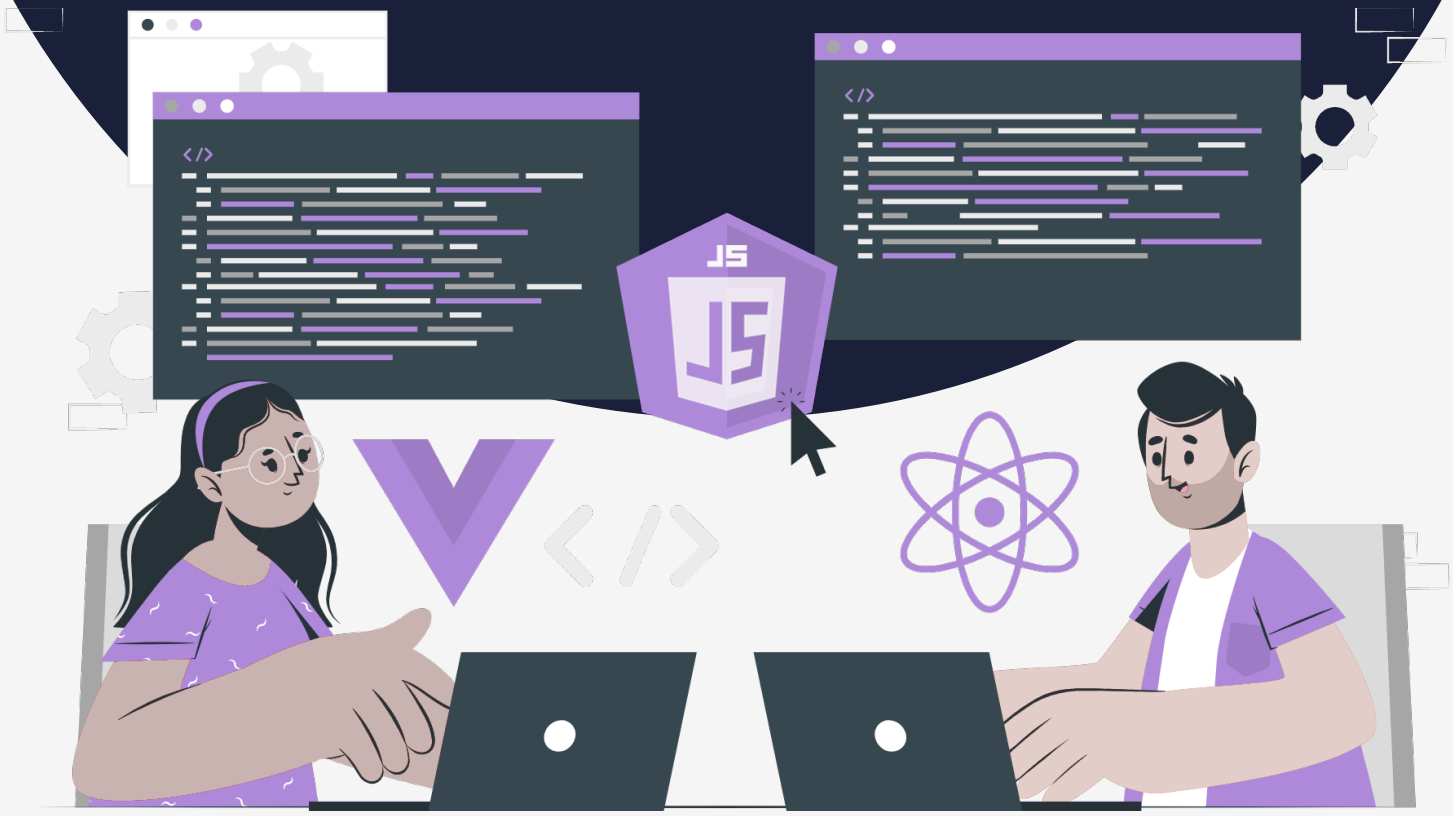
Lesson:

Writing regex with example



PW SKILLS

Topics Covered

1. Introduction.
2. Simple examples of regex.
3. Regex for email validation.

From the previous lectures, we have a clear idea of what regex is, how to use regex, and the applications of regex. In this lecture let's look at some examples where we will be writing regex.

