***regex.test(string)*** – Returns true if the string matches the regular expression “regex” passed.

***String.match(regex)*** *–* Returns an array of the string(s) that match a specific regular expression.

**regex** = /abc|def|ghi/ - Matches the patterns “abc”, or “def”, or “ghi”

**Flags**

**/I** – Ignores the case the string being matched. E.g. if *myString = “FreeCodeCamp is great”.*

*mystring.test(/freecodecamp/i);* returns true.

**/g** – Gets all the strings in a larger string that match some regular expression. E.g. if *mySting = “Twinkle twinkle little star”*,

*myString.match(/twinkle/ig)* returns [“Twinkle”,”twinkle”]

**Wildcards**

These match any number of characters.

**Period(.)**– Returns the string that has the characters specified before or after the period. E.g. “if myStringA = “It was a lot of fun”

myStringB = “It was full of sound”,

myStringA.match(/fu./) returns [“fun”]

myStringB.match(/fun./) returns [“full”]

myStringA = “It’s a bun”;

myStringB = “You should run”,

myStringA.match(/.un/) returns [“bun”]

myStringB.match(/.un/) returns [“run”]

To match specific characters:

**[]** – Matches any one of the characters within them. E.g. If “*myString = ”Beware of bugs*”,

*“/b[aie]g/.match(myString)”* returns [‘bugs’]

*“myString.match(/[a-f2-4]/ig)”* returns [“B”,”e”,”a”,”e”f”,”b”]

**Caret(^)** – When it is within a character set i.e. [], it matches everything that is not the specified pattern. E.g “myString = “Hello”,

myString.match(/[^0-9A-Z]/g) returns [“e”,”l”,”l”,”o”]

**Plus(+)** – Matches one or more characters same characters that are specified right before it. E.g myString = “Mississippi”,

*“myString.match(/s+/)*” returns [“ss”, “ss”]

**Asterisk(\*)** – Matches any one or more same characters that are specified right before it. E.g myString = “Mississippi’s”,

*“myString.match(/s\*/)*” returns [“ss”, “ss”, “s”]

***Curly Braces {min number of occurences, <optional else infinity>Max Number of occurences}*** – Matches any custom numbers of occurences. E.g.

*If regex = /^[A-Za-z]{2,}\d\*/*

*Regex.test(“JackOfAllTrades2023”)* returns true:

* Its starts with an alphabets,
* the digits are at the end,
* it is more that 2 characters long.

***Question Mark(?)*** – Marches zero or one of a character. For example: If

myString = “Favorite”

/Favou?rite/.test(myString) returns true // ”Favorite” can also be written as “Favourite”

**Lazy and Greedy Matching**

**Lazy matching** – Where the regular expression returns the shortest strIng that matches the pattern specified. Denoted by the ***question mark (?)***

**Greedy matching** – Where the regular expression returns the shortest strung that matches the pattern specified. It is the default and is therefore not denoted by anything.

E.g myString = “titanic”

myString.match(/t[a-z]\*i/) returns [“titani”] // Greedy Matching

myString.match(/t[a-z]\*?i/) returns [“ti”] // Lazy matching

**Matching the Beginnings and Endings of a string**

To match characters at the beginning of a string, we use the ***caret(^)*** but outside any character set []. E.g. myString = “Hello Jack”,

/^Hel/.test(myString) returns “true”.

To match characters at the end of a string, we use the ***dollar sign ($)*** but outside any character set []. E.g. myString = “Hello Jack”,

/ck$/.test(myString) returns “true”

**Shorthands**

***\w (lowercase w)***– Matches all uppercase, lowercase letters, digits and the underscore(\_). E.g.

myString = “Hello World in 2023.”

myString.match(/\w/g).length; returns 16, excluding the spaces and the period.

***\W (UPPERCASE W)***– Matches all the characters that are not uppercase, lowercase letters, digits or the underscore. E.g.

myString = “Hello World in 2023.”

myString.match(/\W/g).length; returns 4, the spaces and the period.

***\d*** – Matches all the digits 0-9. E.g.

myString = “Hello World in 2023.”

myString.match(/\d/g).length; returns 4, the spaces and the period.

***\D*** – Matches all the non-digits/ non-numbers. E.g.

myString = “Hello World in 2023.”

myString.match(/\D/g).length; returns 16, the spaces and the period.

***\s*** – Matches whitespaces. E.g.

myString = “Hello World in 2023.”

myString.match(/\s/g).length; returns 3.

***\S*** – Matches non-whitespaces. E.g.

myString = “Hello World in 2023.”

myString.match(/\S/g).length; returns 17.

**Positive and Negative Lookaheads**

These are used to check for more than one pattern in a string. For Example:

“queer”.match(/q(?=u)/) returns [“q”] // Positive Lookahead – There is a “e” later in the string

“quick”.match(/q(?!e)/) returns [“q”] // Negative Lookahead – There is no “u” later in the string

/(?=\w{5})(?=\D\*\d{3})/.test(“abcde123”) returns “true” // There are 5 alphabets followed by 3 digits

**Capture Groups**

These are repeating patterns that are matched in a string. E.g. If:

myString = “regex regex”

myString.match(/((\w+)\s\1)/) returns [“regex regex”, “regex”] // The \1 is shorthand for the the pattern in brackets such that you don’t have to write it again. The return’s first value is because the first match is of the whole regex, and the second value is of the pattern in brackets, (the capture group)

To replace a section of a string that matches a capture group pattern, we use: the **dollar sign($)**

“Code Camp”.replace(/(\w+)\s\1/, ‘$2 $1’) returns “Camp Code” // $2 means the second capture group and $1 means the first capture group