



# Киберсигурност и устойчив бизнес

Упражнение #02

# Съдържание



## Видове валидация на форми Разновидности






# 01

# Валидация на форми

разновидности



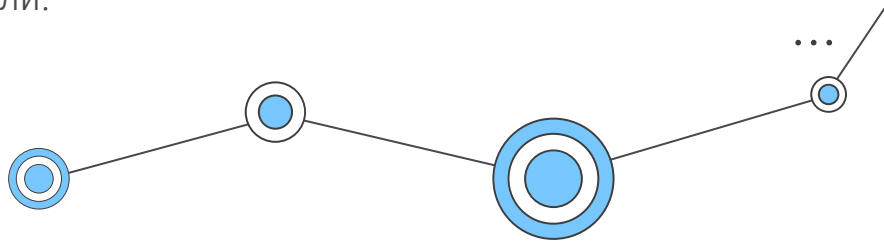


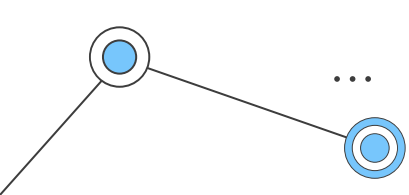
# Валидиране на уеб форми



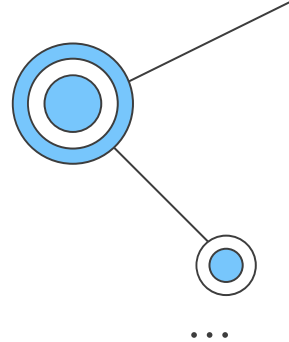
Уеб формулярите са много мощен инструмент за взаимодействие с потребителите. Най-често те се използват за събиране на данни от потребители или им позволяват да контролират потребителския интерфейс. Въпреки това, поради исторически и технически причини, невинаги е очевидно как да ги използваме в пълния им потенциал. Тук ще покрием всички основни аспекти на уеб формулярите. От маркирането на тяхната HTML структура, стилизиране на контролите вътре, валидиране на данните от формуляра и подаване на данни към сървъра.

Овластяването на формулярите изисква повече от обикновени HTML познания. Затова трябва да разгледаме някои специфични техники за стилизиране на контролите вътре, както и да притежаваме известни познания за скриптове, за да можем допълнително да валидираме и създаваме персонализирани контроли.





# Видове валидация на форми



01

Client-side validation

Видове

02

Server-side validation

Видове

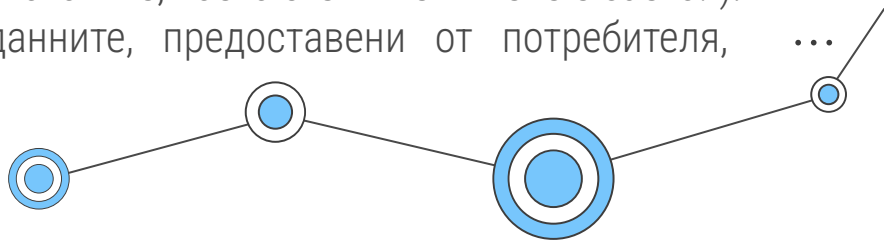


# Шестте компонента на уеб формуляра




За да създадем един уеб формуляр, ние използваме следните HTML елементи: `<form>`, `<label>`, `<input>`, `<textarea>` и `<button>`.

Всяка форма трябва да съдържа следните 6 компонента:

1. **Етикети:** за да обясните на потребителите за какво се отнасят контролите.
  2. **Входящи полета:** за събиране на данни и получаване на обратна връзка. *Те включват:* текстови полета, полета за пароли, радио бутони, бутони за отметки, слайдери и други.
  3. **Действия:** това са събитията, които след натискане на линковете или бутоните, обработват по някакъв начин формата.
  4. **Помощ:** Допълнителни описания какво да правим с конкретна форма.
  5. **Съобщения:** те дават обратна връзка на потребителя, въз основа на входа, който е подал. Съобщенията могат да бъдат позитивни, (когато формата е попълнена правилно) или негативни (напр. "Потребителското име, което сте въвели вече е заето!").
  6. **Валидация:** Тези мерки гарантират, че данните, предоставени от потребителя, отговарят на приемливите параметри.
- 

# Шестте компонента на веб формуляра

5. Message →  Please review information that you have entered.

1. Label → First name\*      Last name\*

2. Input Field →      

Your email address\*      Repeat email\*

6. Validation → Email address not valid  
Note: no-one can see your email address.

☒ By email

saved *choricat*      Can't read the text in the box?  
Refresh Listen Help ← 4. Help

Type the text above here\*

Yes, I have read and I accept the [Skype Terms of Use](#) and the [Skype Privacy Statement](#)

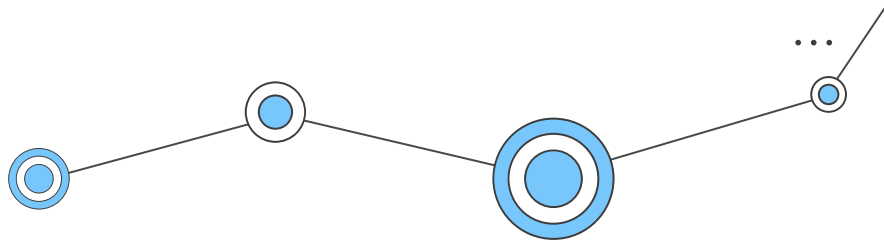
I agree - Continue → 3. Action



# Трите аспекта на формулярите



Въпреки разликите в оформлението, функционалността и целта, според Каролайн Джарет и Джери Гафни в книгата си „Формуляри, които работят: Проектиране на уеб формуляри за използваемост“, **всички формуляри имат три основни аспекта:**

1. **Взаимовръзка (Relationship):** Формулярите установяват връзка между потребителя и организацията.
  2. **Разговор (Conversation):** Те установяват диалог между потребителя и организацията.
  3. **Външен изглед (Appearance):** По начина, по който изглеждат, установяват връзката и интеракцията с потребителя.
- 

