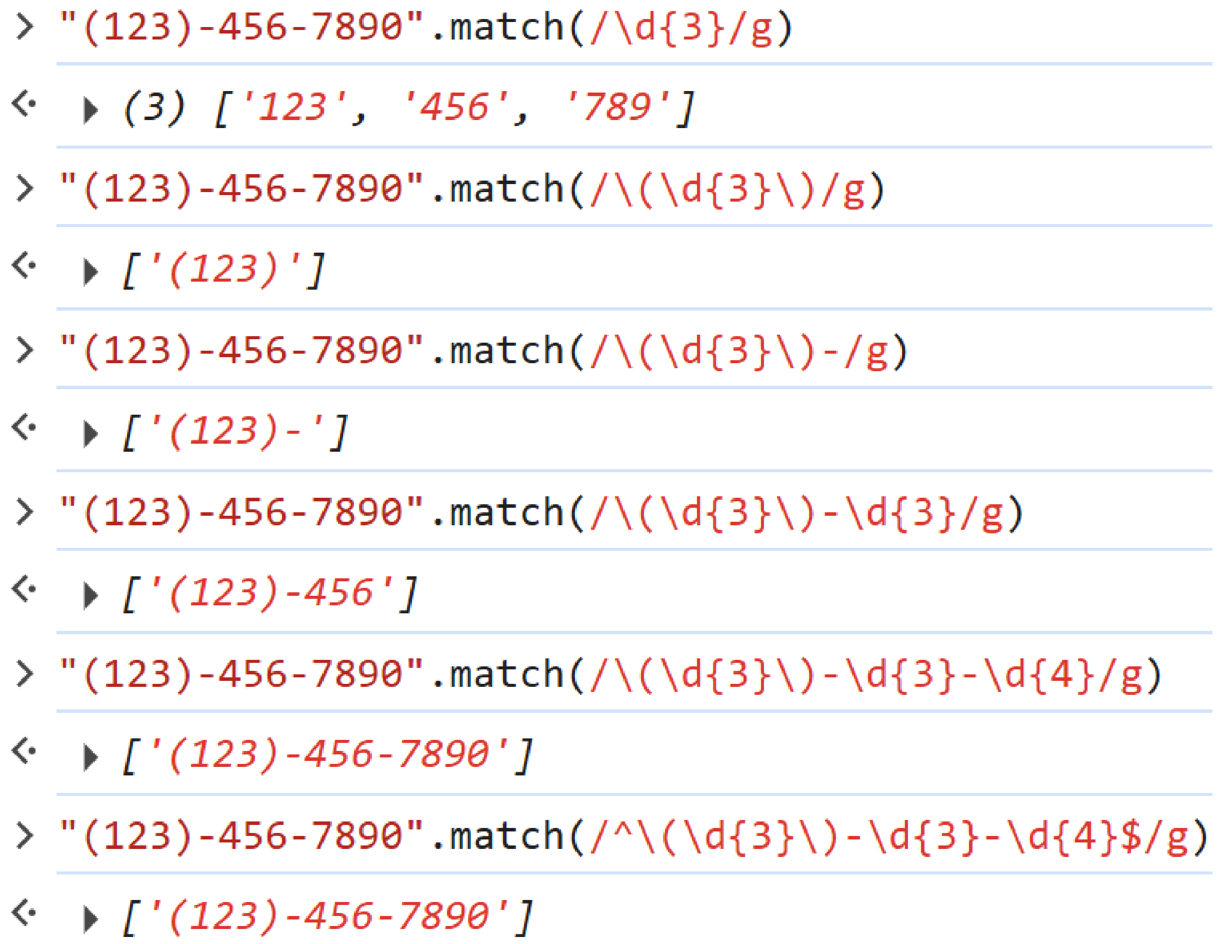
```
> "1234567890".match(/\d/g)
<· ▸ (10) ['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
> "1234567890".match(/\d{10}/g)
<· ▸ ['1234567890']
> "1234567890".match(/^\d{10}$/g)
<· ▸ ['1234567890']
```

