Introduction: Regular Expressions (regexp)

A regular expression (regexp) is a powerful notational algebra that describes a string or a set of strings that are used to find patterns (**pattern matching**). Pattern matching is defined as a true or false answer – in other words, if the pattern described in the regex is found in a string of letters, the answer is true. If the pattern is not found the answer is false.

Regular expressions are made up of **terms**, **operators** and **modifiers**.

Terms are the strings or substrings, for example the term "Kinase", will match to substrings. In the string "Protein Kinase", "Kinase" is then a term that matches a substring.

Operators combine terms and expressions. For example

grouping with expression like ([0-9]+)

alternation with expression or literal characters like zinc | transmem

repetition with * + ? {min,max} specifies how many times the preceding expression may match.

But what about concatenation (combining two substrings)? That is implicit. If you want to combine two expressions you can simply specify them as such.

Operators have precedence, like arithmetic operators. By grouping the expression you can change the precedence.

The regexp MA{5} would match MAAAAA whereas

(MA){5} would match MAMAMAMAMA.

Modifiers change the rules, like the compilation flag IGNORECASE we can set with re.I or re.IGNORECASE. Other flags are re.M for multiline, re.S for to make '.' to match any character including newline.

Protein Motifs can be described by regular expressions.
Most often the binding sites of proteins have particular requirements that limits the number of residues as well as the amino acids that are part of the binding site. Clustering these motifs would enable us to define a pattern for this particular site. Once you make the pattern and make it regexp friendly, it is easy to write computer programs that can search a new protein sequence to find these motifs.

Writing regular expressions:

The necessary regexp notations we need to represent aa in a motif are

N-terminal residue
C-terminal residue
Any residue
Optional residue
Representation of gaps (i.e. variable length regexps)

Before doing that let us familiarize ourselves with the following regexp metacharacter table

AA property	Amino acids	Code
Acidic	DE	0
Alcohol	ST	1
Aliphatic	ILV	2
Aromatic	FHWY	3
Basic	KRH	4
Charged	DEHKR	5
Hydrophobic	AVILMFYW	6
Hydrophilic	KRHDENQ	7
Polar	CDEHKNQRST	8
Small	ACDGNPSTV	9
Tiny	AGS	B
Turnlike	ACDEGHKNQRST	Z
Any	ACDEFGHIKLM NPQRSTVWY	.

Other useful metacharacter:

Meta Character	Name	Meaning	How do we use them?
.	dot	Any character except newline	Any aa/gap
[a-z]	character class	Match any char from a to z	amino acid in one letter code
[^a-z]	negated character class	Match all except a-z	reject certain amino acids
?	optional character	If the previous char is there match it, otherwise don't care otherwise don't care	optionally match previous amino acid
*****	star	match zero or more time	find >=0 'clones'
+	plus	match one or more time	find >=1 'clones'
^	caret	Match at the	N terminal

		beginning	
$	dollar	Match at the end	C-terminal
|	alternation	Match either expression it separates	Match either or
{m,n}	range specifier (operator)	m minimum required n	Match variable number of aa's or allow variable gaps

Examples:

Find genes whose protein product contains a motif pattern that you specify., e.g. "CC6+RK", which means "two cysteines followed by one or more hydrophobic amino acids, followed by arginine, then lysine". The pattern need not be well conserved. If you can describe it in words, you can probably use this tool to create an "expression" that can search other proteins for similar patterns.

Another example is the pattern of the Pexel motif, which can be represented as "R.L.[EQD]", meaning "an arginine, then any amino acid, then a leucine, then any amino acid, then either an aspartic acid, a glutamic acid, or a glutamine".