**Лабораторна 4**

**Регулярні вирази**

Регулярні вирази забезпечують швидкий і легкий спосіб зіставлення рядків в шаблон.

Наприклад, є регулярний вираз, який шукає текст "JavaScript" (без лапок):

JavaScript

Будь-який рядок, що містить текст "JavaScript", відповідає цьому регулярному виразу. Таким чином, регулярний вираз дозволяє нам виявити рядки, що містять конкретно цей текст.

**Базовий синтаксис**

1. Каретка (^) використовується для позначення початку рядка, який необхідно знайти, в той час як знак долара ($) використовується для позначення кінця:

JavaScript // Дорівнює "Isn't JavaScript great?"

^JavaScript // Дорівнює "JavaScript rules!",

// Не дорівнює "What is JavaScript?"

JavaScript$ // Дорівнює "I love JavaScript",

// Не дорівнює "JavaScript is great!"

^JavaScript$ // Дорівнює "JavaScript", and nothing else

1. Щоб видалити спеціальне значення символу, перед ним, ставиться зворотній слеш:

\$\$\$ // Дорівнює "Show me the $$$!"

1. Квадратні дужки можуть використовуватись для визначення набору символів, які можуть збігатися. Наприклад, цьому регулярному виразу будуть відповідати цифри від 1 до 5 включно.

[12345] // Дорівнює "1" і "3", але не "a" або "12"

1. Можуть бути також вказані діапазони чисел і букв.

[1-5]

[a-z]

[0-9a-zA-Z]

1. Поставивши ^ відразу ж після відкриття квадратної дужки, ви можете інвертувати набір символів, тобто пошуку будуть відповідати будь-які не перераховані символи:

[^a-zA-Z] // Все крім

1. Cимволи ?, + та \* також мають особливе значення. Зокрема, ? означає "попередній символ не є обов'язковим", + означає "один або більше з попереднього символу", а \* означає "нуль або більше попереднього символу".

bana?na // Дорівнює "banana" і "banna",

// не дорівнює "banaana".

bana+na // Дорівнює "banana" і "banaana",

// не дорівнює "banna".

bana\*na // Дорівнює "banna", "banana", і "banaaana",

// не дорівнює "bnana".

^[a-zA-Z]+$ // Дорівнює будь-якому string з однією або більше літерами

// тільки літери.

1. Дужки можуть використовуватися для угруповання символів разом, щоб застосувати ?, + або \* до них разом.

ba(na)+na // Дорівнює "banana" і "banananana",

// не дорівнює "bana" або "banaana".

1. Дужки також дозволяють визначити кілька рядків, які можуть відповідати запиту, використовуючи вертикальну риску (|), щоб відокремити їх.

^(ba|na)+$ // Дорівнює "banana", "nababa", "baba",

// "nana", "ba", "na", і тд.

1. Ось кілька спеціальних кодів, які можуть бути використані для зіставлення символів в регулярних виразах:

. // відповідає будь-якому символу крім розриву рядка

\n // відповідає символу переведення рядка

\r // відповідає символу повернення каретки

\t // відповідає символу горизонтальної табуляції

\b // відповідає границі слова

\B // відповідає межі не-слова

\d // теж саме, що й [0-9]

\D // теж саме, що й [^0-9]

\s // відповідає одного символу порожнього простору, включаючи пробіл, табуляцію, прогон сторінки, новий рядок

\S // відповідає одиничному символу порожнього простору

\w // теж саме, що й [A-Za-z0-9\_]

\W // теж саме, що й [^A-Za-z0-9\_]

**Використання регулярних виразів в JavaScript**

Можна створити регулярний вираз в JavaScript наступним чином:

**var** myRE = /regexp/;

Де регулярний вираз є кодом регулярним виразом, як описано вище. Наприклад, в наступному прикладі створено перший приклад регулярного виразу, який представлено в попередньому розділі, той, що визначає рядок "JavaScript":

**var** myRE = /JavaScript/;

Аналогічно, створюється регулярний вираз останнього прикладу:

**var** myRE = /^(ba|na)+$/;

За замовчуванням регулярні вирази в JavaScript чутливі до регістру і пошук відбувається тільки до першого співпадіння в будь-якому заданому рядку. Додаючи g (для глобального), і i (для ігнорування регістра) модифікатори після другого /, ви можете зробити пошук по регулярному виразу для всіх збігів в рядку і ігнорувати регістр відповідно. Ось кілька прикладів їх використання. Для кожного варіанту вказано, яка частина рядку «test1 Test2 TEST3» буде відповідати запиту:

/Test[0-9]+/ тільки “Test2”

/Test[0-9]+/i тільки “test1”

/Test[0-9]+/gi “test1”, “Test2” і “TEST3”

Використовувати регулярні вирази легко. Кожна змінна JavaScript, що містить текстовий рядок, підтримує три методи (або функції, якщо ви не звикли до об'єктно-орієнтованої термінології) для роботи з регулярними виразами: match(), replace(), і search().

match ()

match() приймає регулярний вираз як параметр і повертає масив всіх співпадаючих символів, знайдених в даному рядку. Якщо збігів не виявлено, то match() повертає значення false. Повертаючись до нашого початкового прикладу, припустимо, що нам необхідна функція, яка може перевірити, що рядок, введений користувачем як його номер телефону має вигляд (XXX) XXX-XXXX. Наступний код буде робити перевірку:

**function** checkPhoneNumber(phoneNo) {

**var** phoneRE = /^\(\d\d\d\) \d\d\d-\d\d\d\d$/;

**if** (phoneNo.match(phoneRE)) {

**return** true;

} **else** {

alert( "Номер телефону введено невірно!" );

**return** false;

}

}

В якості першого порядку, ця функція визначає регулярний вираз. Регулярний вираз починається з ^, щоб показати, що будь-який match повинен початися на початку рядка. Далі \ (, що буде відповідати відкритій дужці. Поставивши зворотній слеш, ми видалили її особливе значення в синтаксисі регулярних виразів (позначення початку набору альтернативних для порівняння символів).

Як уже згадувалося раніше, \ d є спеціальний код, який відповідає будь-якій цифрі. Таким чином, \ d \ d \ d відповідає будь-яким трьом цифрам. Ми могли б написати [0-9] [0-9] [0-9] для досягнення того ж ефекту, але перший варіант коротший. Решта коду повинна бути досить очевидною. ) відповідає за закриту дужку, і позначає місце залишене для номеру телефону, отже \ d \ d \ d- \ d \ d \ d \ d відповідає трьом цифрам, за ними слідує тире, а потім ще чотири цифри. І нарешті, $ вказує на те, що будь-який збіг має закінчуватися в кінці рядка.

Між іншим, ми могли б скоротити цей регулярний вираз, переписавши код до наступного вигляду, використовуючи можливості, про які не згадувалося вище.

var phoneRE = /^\(\d{3}\) \d{3}-\d{4}$/;

Наша функція потім перевіряє, чи phoneNo.match(phoneRE) приймає значення істина або брехня. Іншими словами, він перевіряє, відповідає чи ні рядок, що міститься в phoneNo нашому регулярному виразу (таким чином, повертаючи масив, який в JavaScript прирівнюватиметься до true). Якщо буде виявлено збіг, функція повертає true для підтвердження того, що рядок дійсно номер телефону. Якщо ні, то на дисплей виводиться повідомлення з попередженням про проблему і функція повертає false.

Найбільш поширене використання функцій такого типу, перевірка введеної користувачем інформацію до відповідної форми, перш ніж дозволити йому продовжувати роботу. Називаючи нашу функцію onSubmit обробник подій для форми, ми можемо запобігти подання неправильно заповнених інформаційних форм. Ось простий приклад, який демонструє використання нашої функції checkPhoneNumber():

<form action="...">

<label>Enter phone number (e.g. (123) 456-7890):

<input type="text" name="phone">

</label>

<input type="submit">

</form>

<script>

**var** form = document.querySelector('form');

form.addEventListener('submit', **function**() {

**return** checkPhoneNumber(this.phone.value);

});

</script>

Користувач не зможе представити цю форму, якщо не буде коректно введений номер телефону. Будь-яка спроба зробити це, буде викликати повідомлення про помилку, від нашої функції checkPhoneNumber().

**Завдання**

1. Створити форму зворотнього зв’язку на сайті.
2. Зробити щонайменше 5 полів (ім’я, прізвище, email, телефон, коментар).
3. Зробити валідацію полів за допомогою регулярних виразів.
4. Меню має бути написане на HTML+CSS+Javascript, без використання додаткових фреймворків