## Example 4: phone number (2)

- JavaScript code for validating a phone number having the following format: (XXX)-XXX-XXXX

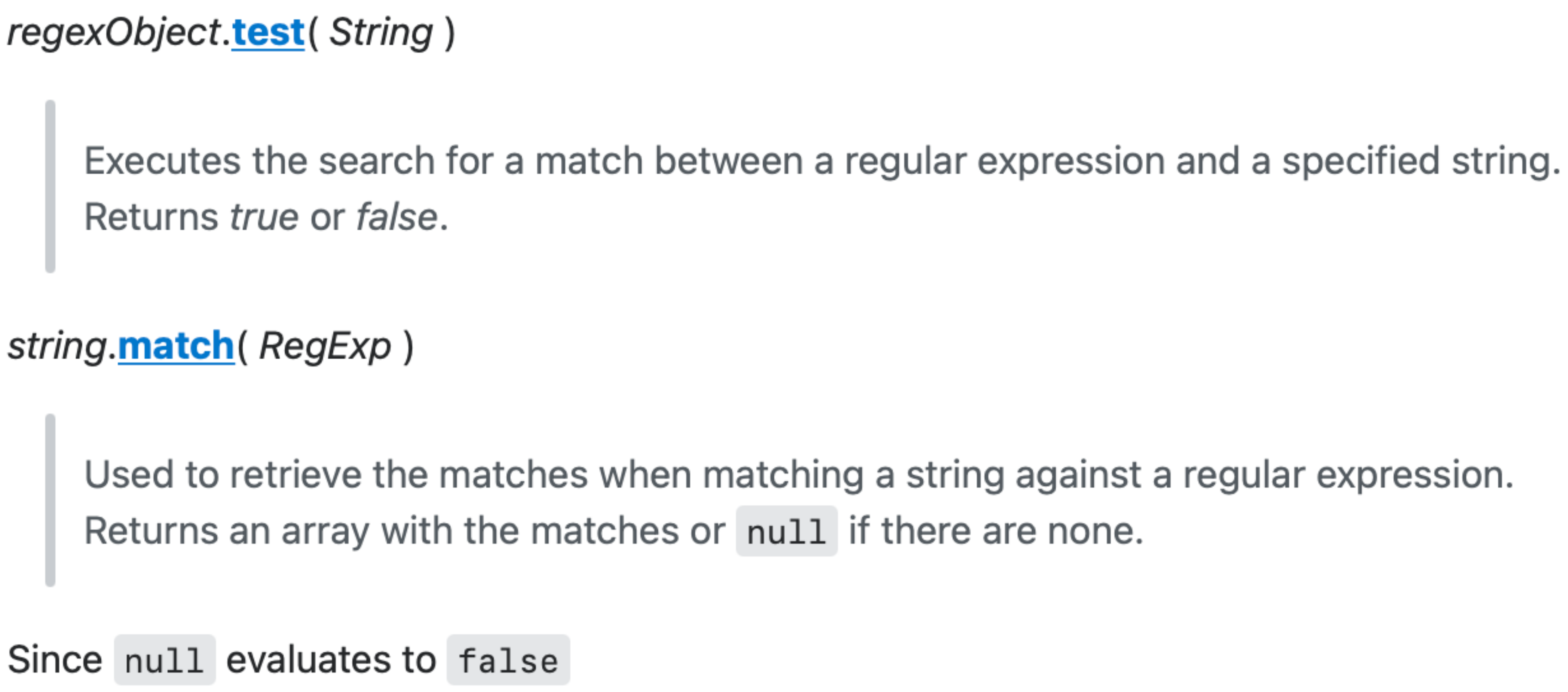
```
function phoneNumber(inputtxt){
  var phoneno = /^\(\d{3}\)-\d{3}-\d{4}$/;
  if((inputtxt.value.match(phoneno)){
        return true;
  } else {
        alert("The phone number is not valid");
        return false;
  }
}
```