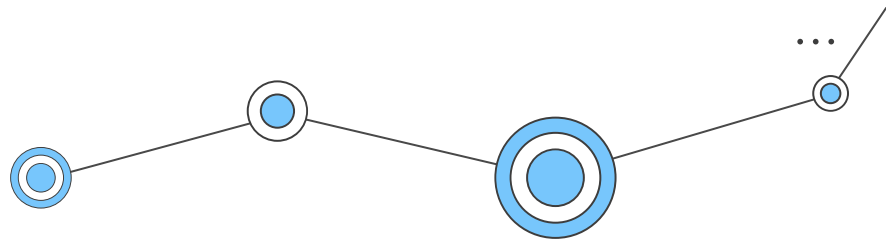




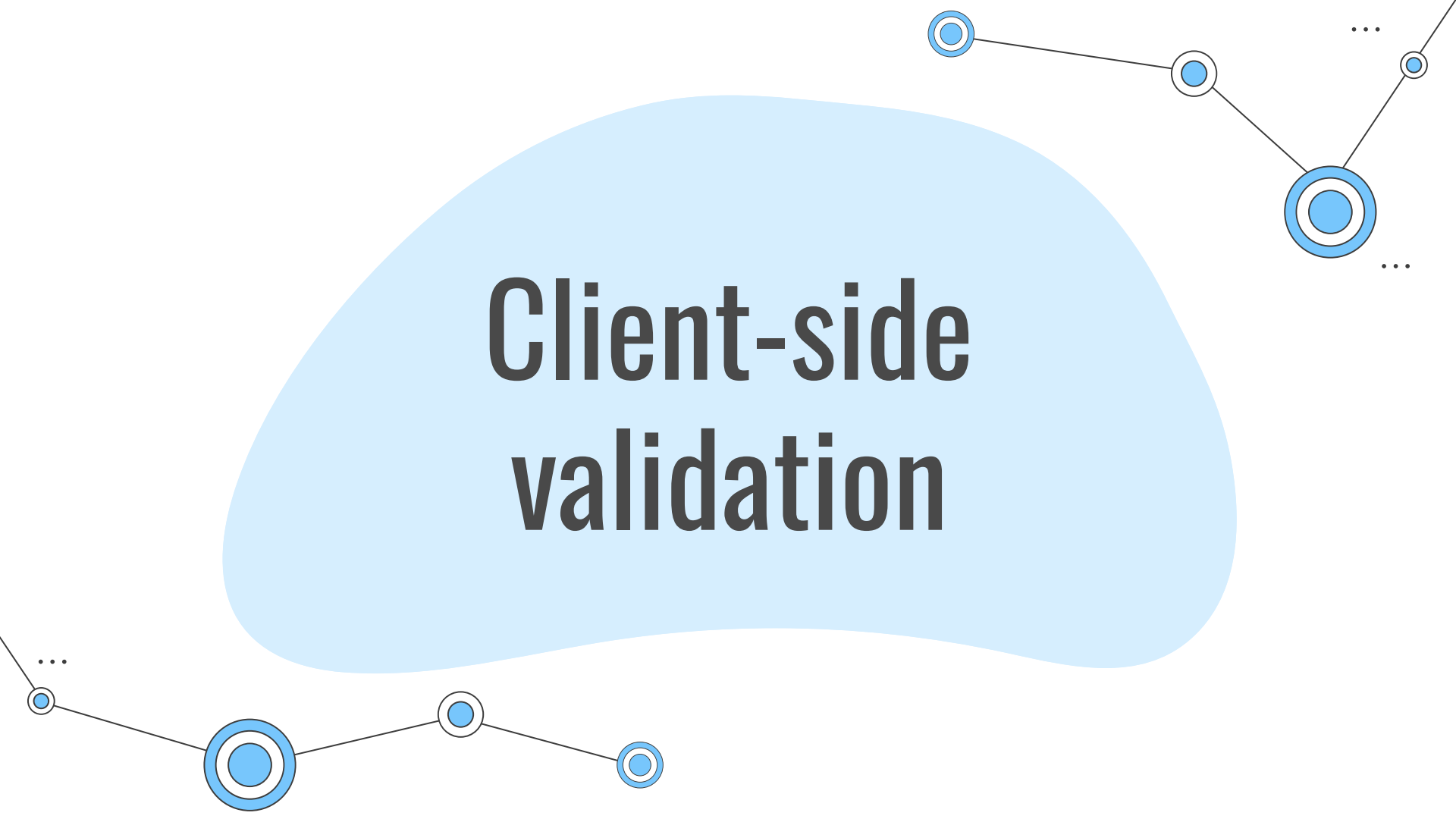
# Трите аспекта на формулярите

|                                      | Top          | Right             | Left          |
|--------------------------------------|--------------|-------------------|---------------|
| Speed of Completion                  | Fastest      | →                 | Slowest       |
| Horizontal Space Needed              | Least        | →                 | Most          |
| Vertical Space Needed                | Most         | →                 | Least         |
| Space Available For Label Text       | Most         | →                 | Least         |
| Proximity To Input                   | Closest      | →                 | Least Close   |
| User Eye Movement                    | Down         | Down & Right      | Down & Right  |
| Time To Move From Label To Input (*) | 50ms         | 240ms             | 500ms         |
| Ideal For                            | Simple Forms | Less Simple Forms | Complex Forms |

<https://www.uxmatters.com/mt/archives/2006/07/label-placement-in-forms.php>



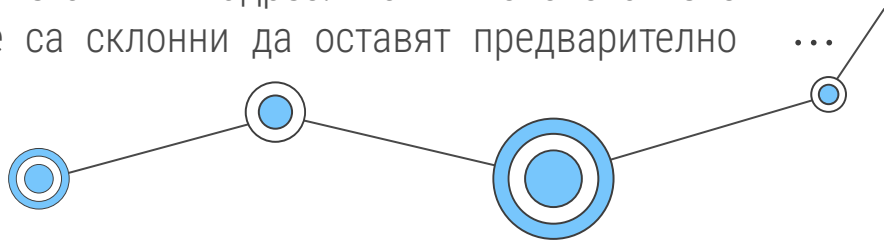
# Client-side validation



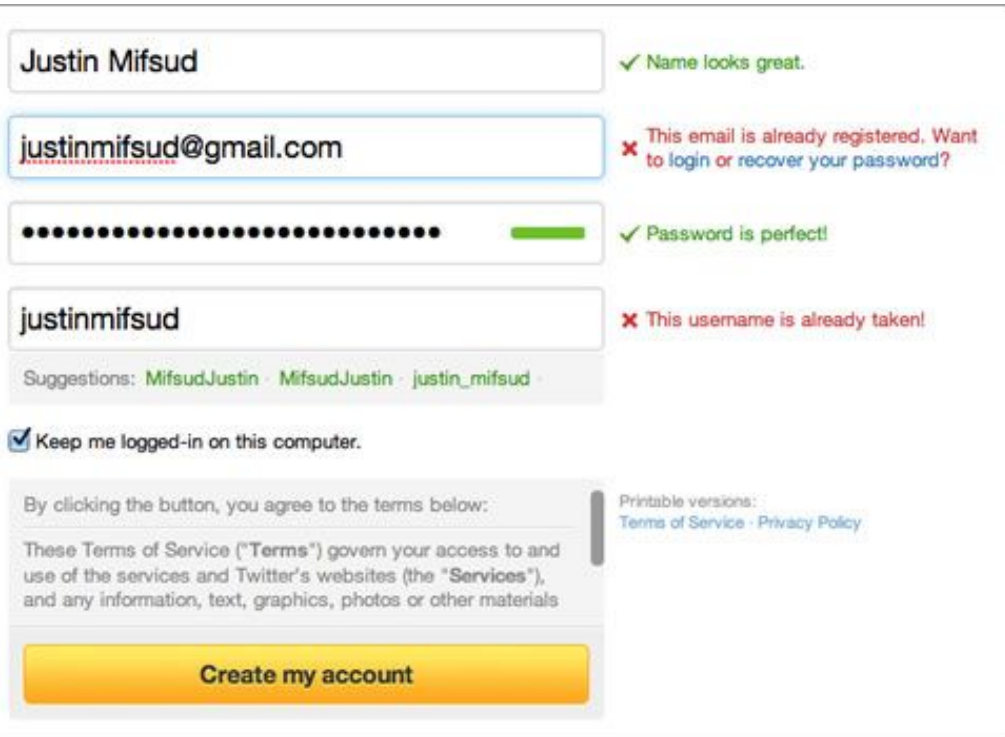


# Валидация



- **Навсякъде, където е необходимо:** Прекомерното валидиране е също толкова лошо, колкото и пълното му отсъствие, защото ще “затормозява” потребителите. Ограничете валидирането до потвърждаване на ключови точки (*като наличието на потребителско име*), осигуряване на реалистични отговори (*като например недопускане на възраст над 130*) и предлагане на отговори, при които диапазонът от възможни данни е краен, но не твърде дълъг, за да се включат в падащо меню меню (*като префикс за код на държавата*).
  - **Интелигентни настройки по подразбиране (smart defaults):** Използвайте интелигентните настройки по подразбиране, за да направите попълването на формуляра от потребителя по-бързо и по-точно. *Например*, предварително изберете държавата на потребителя въз основа на неговия IP адрес. Но ги използвайте с повишено внимание, защото потребителите са склонни да оставят предварително избраните полета такива, каквито са.
- 

# Валидация



Justin Mifsud ✓ Name looks great.

justinmifsud@gmail.com ✗ This email is already registered. Want to login or recover your password?

..... ✓ Password is perfect!

justinmifsud ✗ This username is already taken!

Suggestions: MifsudJustin · MifsudJustin · justin\_mifsud

☒ Keep me logged-in on this computer.

By clicking the button, you agree to the terms below:

These Terms of Service ("Terms") govern your access to and use of the services and Twitter's websites (the "Services"), and any information, text, graphics, photos or other materials

Printable versions: [Terms of Service](#) · [Privacy Policy](#)

Create my account

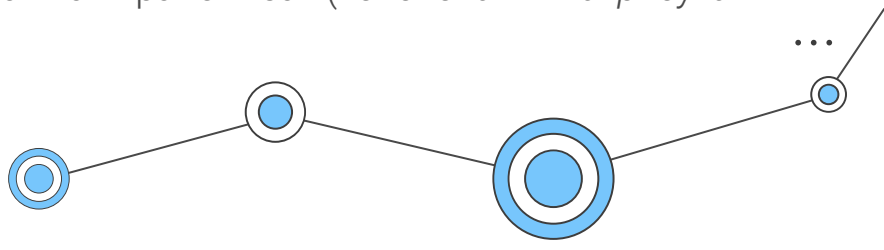
Формулярът за регистрация на Twitter използва, както **динамично валидиране**, така и **интелигентни настройки по подразбиране** (за име, имейл адрес, парола и потребителско име).



# Основни native форм контроли



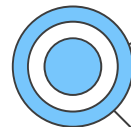
Всички основни текстови контроли споделят някои общи поведения:

- Те могат да бъдат маркирани като: **readonly** (*потребителят не може да променя входа, но вече наличния ще се изпрати с останалите данни от формуляра*) или **disabled** (*въведената стойност не може да бъде променена и никога не се изпраща с останалите данни от формуляра*).
  - Те могат да съдържат **placeholder**: това е текст, който се появява в полето за въвеждане на текст и трябва да се използва за кратко описание на целта на полето.
  - Те могат да бъдат ограничени по размер (*физическият размер на полето*) и **maxlength** (*максималния брой знаци, които могат да бъдат въведени в полето*).
  - Те могат да се възползват от проверката на правописа (*използвайки атрибута **spellcheck***), ако браузърът я поддържа.
- 

...



# Типове HTML5 input и валидацията им



...

- **E-mail address:** `<input type="email" id="email" name="email">`

*Когато се използва този тип, от потребителя се изисква да въведете валиден имейл адрес в полето. Всяко друго съдържание ще предизвика браузъра да покаже грешка при изпращане на формуляра.*

Your comment:

I am a comment

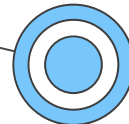
Your email:

k

Please enter an email address.

Send now

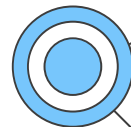
...



...



