Regular Expressions

Regular Expressions (Regexes) are a language for searching for and matching patterns in text. Regexes are supported in just about all programming languages. They are a must for anyone doing serious computing that involves text.

Literal Characters

The most basic regular expression consists of a single literal character, such as `a`. It matches the first occurrence of that character in the string. If the string is `Jack is a boy`, it matches the `a` after the `J`.

Character Classes or Character Sets

A "character class" matches **only one out of several characters**. To match an a or an e, use `[ae]`. You could use this in `gr[ae]y` to match either `gray` or `grey`.

You can use **a hyphen inside a character class to specify a range of characters**. `[0-9]` matches a *single* digit between 0 and 9.

You can use more than one range. `[0-9a-fA-F]` matches a single hexadecimal digit, case insensitively. You can combine ranges and single characters. `[0-9a-fxA-FX]` matches a hexadecimal digit or the letter X.

Negation

Typing a caret after the opening square bracket negates the character class. The result is that the character class matches any character that is *not* in the character class. `q[^x]` matches `qu` in `question`. It does *not* match `Iraq` since there is no character after the `q` for the negated character class to match.

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any digit between the brackets
[^0-9]	Find any digit NOT between the brackets

Shorthand Character Classes (Metacharacters)

Some of the special sequences beginning with `'\'` represent predefined sets of characters that are often useful, such as the set of digits, the set of letters, or the set of anything that isn't whitespace. The following predefined special sequences are a subset of those available.

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\d	Find a digit
\s	Find a whitespace character

Negated Shorthand Character Classes

The above also have negated versions. `\D` is the same as `[^\d]`, `\W` is short for `[^\w]` and `\S` is the equivalent of `[^\s]`.

Metacharacter	Description
\W	Find a non-word character
\D	Find a non-digit character
\S	Find a non-whitespace character

The Dot Matches (Almost) Any Character

In regular expressions, the dot or period is one of the most commonly used metacharacters. Unfortunately, it is also the most commonly misused metacharacter.

The dot matches a single character, without caring what that character is. The only exception are line break characters. In all regex flavors discussed in this tutorial, the dot does *not* match line breaks by default.

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X* *n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s

Start of String and End of String Anchors

Thus far, we have learned about literal characters, character classes, and the dot. Putting one of these in a regex tells the regex engine to try to match a single character.

Anchors are a different breed. They do not match any character at all. Instead, they match a position before, after, or between characters.

The caret `^` matches the position before the first character in the string. Applying `^a` to `abc` matches `a`. `^b` does not match `abc` at all, because the `b` cannot be matched right after the start of the string, matched by `^`. See below for the inside view of the regex engine.

Similarly, `$` matches right after the last character in the string. `c$` matches `c` in `abc`, while `a$` does not match at all.

Alternation with Vertical Bar or Pipe Symbol

If you want to search for the literal text `cat` or `dog`, separate both options with a vertical bar or pipe symbol: `cat|dog`. If you want more options, simply expand the list: `cat|dog|mouse|fish`.

The alternation operator has the lowest precedence of all regex operators. That is, it tells the regex engine to match either everything to the left of the vertical bar, or everything to the right of the vertical bar.

Greedy vs. Lazy

Greediness: Greedy quantifiers first tries to repeat the token as many times as possible, and gradually gives up matches as the engine backtracks to find an overall match.

Laziness: Lazy quantifier first repeats the token as few times as required, and gradually expands the match as the engine backtracks through the regex to find an overall match.

Greedy quantifier	Lazy quantifier	Description
*	*?	Match zero or more times.
+	+?	Match one or more times.
?	??	Match zero or one time.
{n}	{n}?	Match exactly n times.
{n,}	{n,}?	Match at least n times.
{n,m}	{n,m}?	Match from n to m times.

> Add a ? to a quantifier to make it ungreedy i.e lazy.