Let's start with a simple example of finding a string pattern. To write regex we first need a pattern which we need to search for and one more important part of regex syntax is the "/pattern/". The pattern inside the slashes is the search pattern you want to match in a string.

Let's search for "PW Skills" in the string "PW Skills is the best educational platform to learn from".

```
let string = "PW Skills is the best educational platform to learn from.";

let pattern = /PW Skills/;
```

In order to test the regex pattern matching we have the ".test()" method. The test() is a method of the RegExp object that can be used to check if a string matches a given regular expression.

It returns a Boolean value indicating whether or not a match was found.

```
let string = "PW Skills is the best educational platform to learn from.";

let pattern = /PW Skills/;

console.log(pattern.test(string));

// OUTPUT: true
```

It is important to note that we are matching strings and the matching would be case sensitive.

We can use the "/i" flag which is the ignore case, to perform case-insensitive matching.

```
let string = "pw skills is the best educational platform to learn from.";
let pattern = /PW Skills/i;
console.log(pattern.test(string));
// OUTPUT: true


let string = "PW skills is the best educational platform to learn from.";
let pattern = /PW Skills/i;
console.log(pattern.test(string))
;// OUTPUT: true
```

```
let string = "PW Skills is the best educational platform to learn from.";

let pattern = /PW Skills/i;

console.log(pattern.test(string));

// OUTPUT: true
```

The string methods in javascript also help in regex pattern matching. The match() method of the String object can be used to find all matches of a regular expression in a string.

It returns an array containing the matched substrings, or null if no matches are found.

```
let string = "PW Skills is the best educational platform to learn from.";
let pattern = /PW Skills/i;
console.log(string.match(pattern));
/* OUTPUT:
[
    'PW Skills',
    index: 0,
    input: 'PW Skills is the best educational platform to learn from.',
    groups: undefined]
*/
```

Now we know the “test()” and “match()” method, we can use them based on the requirement.

Let’s try the “/g” flag to find multiple matches.

```
let string = "PW Skills is the best educational platform to learn from. PW
Skills has launched its Full Stack Web Development Course.";
let pattern = /PW Skills/g;
console.log(string.match(pattern));

/*
 OUTPUT:[ 'PW Skills', 'PW Skills' ]
*/
```

Now, let’s try to find numbers in the string.

```
let string = "The numbers are: 1 2 3 4 5 6";
let pattern = /[1-5]/g;
console.log(string.match(pattern));

/*
 OUTPUT:[ '1', '2', '3', '4', '5' ]
*/
```

The regular expression /[1-5]/g matches any digit between 1 and 5. The g flag ensures that all matches are returned, not just the first match. When the match() method is called on the string "The numbers are: 1 2 3 4 5 6" with the pattern as the argument, it returns an array containing all of the matches: ['1', '2', '3', '4', '5'].

Till now we have seen the basic implementation of regex. Now, let's look at some real-life challenges like using regex to validate email addresses.

Using regex to verify an email address. An email is a string separated into two parts by @ symbol. The first part contains personal information while the other contains the domain name at which the email is registered.

The personal information can contain:

- Uppercase and lowercase letters (A-Z and a-z)
- Numeric characters (0-9)
- Special characters - ! # $ % & ' * + - / = ? ^ _ ` { | } ~
- Period, dot, or full stop (.) with the condition that it cannot be the first or last letter of the email and cannot repeat one after another.

The domain name contains:

- Letters
- Digits
- Hyphens
- Dots

First, let's write the pattern to be searched for. We can break down the regex pattern as follows:

- ^: This is an anchor that matches the start of the string.
- [a-zA-Z0-9.!#$%&'*+/=?^_{|}~-]+`: This matches one or more characters from the specified character set. The character set includes letters (upper- and lower-case), digits, and special characters that are often used in email addresses.
- @: This matches the at symbol.
- [a-zA-Z0-9-]+: This matches one or more characters from the specified character set, which includes letters (upper- and lower-case) and digits.
- (?:\.[a-zA-Z0-9-]+)*: This matches zero or more instances of a sequence of a dot followed by one or more characters from the specified character set.
- $: This is an anchor that matches the end of the string.

So, now let's implement this.

```
let string = "mithun@pw.live";
let pattern = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;

console.log(pattern.test(string));
// OUTPUT: true

let string = "someone@gmail.com";
let pattern = /^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$/;

console.log(pattern.test(string));
// OUTPUT: true
```