# Типове HTML5 input и валидацията им

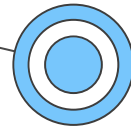


- **E-mail address:** `<input type="email" id="email" name="email" multiple>`

Можете също да използвате атрибута **multiple** в комбинация с типа на въвеждане на имейл, за да позволите да бъдат въведени няколко имейл адреса (разделени със запетаи).

Типът имейл, заедно с останалите по-нови типове за вход, осигуряват вградена проверка на грешки от страна на клиента, извършвана от брауъра, преди данните да бъдат изпратени до сървъра. Това е полезно помощно средство за насочване на потребителите да попълват точно формулярите и спестява време. **Но това не трябва да се счита за изчерпателна мярка за сигурност! Вашите приложения трябва винаги да извършват проверки за сигурност на всички изпратени от формуляра данни, както от страна на сървъра, така и от страна на клиента, тъй като последното се заобикаля лесно от злонамерените потребители.**

...



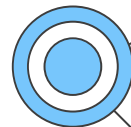
...





...

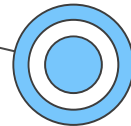
# Типове HTML5 input и валидацията им



...

Имайте предвид, че **a@b** е валиден имейл адрес според предоставените по подразбиране ограничения. За да приложите различно поведение на валидиране, можете да използвате атрибута **pattern** и така да персонализирате съобщенията за грешка!

**Забележка:** Ако въведените данни не са валидни, то псевдокласът `:invalid` ще се активира и свойството `validityState.typeMismatch` ще върне `true`.



...





## Типове HTML5 input и валидацията им

- **Тип search:** `<input type="search" id="search" name="search">`

Тук важна функционалност е, че стойностите на полето за търсене могат да бъдат автоматично записани и използвани повторно, за да се предложи автоматично довършване за множество страници от един и същи уебсайт.

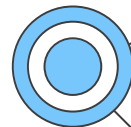
**Това обикновено се случва автоматизирано в повечето съвременни браузъри.**

|  |  |  |
|--|--|--|
| <input type="text" value="has a value"/> | <input type="text" value="has a value"/> | <input type="text" value="has a value"/> |
| <input type="text" value="has a value"/> | <input type="text" value="has a value"/> | <input type="text" value="has a value"/> |
| <input type="text" value="has a value"/> | <input type="text" value="has a value"/> | <input type="text" value="has a value"/> |
| <input type="text" value="has a value"/> | <input type="text" value="has a value"/> | <input type="text" value="has a value"/> |
| <input type="text" value="has a value"/> | <input type="text" value="has a value"/> | <input type="text" value="has a value"/> |



...

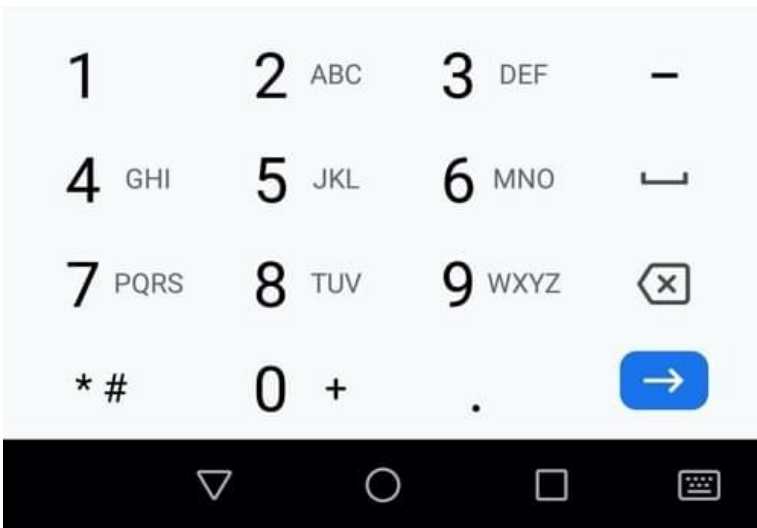
# Типове HTML5 input и валидацията им



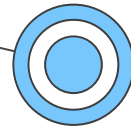
...

- **Тип Phone number:** `<input type="tel" id="tel" name="tel">`

Когато се осъществява достъп чрез сензорно устройство с динамична клавиатура, повечето устройства ще показват цифрова клавиатура, при наличие на `type="tel"`.



Поради разликите във формата, тук е позволено да се въвеждат и букви и нямаме ограничения! За да реализирате валидация на ниво клиент, можете да използвате отново атрибута **pattern**!



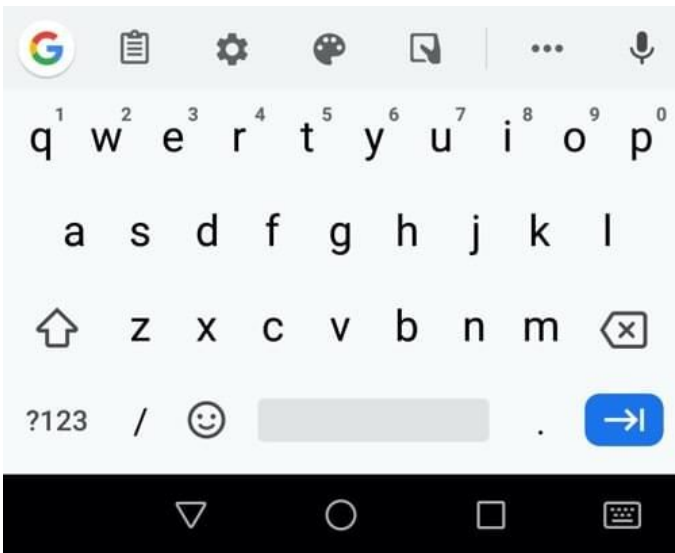
...



## Типове HTML5 input и валидацията им

- **Тип URL:** `<input type="url" id="url" name="url">`

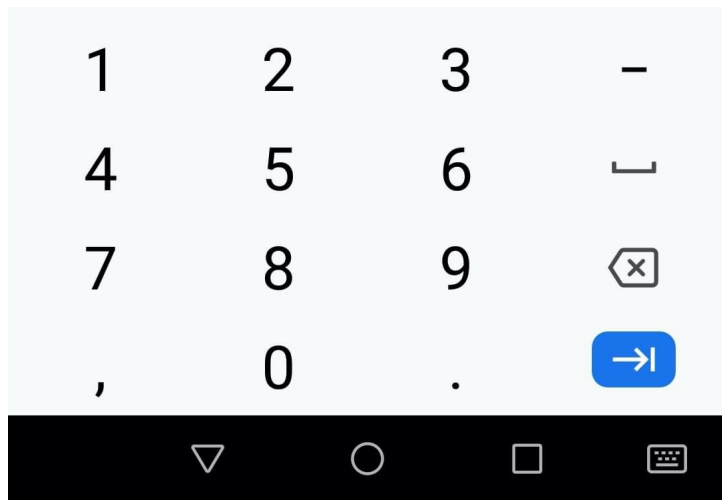
Този тип добавя специални ограничения за валидиране към полето. Браузърът ще съобщи за грешка, ако не е въведен протокол (като `http://` или `https://` ), или ако URL адресът е неправилно форматиран.



## Типове HTML5 input и валидацията им

- **Тип Number:** `<input type="number" name="age" id="age" min="1" max="10" step="2">`

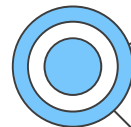
Контролите за въвеждане на числа могат да бъдат създадени с тип `number`. Този `<input>` изглежда като текстово поле, но позволява въвеждането само числа с плаваща запетая и обикновено предоставя бутони за увеличаване и намаляване на стойността на контролата. На устройства с динамични клавиатури обикновено се показва цифровата.



...



# Типове HTML5 input и валидацията им



- **Тип** **range** **(Slider control):**

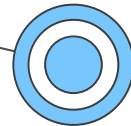
```
<label for="price">Choose a maximum house price: </label>  
<input type="range" name="price" id="price" min="50000" ...  
max="500000" step="100" value="250000">  
<output class="price-output" for="price"></output>
```

Друг начин да изберете число е да използвате плъзгач. Може да намерим такива най-често при сайтове от тип "закупуване на жилище", където искаме да зададем максимална цена на имота, по която да филтрираме, и т.н.

Choose a maximum house price:



Submit



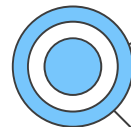
...



...



# Типове HTML5 input и валидацията им



- Тип **range** (Slider **control**):



Проблем с плъзгачите е, че те не предлагат визуална обратна връзка на текущата стойност. Ето защо е добре да се добавя елементът **<output>**, който да я връща. Можете да покажете входяща стойност или резултат на изчисление във всеки елемент, но **<output>** е специален. Тук **<label>** може да вземе атрибут **for**, който ви позволява да го асоциирате с елемент или елементите, от които зависи изходната стойност.

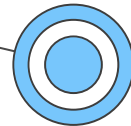
```
const price = document.querySelector('#price');  
const output = document.querySelector('.price-output');
```

```
output.textContent = price.value;
```

```
price.addEventListener('input', function() {  
  output.textContent = price.value;  
});
```

...

...



...



