1.0.1.8.7

Introduction to Perl
Session 7

- global searches
- context of =~
- replacement operator

Recap of substr()

- we’ve seen how substr() can be used to manipulate a string
  - extract, insert, replace, remove
- regions affected by substr() are defined by position, **not content**

```
# get the first 5 characters
substr($string,0,5);

# insert “abc” at position 3 (after 3rd character)
substr($string,3,0,”abc”);

# replace first 5 characters with “abc”
substr($string,0,5) = “abc”;
# replace first 5 characters and retrieve what was replaced
$old = substr($string,0,5,”abc”);

# remove 5 characters at position 3
substr($string,3,5,””);
```

Recap of =~

- we used the operator =~, which binds a string to a pattern match, to test a string using a regular expression

```
if( $string =~ /REGEX/ ) { ... }
```

- so far, we only tested whether the regex matched
- we will now look at how to extract
  - what was matched
    - a*b can match b, ab, aab, aaab, ...
  - how many times a match was found
  - where in the string a match was found
- we will also see how to use the replacement operator s/// to replace parts of a string which match a regex

Capturing Matches

- capture brackets are used to extract parts of a string that matched a regex
- text captured is available via special variables

```
$string = “sheep”;

if ( $string =~ /e*p/ ) {
  # we know it matched, but we don’t know what part of $string matched
}

if ( $string =~ /(e*p)/ ) {
  # text within capture brackets available in pattern buffer $1
  $matched = $1;
  print “$matched in $string matched”;
}

eep in sheep matched
```

Pattern Buffers

- the pattern buffers $1, $2, $3 store the text matched within first, second, third, … set of capture brackets
  - $n is an empty string if no text matched

```
$string = “53 big sheep”;

if ( $string =~ /(\d+) \w+ (\w+)/ ) {
  ($number,$animal) = ($1,$2);
  print “saw $number $animal”;
}

saw 53 sheep
```

Pattern Buffers

- buffers are reassigned values on each successful search

```
$string = “53 big sheep”;

if ( $string =~ /(\d+) \w+ (\w+)/ ) {
  $string =~ /(pig)/;   # pattern buffers not reset
  $string = ~ /(.ig)/;  # pattern buffers reset
  ($number,$animal) = ($1,$2);
  print “saw $number $animal”;
}
```

- be careful when using $n, since values may become reset or go out of scope
  - $n defined until end of current code block or next successful search, which ever first

- use special variables @- and @+ to determine number/location of submatches
  - @- match start
  - @+ match end

Bypassing Pattern Buffers

- the match operator can return the matched text directly, depending on the context
  - in scalar context, =~ returns the number of captured matches
  - in list context, =~ returns the text of captured matches

- we have already seen the use of =~ in scalar context

```
$string = “53 big sheep”;

# scalar context, no capture brackets – returns 0/1 match success
my $result = $string =~ /\w/;      $result → 1
```

- now we turn to =~ in list context

Match List Context

- =~ will return the patterns that matched within the capture brackets

```
$string = “53 big sheep”;
my @matches = $string =~ /(\w)(\w) (\w)/;

@matches → qw(5 3 b)
```

- remember that the pattern buffers $1, $2, $3 will store the contents captured by the brackets
- several special variables store pattern buffer result
  - @+ stores offsets of the end of each pattern match
  - @- stores offsets of the start of each pattern match
  - $+ stores the last pattern match
  - $#- or $#+ stores the number of patterns matched
- $n can be expressed as substr($string, $+[n] , $+[n] - $-[n]);

$+ and @+ and @-

- three special variables help interrogate the search results

```
$string = “0123456789”;

my @matches = $string =~ /.([1-3]+)..([6-8]+)/;

# $+ stores the last successfully matched subpattern
print $+;
678

# @- stores the positions of match starts of subpatterns
# $-[0] holds the offset of start of the whole match
print @-;
0 1 6

# @- stores the positions of match ends of subpatterns
# $-[0] holds the offset of end of the whole match
print @+;
10 4 9
```

Global Matching

- so far, we’ve written a regular expression that may match multiple parts of interest in a string

```
$clone = “M0123B03”;
if ($clone =~ /(\w)(\d{4})(\w)(\d{2})/) {
  ($lib,$plate,$wellchr,$wellint) = ($1,$2,$3,$4);
}
```

- we can find all match instances of a regular expression by using global matching
  - global matching is toggled using /g flag

- in a list context, a global match will return all matches on a string to a pattern

```
$string = “53 big sheep”;
@matches = $string =~ /[aeiou]/g;

@matches → qw( i e e )
```

Example with /g

- extracting all subsequences matching a regex

```
# random 1000-mer
$seq = make_sequence(bp=>"agtc",len=>1000);

# all subsequences matching at.gc
@match = $seq =~ /at.gc/g;

print @match;

sub make_sequence {
  %args = @_;
  @bp = split("",$args{bp});
  $seq = "";
  for (1..$args{len}) {
    $seq .= $bp[rand(@bp)];
  }
  return $seq;
}

atcgc atagc atagc
```

