REGEX (Regular Expression) TUTORIAL IN HINDI

1. What is Regex?

The term **regex** Stands for Regular Expression. The regular expression, **regex or regexp** is a sequence of different characters which is used to describe a search pattern or validate specific strings, document, sentence or any other character input.

Regular expressions use both basic and special characters. Basic characters are standard letters, numbers, and general keyboard characters, while all other characters are considered special.

Regex is supported in all the scripting languages (such as Perl, Python, PHP, and JavaScript); as well as general purpose programming languages such as Java; and even word processors such as Word for searching texts.

History of Regex:

In 1943, Warren S. McCulloch (Neuroscientist) and Walter Pitts (Logician) began to develop models describing how the human nervous system works. Their research focused on trying to understand how the brain could produce complex patterns using simple cells that are bound together. In 1956, mathematician Stephen Kleen described McCulloch-Pitts neural models with an algebra notation that he penned ‘regular expressions’. Influenced by Kleen’s notion, in 1968, mathematician and Unix pioneer, Ken Thompson, implemented the idea of regular expressions inside the text editor, ‘ed’. His aim was that ed users would be able to do advanced pattern matching in text files. ed soon evolved to have the functionality to search based on regular expressions – this is when regexes entered the computing world.

Examples:

Only Alphabets

/^[A-Za-z]+$/

Only Digits

^[0-9]*$

Alphanumeric

/^[a-zA-Z0-9]*$/

Password Strength

Complex: at least 1 lowercase letter, 1 uppercase letter, 1 number, 1 special character and be at least 8 characters long:

/(?=(.*[0-9]))(?=.*[\!@#$%^&*()\\[\]{}\-_+=~`|:;"'<>,./?])(?=.*[a-z])(?=(.*[A-Z]))(?=(.*)).{8,}/

2. Use of Regex?

There are following uses of regex:

- For Validation like form validation in website
- For url rewriting in .htaccess file
- To Find or match any string like find in visual studio editor

3. Limitations of Regex?

There are following limitations of regex:

- It can not solve everything.
- Having readability and performance issues.
- It is not for simple work.

Q4. Regex Commands?

Regex Commands divided into two categories, which are following:

- **Basic Regex Commands**
- **Advanced Regex Commands**

Basic Regex Commands: There are following categories in basic regex commands:

1. **Characters**
2. **Groups**
3. **Quantifiers**
4. **Anchors**

Advanced Regex Commands: There are following categories in advanced regex commands:

1. **Lookarounds**
2. **Flags**

Characters:

The character class is the most basic regex concept. It converts a small group of characters to match a larger number of characters.

Character	Description
\	Escape character
.	Any character
\s	whitespace
\S	Not white space
\d	Digit
\D	Not digit
\w	Word character
\W	Not word character
\b	Word boundary
\B	Not word boundary
^	Beginning of string
$	End of string

Groups:

To operate on all items in a group, use a group expression. You can use a group expression to apply an operator to a group or to find a specified text before or after each member of the group. The grouping operator is the parentheses, while the "|" is used todivide the elements.

Group	Description	Example
[ ]	Characters in brackets are matched.	**xz[atp]r** is an expression which matches with the following strings: **"xzar", "xztr", and "xzpr"**
[^]	Characters not in brackets are matched.	Suppose, **Ab[^pqr]** is an expression which matches only the following string: **"Ab"**
|	Either, or	**My (red|pink|blue) dress**
( )	Capturing the group	**A(xy)** is an expression which matches with the following string: **"Axy"**
[a-z]	It matches letters of a small case from a to z.	
[A-Z]	It matches letters of an upper case from A to Z.	
[0-9]	It matches a digit from 0 to 9.	

Quantifiers:

Quantifiers specify how many characters or expressions should be matched.