# Типове HTML5 input и валидацията им



- **Тип datetime-local, month, time, week (Date и time pickers):**

```
<input type="datetime-local" name="datetime" id="datetime">
```

```
<input type="month" name="month" id="month">
```

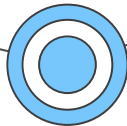
```
<input type="time" name="time" id="time">
```

```
<input type="week" name="week" id="week">
```

```
<label for="myDate">When are you available this  
summer?</label>
```

```
<input type="date" name="myDate" min="2013-06-01"  
max="2013-08-31" step="7" id="myDate">
```

...



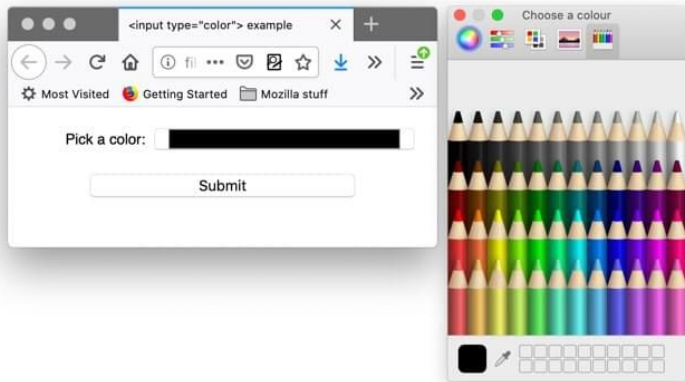
...

# Типове HTML5 input и валидацията им

- Тип **color** (Color picker):

```
<input type="color" name="color" id="color">
```

Цветовите винаги са малко трудни за реализация. Има много записи чрез които да ги представите: **RGB** стойности (десетични или шестнадесетични), **HSL** стойности, ключови думи и т.н. Когато се поддържа, кликуването върху контролата, тя показва функцията за избор на цвят по подразбиране на операционната система.

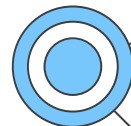




...



# Типове HTML5 input и валидацията им

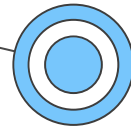


- **Тип** **color** (**Color** **picker**): ...  
Върнатата стойност винаги е шестнадесетичен цвят с малки букви.

Pick a color:

Submit

...





# UI pseudo-classes



Оригиналните псевдокласове, достъпни за нас (от CSS 2.1), подходящи за формуляри са:

**:hover:** Избира елемент, само, когато се задържи курсора на мишката върху него.

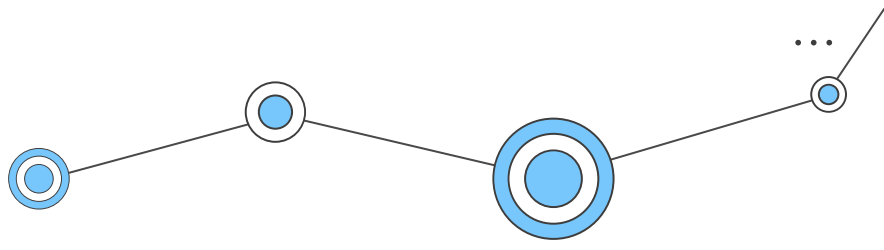
**:focus:** Избира елемент, само когато е фокусиран (т.е. чрез табулиране чрез клавиатурата).

**:active:** Избира елемент, само когато е активиран, (т.е. докато се щракне върху него или когато се натисне клавиша Return / Enter, в случай на активиране с клавиатура).

Псевдокласове, достъпни с CSS 3, са:

**:required** и **:optional:** Задължителни или незадължителни контроли от формуляра.

**:valid** и **:invalid** + **:in-range** и **:out-of-range:** Целеви контроли на формуляра, които са валидни/невалидни, според ограниченията за валидиране на формуляра, зададени върху тях, или върху обхвата: вътре/извън.



# UI pseudo-classes

Други

псевдокласове,

достъпни

с

CSS

3,

са:

...

**:enabled** и **:disabled** + **:read-only** и **:read-write**: Активни или неактивни контроли на формуляри, (напр. чрез набор от HTML атрибути) и контроли за формуляри за само за четене или за четене-писане.

**:checked**, **:indeterminate** и **:default**: Респективно, насочване към квадратчета за отметка и бутони за избор, които са отметнати, в неопределено състояние (нито отметнати или не) и избраната по подразбиране опция, когато страницата е с проверения набор от атрибути или елемент `<option>` с избрания набор от тях).




# HTML атрибут: pattern



Атрибутът **pattern** определя **regular expression**, за който стойността на контролата от формуляра трябва да съвпада. Ако стойност, различна от нула, не отговаря на ограниченията, зададени от стойността на шаблона, то свойството **patternMismatch** ще бъде **true**. Атрибутът **pattern** принадлежи на типовете текст, тел, имейл, url, парола и търсене.

**Употреба:** За да включите **pattern**, вие трябва да предоставите описание на шаблона във видим текст близо до контролата. Освен това е добре да добавите и атрибута **title**, който дава описание на шаблона. Браузърите могат да използват съдържанието на заглавието по време на валидиране на ограниченията, за да съобщят на потребителя, че определен модел не съвпада. Някои браузъри показват подсказка със заглавно съдържание, подобрявайки използваемостта за зрящи потребители. Освен това, при наличие на помощна assistive технология, заглавието може да бъде прочетено на глас, ако контролът е на фокус.



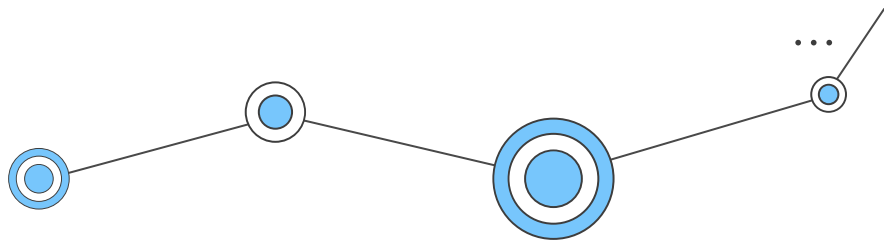


# HTML атрибут: pattern



## Пример:

```
<p>
  <label>Enter your phone number in the format (123) - 456 - 7890
    (<input  name="tel1"  type="tel"  pattern="[0-9]{3}"  placeholder="###"
aria-label="3-digit area code" size="2"/>) -
    <input  name="tel2"  type="tel"  pattern="[0-9]{3}"  placeholder="###"
aria-label="3-digit prefix" size="2"/> -
    <input  name="tel3"  type="tel"  pattern="[0-9]{4}"  placeholder="####"
aria-label="4-digit number" size="3"/>
  </label>
</p>
```



# HTML атрибут: pattern

## Пример:

```
<p>
  <label>Enter your phone number in the format (123) - 456 - 7890
    (<input  name="tel1"  type="tel"  pattern="[0-9]{3}"  placeholder="###"
  aria-label="3-digit area code" size="2"/>) -
    <input  name="tel2"  type="tel"  pattern="[0-9]{3}"  placeholder="###"
  aria-label="3-digit prefix" size="2"/> -
    <input  name="tel3"  type="tel"  pattern="[0-9]{4}"  placeholder="####"
  aria-label="4-digit number" size="3"/>
  </label>
</p>
```

Enter your phone number in the format (123) - 456 - 7890 (###) - ### - ####



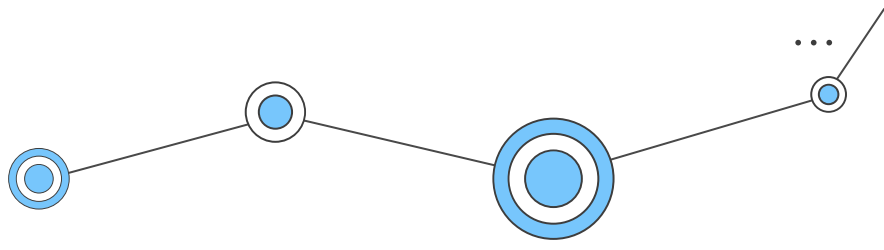
# HTML атрибут: pattern



## Определяне на pattern:

Можете да използвате атрибута **pattern**, за да укажете регулярен израз, чиято въведена стойност трябва да съвпадне с него, за да се счита за валидна. Примерът по-долу ограничава стойността до 4-8 знака и изисква тя да съдържа само малки букви. ...

```
<form>
  <div>
    <label for="uname">Choose a username: </label>
    <input type="text" id="uname" name="name" required size="45"
      pattern="[a-z]{4,8}" title="4 to 8 lowercase letters">
    <span class="validity"></span>
    <p>Usernames must be lowercase and 4-8 characters in length.</p>
  </div>
  <div>
    <button>Submit</button>
  </div>
</form>
```



# HTML атрибут: pattern

Результат:

Choose a username:



Names must be lowercase and 4-8 characters in length.