## Try step by step



```
> "(123)-456-7890".match(/\d{3}/g)
<· ▸ (3) ['123', '456', '789']
> "(123)-456-7890".match(/\(\d{3}\)/g)
<· ▸ ['(123)']
> "(123)-456-7890".match(/\(\d{3}\)-/g)
<· ▸ ['(123)-']
> "(123)-456-7890".match(/\(\d{3}\)-\d{3}/g)
<· ▸ ['(123)-456']
> "(123)-456-7890".match(/\(\d{3}\)-\d{3}-\d{4}/g)
<· ▸ ['(123)-456-7890']
> "(123)-456-7890".match(/^\(\d{3}\)-\d{3}-\d{4}$/g)
<· ▸ ['(123)-456-7890']
```

# match vs. test



*regexObject*.**test**( *String* )

> Executes the search for a match between a regular expression and a specified string. Returns *true* or *false*.

*string*.**match**( *RegExp* )

> Used to retrieve the matches when matching a string against a regular expression. Returns an array with the matches or `null` if there are none.

Since `null` evaluates to `false`

## Examples (with test)

```
var dateTime = /\d{1,2}-\d{1,2}-\d{4} \d{1,2}:\d{2}/;
dateTime.test("30-5-2017 11:25")); // -> true
dateTime.test("30-5-2017 11:5")); // -> false
```

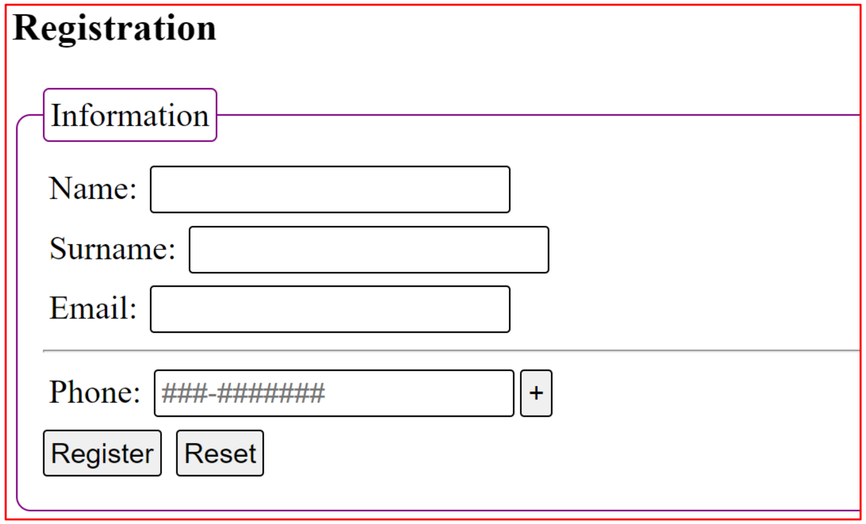
# More information on regular expressions

