# Pattern Matching using Regular Expression

**Pattern matching :**

Pattern matching, involves the use of one-dimensional string matching.
Patterns are either tree structures or sequences.
There are different classes of programming languages and machines which make use of pattern matching.
In the case of machines, the major classifications include deterministic finite state automata, deterministic pushdown automata, nondeterministic pushdown automata and Turing machines.
Regular programming languages make use of regular expressions for pattern matching.
Tree patterns are also used in certain programming languages like Haskell as a tool to process data based on the structure.
Compared to regular expressions, tree patterns lack simplicity and efficiency.
There are many applications for pattern matching in computer science.
High-level language compilers make use of pattern matching in order to parse source files to determine if they are syntactically correct.
In programming languages and applications, pattern matching is used in identifying the matching pattern or substituting the matching pattern with another token sequence.

**Regular Expression :**

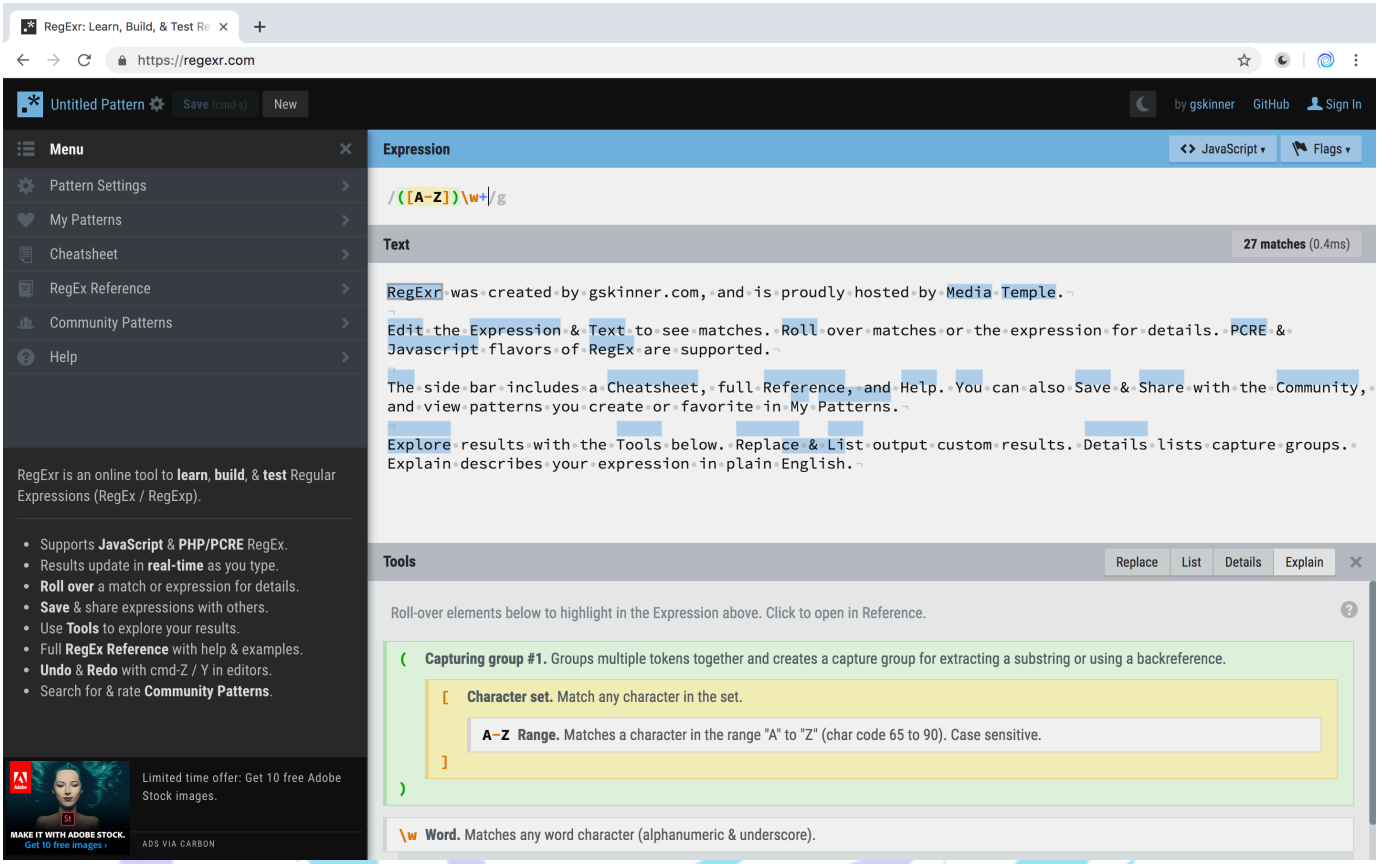
A regular expression in a programming language is a special text string used for describing a search pattern.
It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.
While using the regular expression the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string.
Ascii or latin letters are those that are on your keyboards and Unicode is used to match the foreign text.
It includes digits and punctuation and all special characters like $#@!%, etc.
Regular expression could tell a program to search for specific text from the string and then to print out the result accordingly. Expression can include

- Text matching
- Repetition
- Branching
- Pattern-composition etc.