Submit



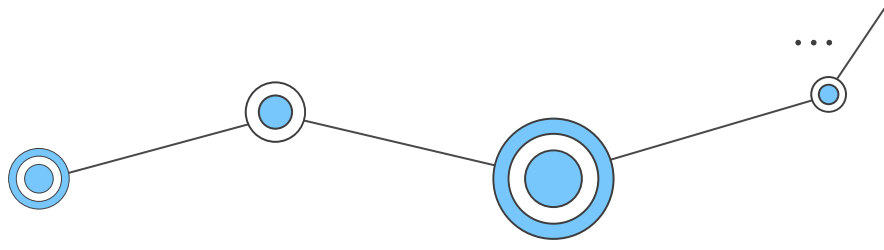


# HTML атрибут: pattern



## Важно!:

Когато контрола съдържа **pattern** и се използва атрибутът **title**, то той трябва да описва шаблона. Разчитането на атрибута **title** не се препоръчва обикновено за визуално показване на текстово съдържание, тъй като много потребителски агенти не го показват по достъпен начин. Някои браузъри показват **tooltip** при задържане на курсора на елемент със заглавие, но това работи само за потребители с клавиатура и или сензорен екран. Това е една от няколко причини, поради които трябва да включите информация за потребителите как да попълнят контрола, за да отговаря на изискванията.

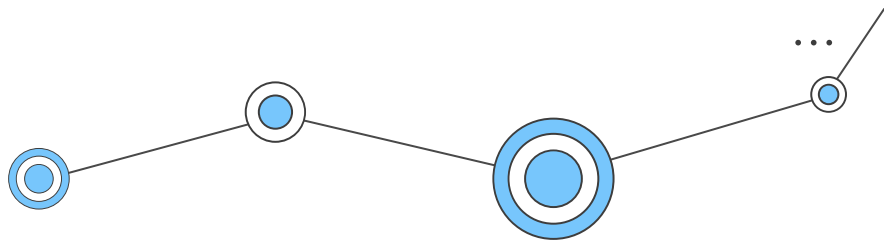




# Client-side валидация



При изпращане на данните от формуляр, ние трябва да се уверим, че те са правилно въведени от потребителя, и след това да ги обработим, без да "счупим" нашите приложения. Освен това е необходимо да помогнем на нашите потребители при попълването на формулярите. Валидирането на формите ни помага да постигнем и двете цели.



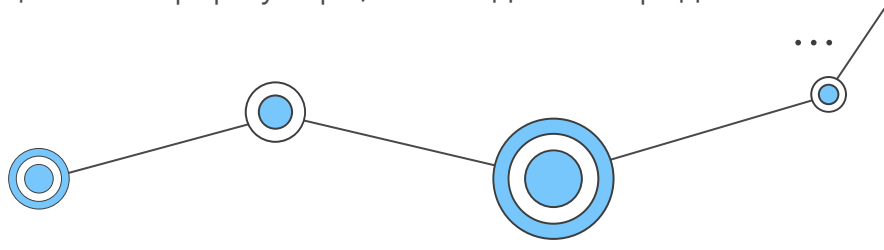


# Client-side валидация



Проверката на въведените данни, преди да бъдат подадени на сървъра, се нарича валидиране на формуляр от страна на клиента и помага да се гарантира, че предоставената информация съответства на изискванията, уточнени в различните контроли на формуляра.

*Съществуват три основни причини, поради които е необходима валидацията:*

1. **Искаме да получим правилни данни, в коректен формат.** Нашите приложения няма да работят правилно, ако данните на нашите потребители се съхраняват в грешен формат, неверни са или не са попълнени въобще.
  2. **Искаме да защитим данните на нашите потребители.** Принуждаването на нашите потребители да въвеждат защитени пароли, улеснява защитата на информацията за акаунта им.
  3. **Ние искаме да се защитим.** Налични са много начини, по които злонамерените потребители могат да използват незащитени формуляри, за да повредят приложението.
- 

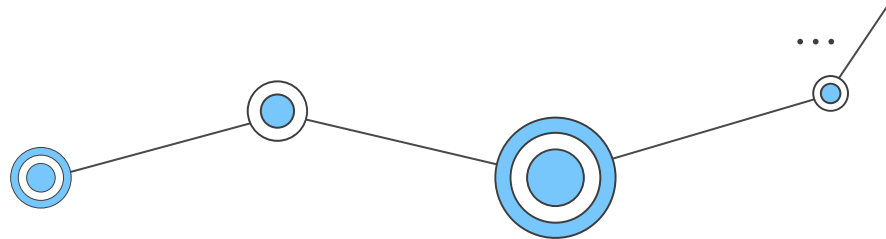


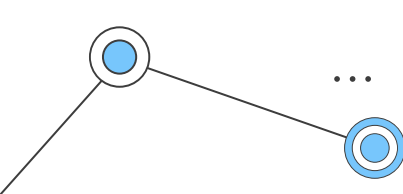
# Видове Client-side валидация



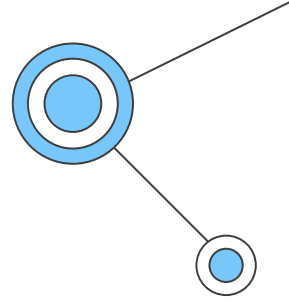
Съществуват два вида валидация от страна на клиента: ...

1. **Вградената проверка на формуляр:** тя използва функции за валидиране на HTML5 формуляри. Тази проверка обикновено не изисква JavaScript. Вградената проверка на формуляр има по-добра производителност от JavaScript-разновидността, но не е толкова гъвкава.
2. **JavaScript валидирането** е много добра и гъвкава проверка, но трябва да я създадете изцяло сами или да използвате подходяща библиотека.





# Употреба на вградена проверка на формуляр (built-in form validation)



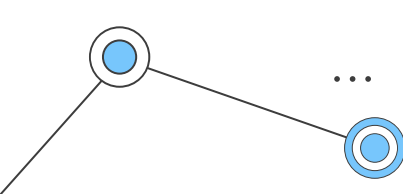
Една от най-важните характеристики, на контролите от HTML5 формулярите, е възможността им за вградена проверка на повечето потребителски данни, без да се разчита на JavaScript. Това се прави чрез използване на атрибутите за валидиране на елементи във формуляр: ...

**required:** *Посочва дали трябва да се попълни даденото поле, преди формулярът да се изпрати.*

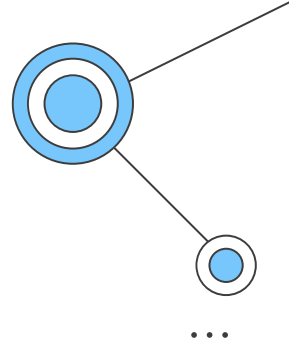
**minlength** и **maxlength:** *Определя минималната и максималната дължина на текстовите данни (низове).*

**min** и **max:** *Определя минималните и максималните стойности на цифровите данни.*





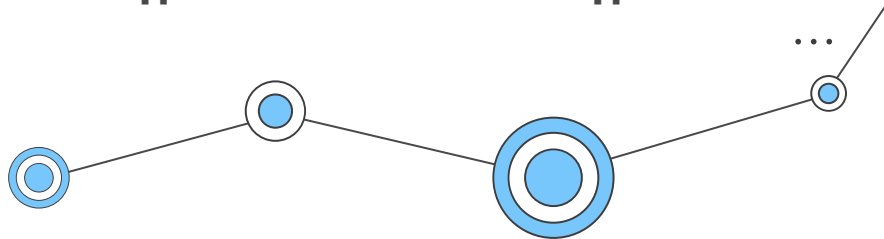
# Употреба на вградена проверка на формуляр (built-in form validation)

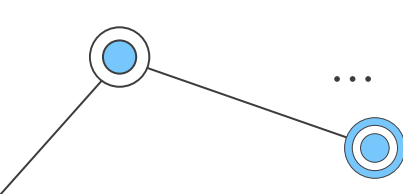


**type:** Указва дали данните трябва да са число, имейл адрес или друг предварително зададен, специфичен тип.

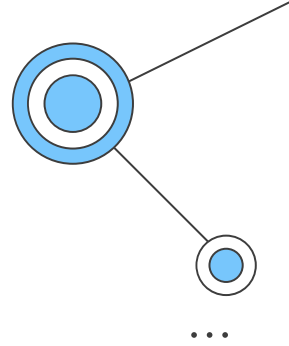
**pattern:** Определя регулярен израз, дефиниращ модел, който трябва да се следва от въведените данни.

**Ако данните, въведени в полето на формуляр, следват всички правила, посочени от горните атрибути, те се считат за валидни. Ако не: за невалидни.**

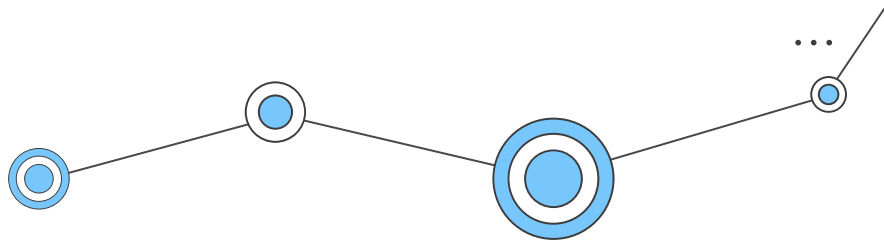


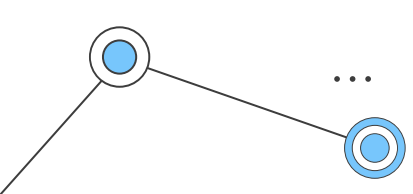


# Употреба на вградена проверка на формуляр (built-in form validation)

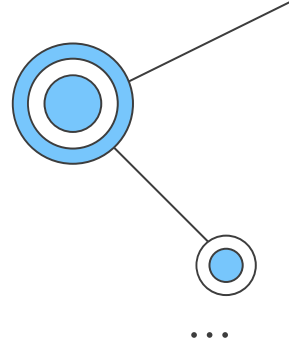


**Забележка:** Съществуват няколко грешки, които ще попречат на изпращането на формуляр. Това са: **badInput**, **patternMismatch**, **rangeOverflow** или **rangeUnderflow**, **stepMismatch**, **tooLong** или **tooShort**, **typeMismatch**, **valueMissing** или **customError**.