- JavaScript RegExp Reference

  https://www.w3schools.com/jsref/jsref_obj_regexp.Asp

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

# Dynamic form and validation (see DynamicFormOnCangeAndClick.zip)


Registration

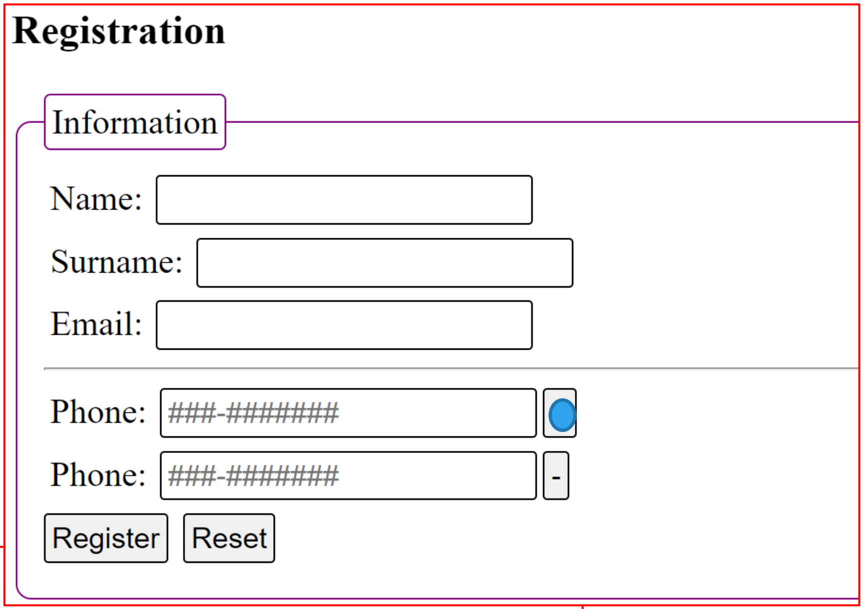
Information

Name:

Surname:

Email:

Phone: ###-#######


Registration

Information

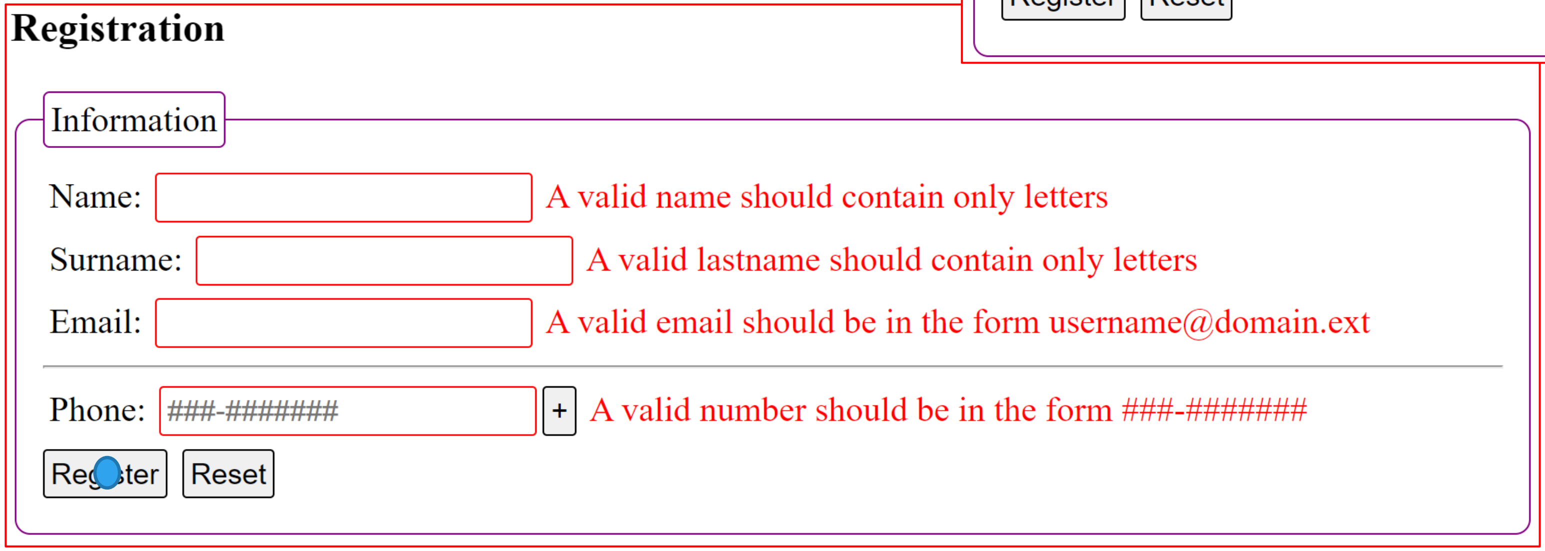
Name:

Surname:

Email:

Phone: ###-#######

Phone: ###-#######


Registration

Information

Name: A valid name should contain only letters

Surname: A valid lastname should contain only letters

Email: A valid email should be in the form username@domain.ext

Phone: ###-####### A valid number should be in the form ###-#######

## The form

```
<h3>Registration</h3>
<form id="regForm" action="Registration">
    <fieldset>
        <legend>Information</legend>
        <div>
            <label for="firstname">Name:</label><input type="text"
                name="firstname" id="firstname"
                onchange="validateFormElem(this, nameOrLastnamePattern, document.getElementById('errorName'), nameErrorMessage)"><span
                id="errorName"></span>
        </div>
        <div>
            <label for="lastname">Surname:</label><input type="text"
                name="lastname" id="lastname"
                onchange="validateFormElem(this, nameOrLastnamePattern, document.getElementById('errorLastname'), lastnameErrorMessage)"><span
                id="errorLastname"></span>
        </div>
        <div>
            <label for="email">Email:</label><input type="text" name="email"
                id="email"
                onchange="validateFormElem(this, emailPattern, document.getElementById('errorEmail'), emailErrorMessage)"><span
                id="errorEmail"></span>
        </div>
        <hr>
        <div id="phones">
            <div id="phoneRow0">
                <label for="phone0">Phone:</label><input type="text" name="phone"
                    id="phone0" placeholder="###-#######"
                    onchange="validateFormElem(this, phonePattern, document.getElementById('errorPhone0'), phoneErrorMessage)"><input
                    type="button" value="+" onclick="addPhone()"><span
                    id="errorPhone0"></span>
            </div>
        </div>
        <div>
            <input type="submit" value="Register" onclick="return validate()">
            <input type="reset" value="Reset">
        </div>
    </fieldset>
</form>
```

# CSS

```css
body {
    width: 700px;
    margin: 0 auto;
}

legend {
    padding: 3px;
    border: 1px solid purple;
    border-radius: 3px;
}

fieldset {
    border: 1px solid purple;
    border-radius: 7px;
}

input {
    border: 1px solid black;
    border-radius: 2px;
}
```

```css
label, input, span {
    padding: 3px;
}

label, input {
    margin-right: 3px;
}

div {
    margin: 6px 0 6px 0;
}

.error {
    border: thin solid red;
}
```

## Regular expressions

```
const nameOrLastnamePattern = /^[A-z]+$/g;
const emailPattern = /^\S+@\S+\.\S+$/g;
const phonePattern = /^([0-9]{3}-[0-9]{7})$/g;
const nameErrorMessage = "A valid name should contain only letters";
const lastnameErrorMessage = "A valid lastname should contain only letters";
const emailErrorMessage = "A valid email should be in the form username@domain.ext";
const phoneErrorMessage = "A valid number should be in the form ###-#######";
function validateFormElem(formElem, pattern, span, message) {
    if(formElem.value.match(pattern)){
        formElem.classList.remove("error");
        span.style.color = "black";
        span.innerHTML = "";
        return true;
    }
    formElem.classList.add("error");
    span.innerHTML = message;
    span.style.color = "red";
    return false;
}
```

# Validation

```javascript
function validate() {
    let valid = true;
    let form = document.getElementById("regForm");

    let spanName = document.getElementById("errorName");
    if(!validateFormElem(form.firstname, nameOrLastnamePattern, spanName, nameErrorMessage)){
        valid = false;
    }
    let spanLastname = document.getElementById("errorLastname");
    if (!validateFormElem(form.lastname, nameOrLastnamePattern, spanLastname, lastnameErrorMessage)){
        valid = false;
    }
    let spanEmail = document.getElementById("errorEmail");
    if (!validateFormElem(form.email, emailPattern, spanEmail, emailErrorMessage)){
        valid = false;
    }

    for (let i = 0; i < count; i++){
        let spanPhone = document.getElementById("errorPhone" + i);
        if (spanPhone == null){ // It has been removed
            continue;
        } else {
            if (!validateFormElem(document.getElementById("phone" + i), phonePattern, spanPhone, phoneErrorMessage)){
                valid = false;
            }
        }
    }
    return valid;
}
```