Example:
test string : *stackoverflow*
greedy reg expression : `s.*o` output: **stackoverflo**w
lazy reg expression : `s.*?o` output: **stacko**verflow

Using Back references To Match Same Text Again

Back references match the same text as previously matched by a capturing group. Suppose you want to match a pair of opening and closing HTML tags, and the text in between. By putting the opening tag into a back reference, we can reuse the name of the tag for the closing tag. Here's how: `<([A-Z][A-Z0-9]*)\b[^>]*>.*?</\1>`.

This regex contains only one pair of parentheses, which capture the string matched by `[A-Z][A-Z0-9]*`. This is the opening HTML tag. (Since HTML tags are case insensitive, this regex requires case insensitive matching.) The backreference `\1` (backslash one) references the first capturing group. `\1` matches the exact same text that was matched by the first capturing group. The `/` before it is a literal character. It is simply the forward slash in the closing HTML tag that we are trying to match.

To figure out the number of a particular backreference, scan the regular expression from left to right. Count the opening parentheses of all the numbered capturing groups. The first parenthesis starts backreference number one, the second number two, etc.

Skip parentheses that are part of other syntax such as non-capturing groups. This means that non-capturing parentheses have another benefit: you can insert them into a regular expression without changing the numbers assigned to the backreferences. This can be very useful when modifying a complex regular expression.

You can reuse the same backreference more than once. `([a-c])x\1x\1` matches `axaxa`, `bxbxb` and `cxcxc`.

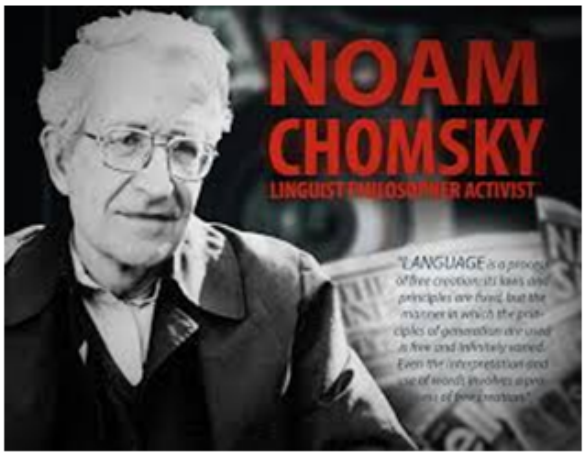
Most regex flavors support up to 99 capturing groups and double-digit backreferences. So `\99` is a valid backreference if your regex has 99 capturing groups.

Regular expression tester: http://regexpal.com/

Very helpful site.

The World of Grammars

1950s Noam Chomsky – Philosopher/Linguist



All written languages, L have:

- an alphabet : define by listing all the accepted letters in L

- words : define letter combinations with a Dictionary for L

- sentences : legal arrangement of words in L. May be infinite or close.

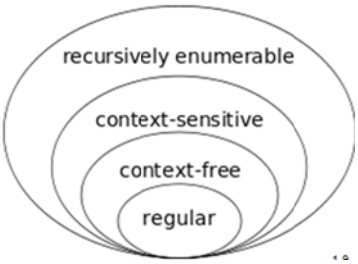
Chomsky: we need a way to determine if some string of words is a valid sentence in language. He formalized the concept of a grammar.

REGEX in ANTLR:

Antlr uses a subset of REGEX syntax:

- DIGIT : [0-9] ;
- NUMBER : DIGIT+ ;
- LETTER : [A-Za-z] ;
- ID : LETTER (LETTER | DIGIT)* ;

Chomsky identified **four kinds of languages** -- from simple (Regular) to the most complex Recursively Enumerable (e.g. English)



Regexes are a considered a kind of computer language. In the 1950s, Noam Chomsky, a linguist at MIT developed a theory of language and showed how well-defined grammars could be used to categorize different kinds of languages. In his work he identified four different kinds of languages based on the struture of the grammar rules.

Among the grammars are regular grammars for regexes and contect free grammars for high-level programmin languages. Chomsky's work was quickly adopted by comptuer scientists who used grammars to define the language ALGOL In the 1960s. John Backus, one of the developers of FORTRAN created a technology called BNF (Backus-Naur Form) that became the standard for defining languages.