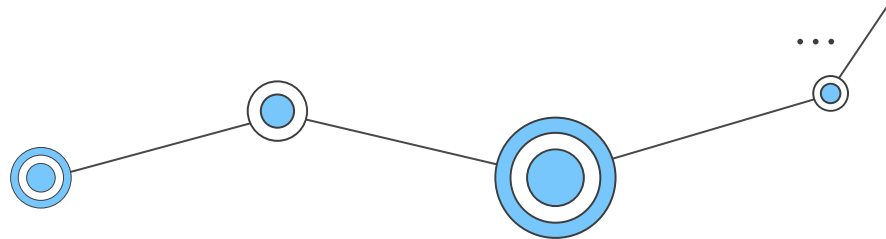




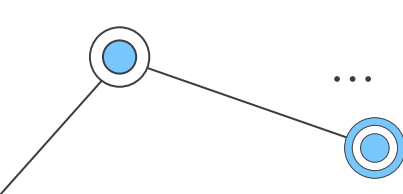
# Примери за вградена валидация от страна на клиента



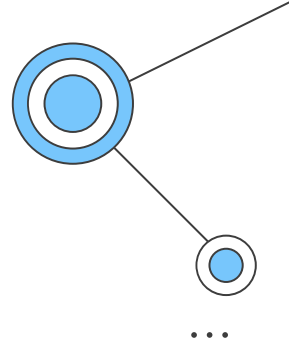
```
<form>  
    <label                                for="choose">  
        Would you prefer a banana or cherry?</label>  
    <input id="choose" name="i_like">  
    <button>Submit</button>  
</form>
```





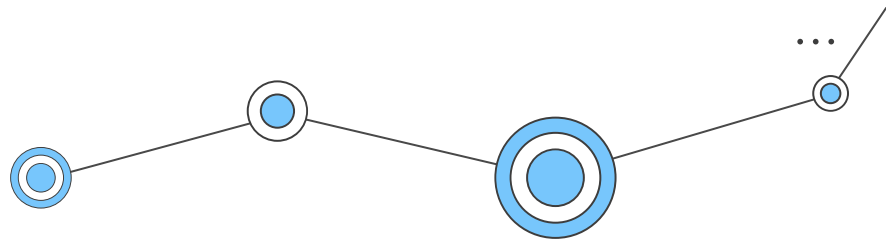


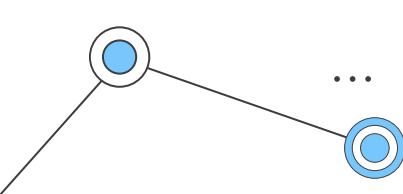
# Примери за вградена валидация от страна на клиента



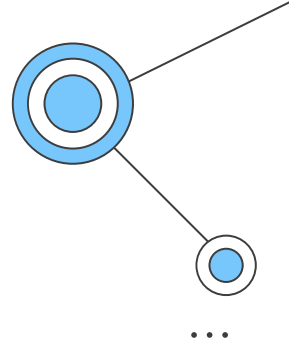
```
input:invalid {  
  border: 2px dashed red;  
}
```

```
input:valid {  
  border: 2px solid black;  
}
```





# Примери за вградена валидация от страна на клиента

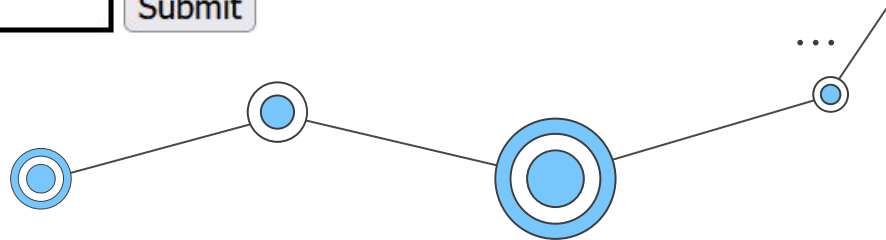


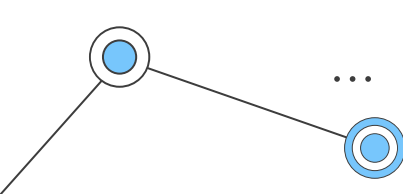
```
input:invalid {  
  border: 2px dashed red;  
}
```

```
input:valid {  
  border: 2px solid black;  
}
```

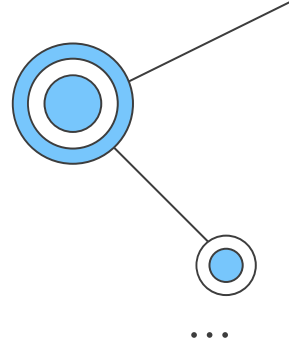
Would you prefer a banana or cherry?

Submit

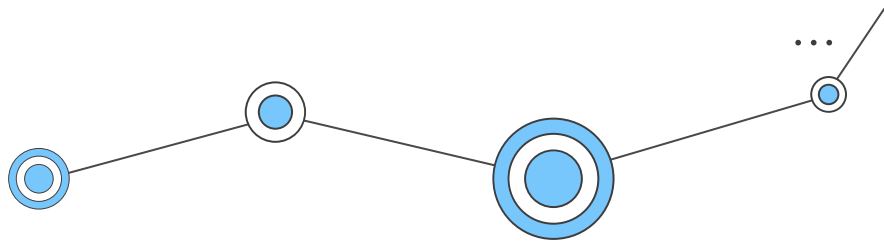


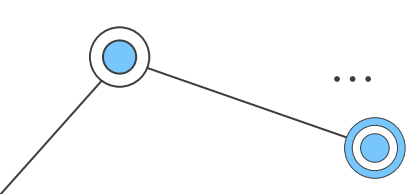


## Примери за вградена валидация от страна на клиента

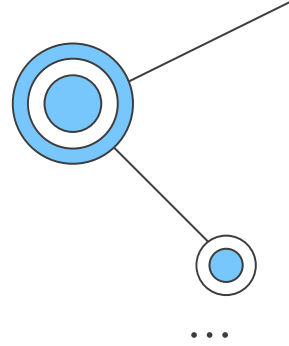


Най-простата функция за проверка на HTML5 е атрибутът **required**. Ако искате да направите въвеждането задължително, трябва да добавите този атрибут към елемента. Когато този атрибут е зададен за дадена контрола, то тя съответства на псевдокласа **:required UI** и формулярът няма да бъде изпратен, ако входът е празен, а ще се покаже съобщение за грешка. Освен това той ще се счита и за невалиден, съответстващ на псевдокласа **:invalid UI**.

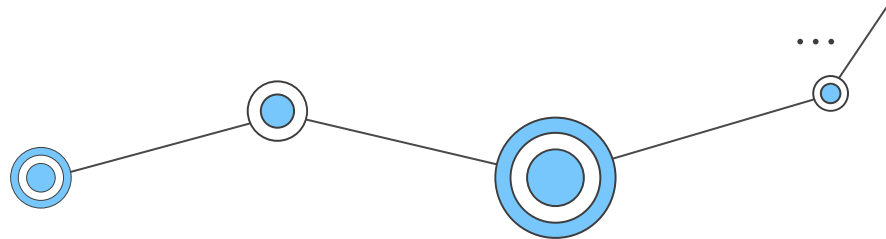


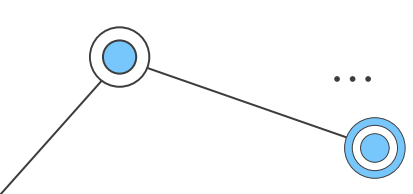


# Примери за вградена валидация от страна на клиента

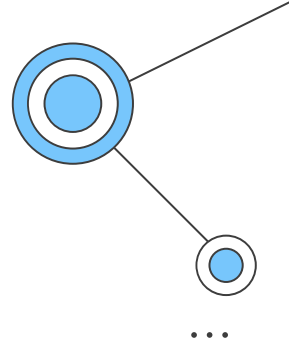


```
<form>  
  <label for="choose">Would you prefer a banana or  
cherry? (required)</label>  
  <input id="choose" name="i_like" required>  
  <button>Submit</button>  
</form>
```





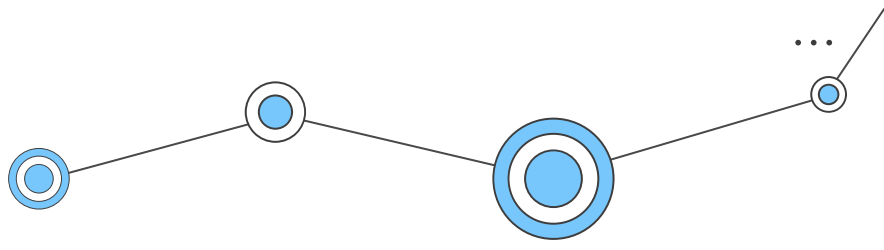
# Примери за вградена валидация от страна на клиента