Quantifiers	Description	Example
*	0 or more	**Br*** is an expression which gives **"B", "Br", "Brr", "Brrr"**, and so on…
+	1 or more	**Br+** is an expression which gives **"Br", "Brr", "Brrr"**, and so on…
?	0 or 1	**Br+** is an expression which gives **"B", "Br", but not "Brrr"**,
{x}	Exact number of characters	**Br{3}** is an expression which gives the following string which contains 3 r's: **“Brrr"**
{x,}	It specifies an expression to its left for x or more times.	**Xb{3, }** is an expression which gives various strings containing at least 3 b's. Such strings are "**Xbbb", "Xbbbb**", and so on.
{min,max}	Range of characters	**Br{3,6} a** is an expression which provides two strings.Both strings are as follows:**“Brrrr" and “Brrrrr"**

Anchors:

Based on the current position in the string, determines whether the match will succeed or fail.

Quantifiers	Description
^	Beginning of the string
$	End of the string
\A	The match is at the beginning of the string.
\G	Beginning of the match
\Z	At the end of the string or before \n at the end of the string, the match occurs.
\z	Absolute end of the string
\B	No word boundary
\b	Word boundary

Lookarounds:

Sometimes we need to find only those matches for a pattern that are followed or preceded by another pattern.There's a special syntax for that, called "lookahead" and "lookbehind", together referred to as "lookaround".

Lookaround	Description	Example
(?=check)	Positive Lookahead	X(?=Y), it means "look for X, but match only if followed by Y".
(?!check)	Negative Lookahead	X(?!Y), it means "search X, but only if not followed by Y".
(?<=check)	Positive Lookbehind	(?<=Y)X, matches X, but only if there's Y before it.
(?<!check)	Negative Lookbehind	(?<!Y)X, matches X, but only if there's no Y before it.

Flags:

A flag is a parameter that can be added to a regex to change how it searches. A flag modifies a regular expression's default searching behaviour. It does a regex search in a unique method.

Flag	Description
i	Makes the expression search case-insensitively.
g	Makes the expression search for all occurrences.
s	Makes a wild character. match newlines as well.
m	Instead of matching the beginning and conclusion of the entire string, the boundary characters ^ and $ match the beginning and ending of each individual line.

How to work with Regex?

There are following best practices to work with regex:

1. Define a scenario

Using natural language to define the problem will give you a better idea of the approach to use. The words **could** and **must**, used in a definition, are useful to describe mandatory constraints or assertions.

Example:

- The string must start with ‘h’ and finish with ‘o’ (e.g. hello, halo).
- The string could be wrapped in parentheses.

2. Develop a plan

After having a good definition of the problem, we can understand the kind of elements that are involved in our regular expression:

- What are the types of characters allowed (word, digit, new line, range, …)?
- How many times must a character appear (one or more, once, …)?
- Are there some constraints to follow (optionals, lookahead/behind, if-then-else, …)?

3. Implement/Test

It’s very important to have a real-time test environment to test and improve your regular expression. There are websites like **regex101.com**, **regexr.com** and **regexpal.com** that provide some of the best environments.

To improve the efficiency of the regex, you could try to answer some of these additional questions:

- Are the character classes correctly defined for the specific domain?
- Should I write more test strings to cover more use cases?
- Is it possible to find and isolate some problems and test them separately?
- Should I refactor my expression with subpatterns, groups, conditions, etc., to make it smaller, clearer and more flexible?

Practical Examples?

There are some practical examples following:

Only Alphabets:

/^[A-Z]+$/I

/^[A-Za-z]+$/

Only Digits

/^[0-9]+$/

Alphabets with Space

/^[a-zA-Z]*$/

/^[a-zA-z\s]+$/

Alphanumeric

/^([A-Za-z]|[0-9]|_)+$/

/^\w+$/

Only 10 Digit numeric

/^[0-9]{10}$/

Indian Mobile Number Validation

/^[6-9]\d{9}$/

Strong Password

/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{6,12}$/

Email Validation

/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/

Some websites for Practice Regex:

There are some websites for practice regex, which are following:

https://regex.sketchengine.eu/tutorial.html

https://regexone.com/

https://www.w3resource.com/python-exercises/re/