BST281: Genomic Data Manipulation, Spring 2017

Wednesday 09: Regular Expressions

Regular expressions (also REs or regexes or regexps) are a special language describing search patterns within text

 REs s*earch* - does a pattern occur in a text?

 REs r*eplace* - take out whatever's in a pattern and replace it with something else

 REs c*apture groups* - tell me exactly what string contents matched some wildcard(s) or pattern

Default character classes

 The wildcard . matches any single character

 \d matches a digit (\D a non-digit), \w matches a “word” character (A-Z, a-z, 0-9, \_; \W a non-word character)

 \s matches a whitespace character; \t and \n match “tab” and “newline” specifically (as in Python)

Custom character classes

 [ABC] matches A or B or C

 [a-d] matches a or b or c or d; [0-3] matches 0 or 1 or 2 or 3

 [^ABC] matches any character except A, B, C

Boundaries

 ^ and $ match the beginning and end of a string, respectively

 \b matches a “word boundary” (non-word character OR start/end of string)

Repetition

 A? matches an optional A

A+ matches 1 or more As while A\* matches 0 or more As

 A{*n*} matches exactly *n* As

A{*n*,*m*} matches between *n* and *m* As (inclusive)

A{n,} matches at least *n* As while A{,m} matches at most *m* As

Sub-patterns and capture groups

 Use ( )s to define a (possibly repeating) sub-pattern

 Such sub-patterns are “captured” for use later; use (?:*sub-pattern*) to avoid capturing

(AA|BB)+ matches one or more instances of sub-pattern AA or BB (logical OR)

Regular expressions in Python

the re module provides access to Python’s regular expression engine (import re to use)

 re.search( *pattern*, *text* )

 Returns the first match of *pattern* in *text* if one is found, otherwise returns None

re.finditer( *pattern*, *text* )

 Used in a for loop to find multiple matches

 Matches are returned as special Match objects (data + associated functions)

Match.start( ) and Match.end( ) return Python-style start/end coordinates of the match in the text

 Match.group( *n* ) returns the *n*th captured group; Match.groups( ) returns all captured groups

 re.sub( *find*, *replace*, *text* )

Replaces all instances of string *find* with string *replace* in string *text*; returns a modified string

 Use [\\1](file:///%5C%5C1), [\\2](file:///%5C%5C2), etc. to refer to groups captured in *find* within *replace*

Miscellany

 REs are “greedy”

Reading left-to-right, find longest match; proceed to next-longest, non-overlapping match; and so forth

Change this behavior with ?: re.search( r“A.\*?B”, “ACBCB” ) matches ACB and not ACBCB

REs expand the capabilities of command-line tools such as grep and sed

REs are prone to false positives and false negatives (review patterns and matches carefully!)

# Reading

REs in Python: Haddock and Dunn, Chapter 2 p17-29, Chapter 3 p31-43

 <https://docs.python.org/3/howto/regex.html> & <https://docs.python.org/3/library/re.html>