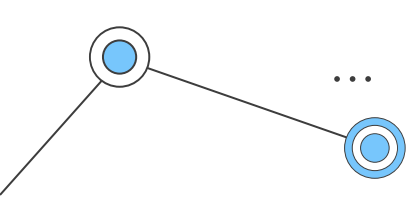


```
input:invalid {  
  border: 2px dashed red;  
}
```

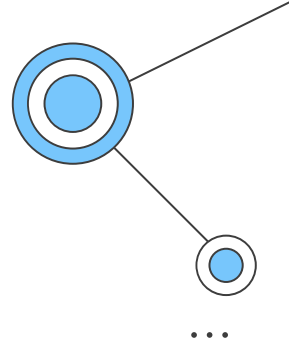
```
input:invalid:required {  
  background-image: linear-gradient(to right, pink,  
lightgreen);  
}
```

```
input:valid {  
  border: 2px solid black;  
}
```

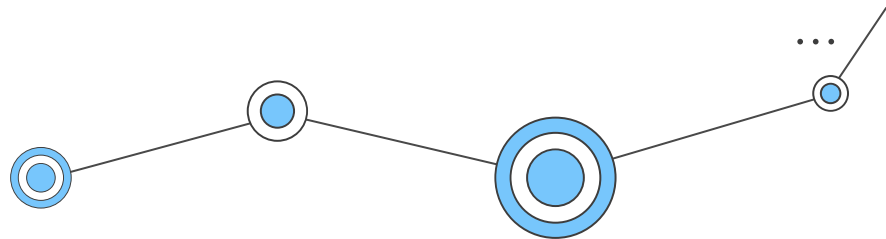


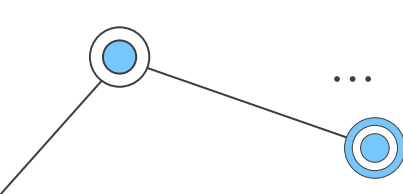


## Примери за вградена валидация от страна на клиента

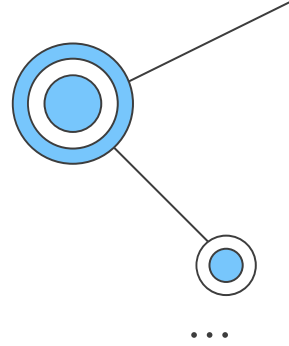


Ако се опитаме да изпратим формуляра без стойност, се появява съобщение за грешка по подразбиране („Моля, попълнете това поле“) и формата не може да бъде изпратена.





# Примери за вградена валидация от страна на клиента



**a** — Съвпада с един знак, който е точно a (не b, не aa и т.н.).

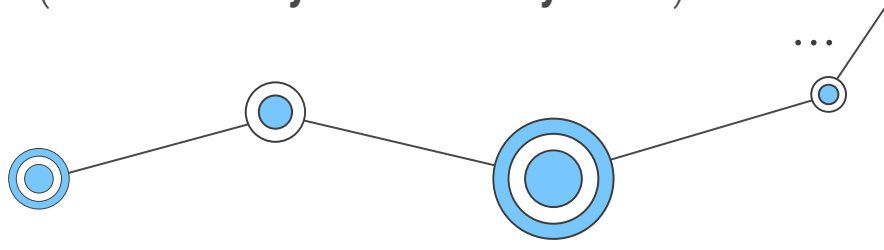
**abc** — Съвпада с **a**, последвано от **b**, последвано от **c**.

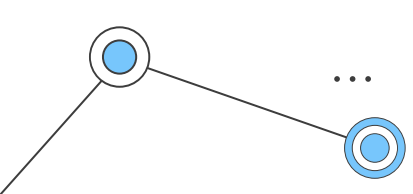
**ab?c** — Съвпада с **a**, по избор последвано от единично **b**, последвано от **c** (**ac** или **abc**).

**ab\*c** — Съвпада с **a**, по избор последвано от произволен брой **b**, последвано от **c** (**ac**, **abc**, **abbbbbs** и така нататък).

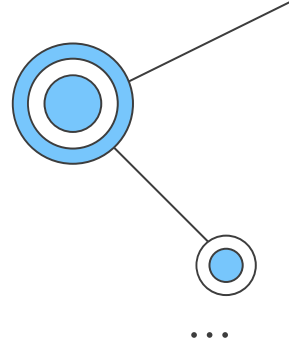
**a|b** — Съвпада с един знак, който е **a** или **b**.

**abc|xyz** — Съвпада точно с **abc** или точно с **xyz** (но не с **abcxyz** или **a** или **y** и т.н.).

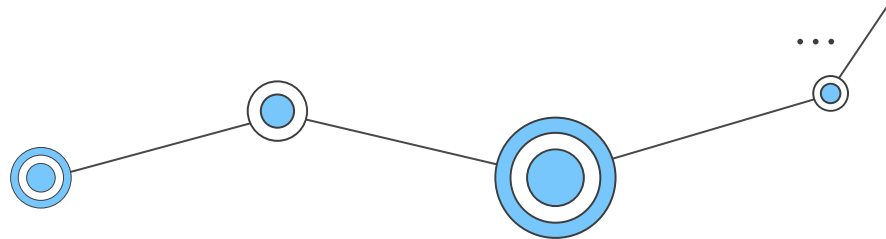




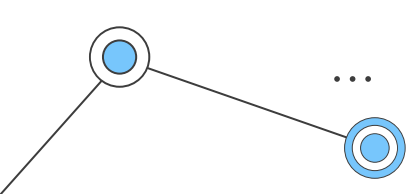
# Примери за вградена валидация от страна на клиента



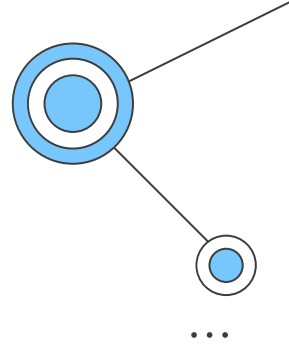
```
<form>
  <label for="choose">Would you prefer a banana or a
  cherry?</label>
    <input      id="choose"      name="i_like"      required
  pattern="[Bb]anana|[Cc]herry">
    <button>Submit</button>
</form>
```



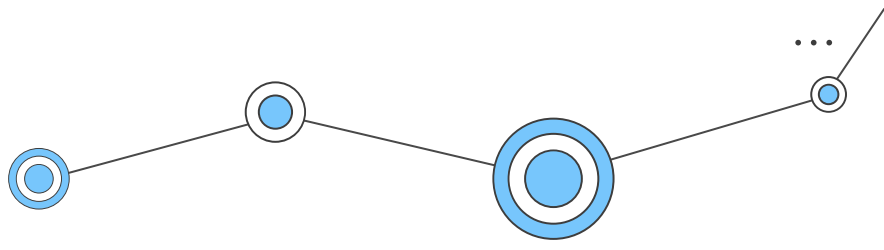




# Примери за вградена валидация от страна на клиента



```
<form>
  <label for="choose">Would you prefer a banana or a
cherry?</label>
  <input      id="choose"      name="i_like"      required
pattern="[Bb]anana|[Cc]herry">
  <button>Submit</button>
</form>
```





# Проверка на формуляри с помощта на JavaScript

Ако искате да имате по-голям контрол върху изгледа на грешките и съобщенията, които съпътстват формулярите, то решението е употребата на JavaScript.

## **Constraint Validation API**

Повечето браузъри поддържат API-то за валидиране на ограничения, който се състои от набор с методи и свойства, налични в следните DOM интерфейси на елементите от един формуляр:

**HTMLButtonElement** (представява елемент <button>)

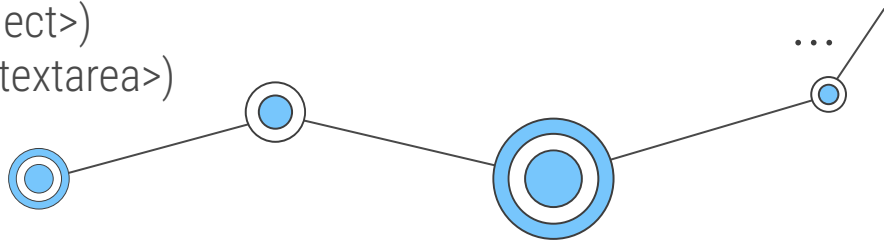
**HTMLFieldSetElement** (представява елемент <fieldset>)

**HTMLInputElement** (представява елемент <input>)

**HTMLOutputElement** (представява елемент <output>)

**HTMLSelectElement** (представява елемент <select>)

**HTMLTextAreaElement** (представява елемент <textarea>)





# Проверка на формуляри с помощта на JavaScript



**API-то предоставя следните свойства, достъпни за горните елементи:**

**validationMessage:** Връща локализирано съобщение, описващо ограниченията за валидиране, които контролата не удовлетворява, (ако има такива). Ако контролата не е кандидат за валидиране на ограничение (`willValidate` е `false`) или стойността на елемента удовлетворява неговите ограничения (`is valid`), това ще върне празен низ.