## Add/remove phone input fields

```
let count = 1;

function addPhone() {
    let container = document.getElementById("phones");
    let div = document.createElement("div");
    div.id = "phoneRow" + count;

    let label = document.createElement("label");
    label.htmlFor = "phone" + count;
    label.appendChild(document.createTextNode("Phone:"));
    div.appendChild(label);

    let element = document.createElement("input");
    element.type = "text";
    element.name = "phone";
    element.id = "phone" + count;
    element.placeholder = "###-#######";
    div.appendChild(element);

    let input = document.createElement("input");
    input.type = "button";
    input.value = "-";
    input.addEventListener("click", function() {removePhone(div)});
    div.appendChild(input);

    let span = document.createElement("span");
    span.id = "errorPhone" + count;
    div.appendChild(span);
    // To add the onchange handler, it is needed to create the span first
    element.addEventListener("change", function(){
        validateFormElem(element, phonePattern, span, phoneErrorMessage)});

    count++;
    container.appendChild(div);
}
```

```
function removePhone(element) {
    element.parentNode.removeChild(element);
}
```

# Dynamic form and validation (see DynamicFormOnChangeAndClick2.zip)

- Same as the example before but with the use of the constraint validation API

```
<h3>Registration</h3>
<form id="regForm" action="Registration">
    <fieldset>
        <legend>Information</legend>
        <div>
            <label for="firstname">Name:</label><input type="text"
                name="firstname" id="firstname" required pattern="^[A-Za-z]+$"
                onchange="validateFormElem(this, document.getElementById('errorName'), nameOrLastnameErrorMessage)"><span
                id="errorName"></span>
        </div>
        <div>
            <label for="lastname">Surname:</label><input type="text"
                name="lastname" id="lastname" required pattern="^[A-Za-z]+$"
                onchange="validateFormElem(this, document.getElementById('errorLastname'), nameOrLastnameErrorMessage)"><span
                id="errorLastname"></span>
        </div>
        <div>
            <label for="email">Email:</label><input type="email" name="email"
                required
                onchange="validateFormElem(this, document.getElementById('errorEmail'), emailErrorMessage)"
                id="email"><span id="errorEmail"></span>
        </div>
        <hr>
        <div id="phones">
            <div id="phoneRow0">
                <label for="phone0">Phone:</label><input type="tel" name="phone"
                    id="phone0" placeholder="###-#######" required
                    pattern="^([0-9]{3}-[0-9]{7})$"
                    onchange="validateFormElem(this, document.getElementById('errorPhone0'), phoneErrorMessage)"><input
                    type="button" value="+" onclick="addPhone()"><span
                    id="errorPhone0"></span>
            </div>
        </div>
        <div>
            <input type="submit" value="Register" onclick="return validate()">
            <input type="reset" value="Reset">
        </div>
    </fieldset>
</form>
```

# Validation

```
const nameOrLastnameErrorMessage = "This field should contain only letters";
const emailErrorMessage = "The email field should be in the form username@domain.ext";
const phoneErrorMessage = "The number field should be in the form ###-#######";
const emptyFieldErrorMessage = "This field cannot be empty"

function validateFormElem(formElem, span, errorMessage) {
    if(formElem.checkValidity()){
        formElem.classList.remove("error");
        span.style.color = "black";
        span.innerHTML = "";
        return true;
    }
    formElem.classList.add("error");
    span.style.color = "red";
    if (formElem.validity.valueMissing){
        span.innerHTML = emptyFieldErrorMessage;
    } else {
        span.innerHTML = errorMessage;
    }
    return false;
}
```

## Other resources

- Form with Multiple Steps
  - https://www.w3schools.com/howto/howto_js_form_steps.asp
- Autocomplete
  - https://www.w3schools.com/howto/howto_js_autocomplete.asp
- Modal Login Form
  - https://www.w3schools.com/howto/howto_css_login_form.asp
- Checkout Form
  - https://www.w3schools.com/howto/howto_css_checkout_form.asp
- Form with Icons
  - https://www.w3schools.com/howto/howto_css_form_icon.asp
- Password Validation
  - https://www.w3schools.com/howto/howto_js_password_validation.asp
- …