OBJECTIVES

- Regular Expression (regex)

LECTURE

Regular Expression

Validate Text Input:

https://www.w3schools.com/js/js_regexp.asp

https://www.w3schools.com/jsref/jsref_regexp_test.asp

https://www.youtube.com/watch?v=rhzKDrUiJVk&t=835s

https://thenewstack.io/dont-fear-regex-getting-started-regular-expressions/

```
// Regular expression pattern to match alphabetic characters
const AlphabeticPattern = /^[A-Za-z]+$/;
```

```
NumberPattern.test(input);
```

- Square brackets (“[ ]”) essentially mean “**anything within these brackets**”
- **^** tells the search to **start** with the first character of the string.
- **$** means ‘search at the **end** of the string.’
- The dash (“-”) means range, as in between sequential numbers or letters of the alphabet, inclusive of the beginning and ending values. “A-D” equates to “A, B, C, and/or D.”
- “+” looks for *one or more* occurrences of the search pattern, and will throw a false if there are NO occurrences.

- The test() method is a regex expression method.

- It searches a string for a pattern, and returns true or false, depending on the result.

```
function isValidEmail(sText) {
    var reEmail = /^(?:\w+\.?)*\w+@(?:\w+\.?)*\w+$/;
    return reEmail.test(sText);
}
```

- ^: This symbol indicates the start of the string. It means the pattern should match from the beginning of the string.

- (?:\w+\.?)*: This part defines a non-capturing group (?: ...)*. Inside the group, \w+ matches one or more word characters (letters, digits, or underscores), and \.? matches an optional dot. The * outside the group quantifies the group, meaning it can occur zero or more times.

- \w+: This matches one or more word characters. This part ensures there is at least one-word character before the '@' symbol.

- @: This matches the '@' symbol, which separates the local part of the email address from the domain part.

- (?:\w+\.?)*: Similar to the first group, this matches the domain part of the email address. It allows for multiple subdomains separated by dots.

- \w+: This matches one or more word characters. It ensures there is at least one-word character after the '@' symbol and the domain.

- $: This symbol indicates the end of the string. It means the pattern should match until the end of the string.

To summarize, the regular expression checks if the string follows the pattern of having one or more word characters before the '@' symbol, followed by the '@' symbol, and then followed by one or more word characters after the '@' symbol, separated by optional dot characters. This pattern ensures basic validation for email addresses but may not cover all possible variations and edge cases.

```
function isValidDate(sText) {
    var reDate = /(?:0[1-9]|[12][0-9]|3[01])\/(?:0[1-9]|1[0-2])\/(?:19|20\d{2})/;
    return reDate.test(sText);
}
```

This part matches the day portion of a date. Let's break it down further:

- (?: ...) defines a non-capturing group.
- 0[1-9] matches a day from 01 to 09.
- [12][0-9] matches a day from 10 to 29.
- 3[01] matches a day from 30 to 31.
- The pipe symbol | is used as an OR operator to combine the different day patterns.
- \/: This matches the forward slash '/', which separates the day, month, and year in the date format.
- (?:0[1-9]|1[0-2]): This part matches the month portion of a date. It's similar to the day part but matches months from 01 to 12.
- \/: Another forward slash '/' to separate the month and year.
- (?:19|20\d{2}): This part matches the year portion of a date. Let's break it down:
- (?: ...) defines a non-capturing group.
- 19 matches years starting with 19.
- 20\d{2} matches years starting with 20, followed by exactly 2 digits (0-9).
- This part ensures that the year is either in the 20th or 21st century.

To summarize, the regular expression checks for dates in the format DD/MM/YYYY where DD is a day from 01 to 31, MM is a month from 01 to 12, and YYYY is a year in the range of 1900 to 2099. This pattern ensures basic validation for dates in this specific format.

```
function isValidVisa(sText) {
    var reVisa = /^(4\d{12}(?:\d{3})?)$/;
    if (reVisa.test(sText)) {
        return luhnCheckSum(RegExp.$1);
    } else {
        return false;
    }
}
```

This JavaScript function isValidVisa(sText) is used to validate Visa credit card numbers. Let's break down the function and its components:

- function isValidVisa(sText) { ... }: This declares a function named isValidVisa that takes one parameter sText, which presumably represents the credit card number to be validated.

- var reVisa = /^(4\d{12}(?:\d{3})?)$/;: This line defines a regular expression stored in the variable reVisa. Let's break down the regular expression:

- ^: This symbol indicates the start of the string.
- 4: This matches the starting digit of a Visa credit card number, which must be 4.
- \d{12}: This matches exactly 12 digits after the initial digit 4.
- (?: ...)?: This is a non-capturing group followed by ?, which makes the group optional.
- \d{3}: This matches an optional additional group of 3 digits (making the total length of the number either 13 or 16 digits).
- $: This symbol indicates the end of the string.
- So, the regular expression ensures that the string consists of either 13 or 16 digits, starting with a 4.
- if (reVisa.test(sText)) { ... }: This condition checks if the input string sText matches the regular expression reVisa. If the string matches the pattern (i.e., if it represents a valid Visa credit card number), the code inside the if block will execute.

- return luhnCheckSum(RegExp.$1);: If the input string matches the Visa pattern, the function luhnCheckSum() is called with the captured substring RegExp.$1 as an argument. RegExp.$1 holds the value of the first capturing group in the regular expression ((4\d{12}(?:\d{3})?)). The Luhn algorithm is commonly used to validate credit card numbers.

- return false;: If the input string does not match the Visa pattern, the function returns false, indicating that the credit card number is invalid.

Overall, this function checks whether the provided string sText represents a valid Visa credit card number by applying a regular expression pattern and then performing a Luhn checksum validation if the pattern matches. If the number is valid, the function returns the result of the Luhn checksum validation; otherwise, it returns false.