**validity:** Връща обект `ValidityState`, който съдържа няколко свойства, описващи състоянието на валидност на елемента:

<https://developer.mozilla.org/en-US/docs/Web/API/ValidityState>





# Проверка на формуляри с помощта на JavaScript




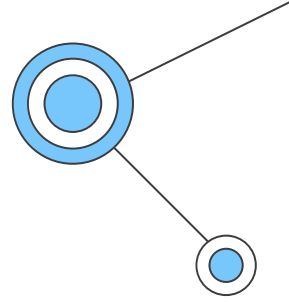
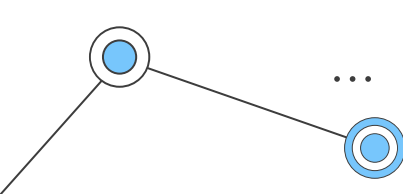
По-долу следват и наличните методи, които може да използваме за контролите в един формуляр и за самия него: ...

**checkValidity()** : Връща true, ако стойността на елемента няма проблеми с валидността; false > в противен случай. Ако елементът е невалиден, този метод също задейства невалидно събитие за елемента.

**reportValidity()** : Отчита невалидни полета, използващи събития. Полезен е в комбинацията с **preventDefault()** в манипулатор на събитие при **onSubmit**.

**setCustomValidity (message)** : Добавя персонализирано съобщение за грешка към елемента; ако зададете такова, то елементът се счита за невалиден и се показва посочената грешка. Това ни позволява да използваме JavaScript код, за да установим проблем при валидиране, различен от тези, предлагани от стандартните ограничения за проверка на HTML5. Съобщението се показва на потребителя, при възникване на проблем. ...



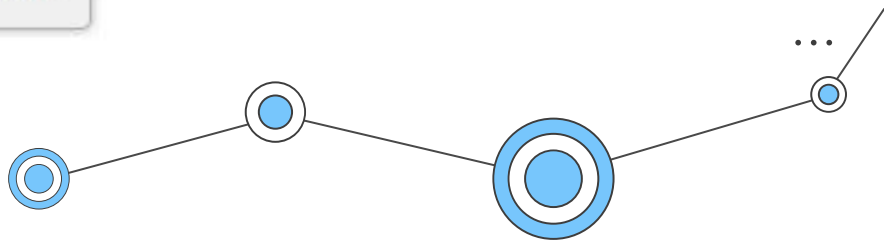


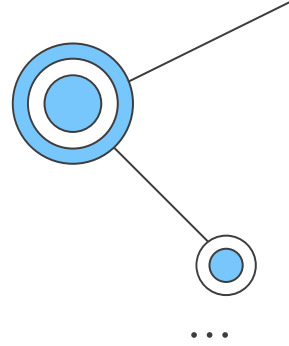
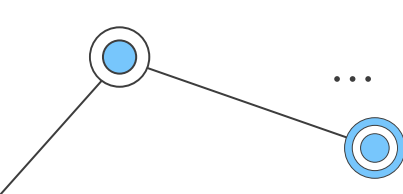
# Разработване на потребителски съобщения

При употребата на вградената HTML5 валидация, изкачащите съобщения често се показват на език, различен от този на формата. Освен това нямаме консистентна визия (по подразбиране) за различните браузъри и трябва да си “поиграем” доста, за да я уеднаквим. Това е първата причина за честата употреба на constraint validation API.

What's your favorite fruit?\*

Veuillez compléter ce champ.

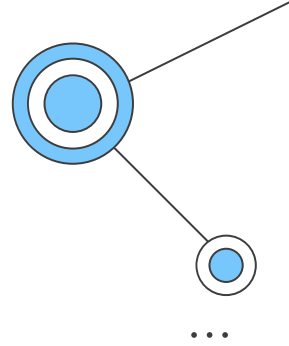
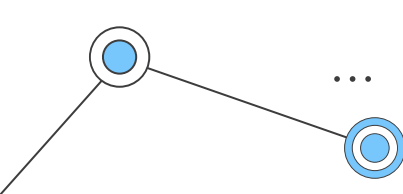




# Разработване на потребителски съобщения

## ПРИМЕРИ

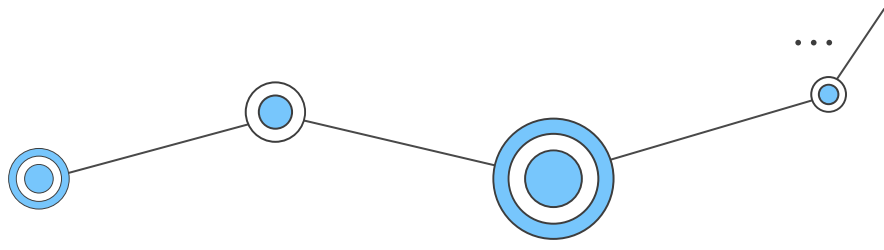


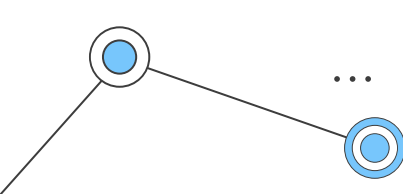


# Проверка на формуляри без употреба на вграденото API

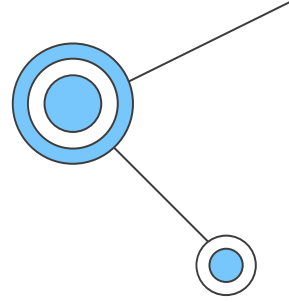
В някои случаи, като поддръжка на по-стари браузъри или персонализирани контроли, няма да можете или няма да искате да използвате стандартното API. Това не е проблем, тъй като отново е възможно да използваме JavaScript, за да потвърдите формуляра, но трябва да напишем всичко сами.

**При това положение, преди самото кодиране,  
трябва да си отговорим на следните въпроси:**





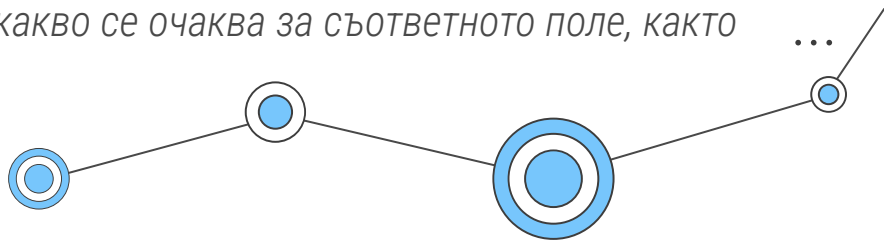
# Проверка на формуляри без употреба на вграденото API



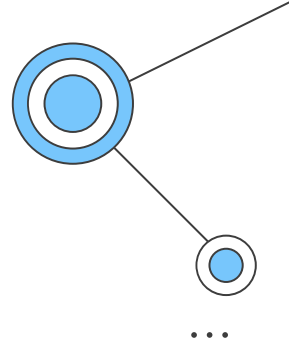
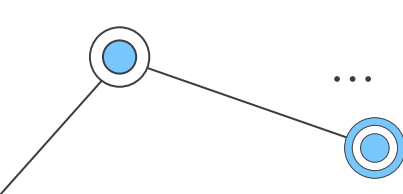
**1) Какъв вид проверки трябва да осъществим?** операции с низове, преобразуване на типове, регулярни изрази и т.н. ...

**2) Какво трябва да направя, ако формулярът не се изпрати след валидация (тоест проверката не е минала успешно)?** Това очевидно е въпрос на потребителски интерфейс. Трябва да решим как ще се държи формата. Формулярът изпраща ли данните все пак? Трябва ли да се подчертаят полетата, които са с грешка? Трябва ли да покажем съобщения за грешки? Как можем да помогнем на потребителя да коригира невалидни данни?

**3) Как можем да помогнем на потребителя да коригира невалидните данни?** За да намалим разочарованието на потребителя, е много важно да му предоставим възможно най-много полезна информация, която да го насочва как да коригира своите въведени данни. Трябва да предложим предварителни алтернативи, за да знае какво се очаква за съответното поле, както и ясни съобщения за грешки. ...







# Проверка на формуляри без употреба на вграденото API

## ПРИМЕРИ





# Проверка на формуляри, изградени с рамката Bootstrap 5

<https://getbootstrap.com/docs/5.1/forms/overview/>



# Литература

- <https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/pattern>
- [https://developer.mozilla.org/en-US/docs/Learn/Forms/UI\\_pseudo-classes](https://developer.mozilla.org/en-US/docs/Learn/Forms/UI_pseudo-classes)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_Expressions/Cheatsheet](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet)
- <https://www.smashingmagazine.com/2012/06/form-field-validation-errors-only-approach/>
- <https://www.smashingmagazine.com/2009/07/web-form-validation-best-practices-and-tutorials/>
- <https://www.webfx.com/blog/web-design/10-tips-for-optimizing-web-form-submission-usability/>
- <https://alistapart.com/article/inline-validation-in-web-forms/>



# Благодаря!

Въпроси?

mstoeva@uni-plovdiv.bg  
<http://edesign-bg.com>