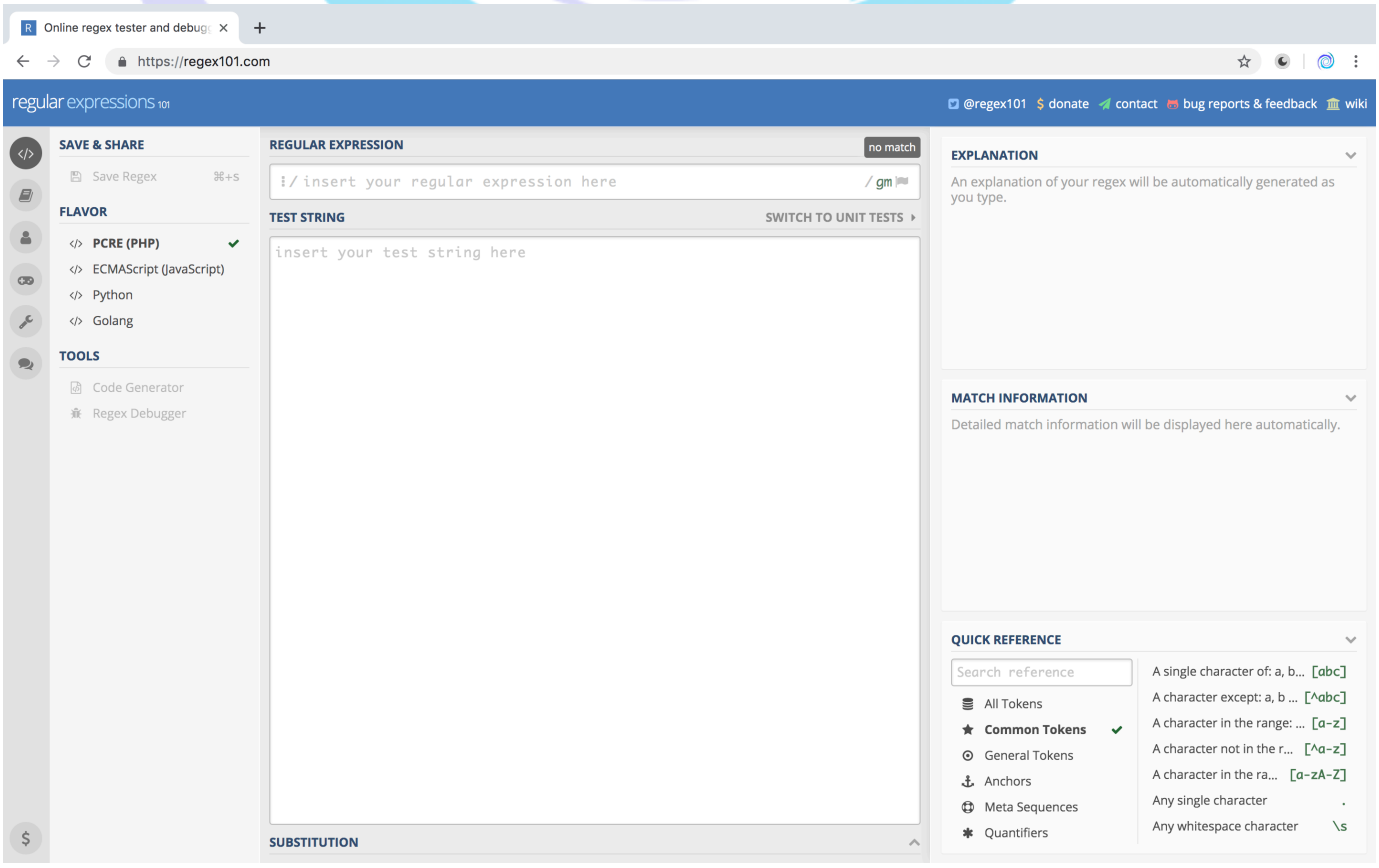
In Python, a regular expression is denoted as RE (REs, regexes or regex pattern) are imported through re module.
Python supports regular expression through libraries.
In Python regular expression supports various things like Modifiers, Identifiers, and White space characters.

To learn regular expressions we can use below sites

https://regexr.com/




https://regex101.com/



# Important concepts of regular expressions

## Metacharacters :

Metacharacters are characters with a special meaning

| Character | Description | Example |
|---|---|---|
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "world$" |
| * | Zero or more occurrences | "aix*" |
| + | One or more occurrences | "aix+" |
| {} | Exactly the specified number of occurrences | "al{2}" |
| \| | Either or | "falls\|stays" |
| () | Capture and group | |

## Special Sequences :

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

| Character | Description | Example |
|---|---|---|
| \A | Returns a match if the specified characters are at the beginning of the string | "\AThe" |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word | r"\bain" r"ain\b" |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word | r"\Bain" r"ain\B" |
| \d | Returns a match where the string contains digits (numbers from 0-9) | "\d" |
| \D | Returns a match where the string DOES NOT contain digits | "\D" |
| \s | Returns a match where the string contains a white space character | "\s" |
| \S | Returns a match where the string DOES NOT contain a white space character | "\S" |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character) | "\w" |
| \W | Returns a match where the string DOES NOT contain any word characters | "\W" |
| \Z | Returns a match if the specified characters are at the end of the string | "Spain\Z" |

## Sets :

A set is a set of characters inside a pair of square brackets [] with a special meaning

| Set | Description |
|---|---|
| [arn] | Returns a match where one of the specified characters ( `a` , `r` , or `n` ) are present |
| [a-n] | Returns a match for any lower case character, alphabetically between `a` and `n` |
| [^arn] | Returns a match for any character EXCEPT `a` , `r` , and `n` |
| [0123] | Returns a match where any of the specified digits ( `0` , `1` , `2` , or `3` ) are present |
| [0-9] | Returns a match for any digit between `0` and `9` |
| [0-5][0-9] | Returns a match for any two-digit numbers from `00` and `59` |
| [a-zA-Z] | Returns a match for any character alphabetically between `a` and `z` , lower case OR upper case |
| [+] | In sets, `+` , `*` , `.` , `\|` , `()` , `$` , `{}` has no special meaning, so `[+]` means: return a match for any `+` character in the string |