/g with capture brackets

- capture brackets can be used with /g to narrow down what is returned
- if no capture brackets are used, /g behaves as if they flanked the whole pattern
  - /at.gc/g equivalent to /(at.gc)/g

```
# random 1000-mer
$seq = make_sequence(bp=>"agtc",len=>1000);

# all subsequences matching at.gc
@match = $seq =~ /at(.)gc/g;

print @match;

c a a
```

/g with multiple capture brackets

- if you have multiple capture brackets in a /g match, each matched subpattern will be added to the list

```
$string = “a1b2c3”;

# on each iteration of the match two elements will be pushed onto the list
@match = $string =~ /(.)(.)/g;

print @match;

a 1 b 2 c 3
```

/g in scalar context

- in scalar context, the global match returns 0 or 1 based on the success of the next match in the string
  - it keeps track of the previous match
  - used in conjunction with while

```
$seq = make_sequence(bp=>"agtc",len=>1000);

while ($seq =~ /(at.gc)/g) {
  $match = $1;
  print “matched $match”;
}

matched atcgc
matched attgc
matched attgc
matched atcgc
```

/g in scalar context

- to determine where the match took place, use pos
    - pos $string returns the position after the last match

```
$seq = make_sequence(bp=>"agtc",len=>1000);

while ($seq =~ /(at.gc)/g) {
  $match = $1;
  $matchpos = pos $seq;
  print "matched $match at ",$matchpos-5,” around “,substr($seq,$matchpos-7,9);
}

matched atcgc at 106 around ccatcgccc
matched atggc at 241 around atatggcga
matched atggc at 271 around agatggctc
matched attgc at 507 around tcattgcgc
```

Manipulating Search Cursor

- pos($string) returns the current position of the search cursor
  - within a while loop, this is the position at the end of the last successful match
- you can adjust the position of the cursor by changing the value of pos($string)
  - pos can act like an l-value (just like substr)

```
$seq = make_sequence(bp=>"agtc",len=>10);

while ($seq =~ /(..)/g) {
  print "matched $1 at “, pos $seq;
  # back up the cursor one character
  pos($seq)--;
}

attgatgatt
matched at at 2
matched tt at 3
matched tg at 4
matched ga at 5
...
```

- adjusting cursor position is the only way to return overlapping search results
  - in this example, we return all pairs of adjacent bases in the string, not just abutting ones
- a search finds pair bp[i]bp[i+1] and the cursor is at i+2 at the end of the search
- to find bp[i+1]bp[i+2] we need to back the cursor up to i+1

Replacement Operator

- we have seen how `substr()` can be used to replace subtext at specific position
- what if we want to replace all occurrences of one substring with another?
  - we use `s/REGEX/REPLACMENT/`
  - REPLACEMENT is not a regular expression – it is a string

```
$seq = make_sequence(bp=>"agtc",len=>60);

print $seq
# replaces first substring matching “a” with “x”
$seq =~ s/a/x/;
print $seq;

gtattgtgggaccttcctttcatcccgaagcattccgcgatgtggtccccggacctcagt
gtxttgtgggaccttcctttcatcccgaagcattccgcgatgtggtccccggacctcagt

# /g forces replacement everywhere
$seq =~ s/a/x/g;
print $seq;

gtxttgtgggxccttcctttcxtcccgxxgcxttccgcgxtgtggtccccggxcctcxgt
```

Replacement Operator

- s/// works nicely with capture brackets

```
$seq = make_sequence(bp=>"agtc",len=>40);

print $seq
$seq =~ s/(a)/($1)/g;

cccgttaggctgtaccgaacaagtactaacaaagttacta
cccgtt(a)ggctgt(a)ccg(a)(a)c(a)(a)gt(a)ct(a)(a)c(a)(a)(a)gtt(a)ct(a)
```

- here we refer to the successfully captured pattern buffer as $1 in the replacement string
- s/// returns the number of replacements made

Replacement Operator

- remember that the replacement string is not a regular expression, but a regular string which may incorporate $1, $2, etc

```
$seq = make_sequence(bp=>"agtc",len=>40);

print $seq;
$seq =~ s/..(a)../..$1../g;
print $seq;

cccgtcaattgtttagtttactttaaaagtaacgaatttc
cccg..a..tgt..a....a....a..a..a....a..tc
```

/e with Replacement Operator

- the replacement operator has a allows you to execute the replacement string as if it were Perl code

```
$string = “12345”;

$seq =~ s/(\d)/1+$1/eg;
print $seq;

23456
```

- in this example, the replacement is global, so it continues to replace all instances of \d
- for each instance (a digit) it replaces it with 1+$1 (e.g. 1+2, 1+3, 1+4...)
- before the replacement is made, it evaluates the expression (e.g. to yield 3, 4, 5...)

Example of /e

- replace all occurrences of a given basepair with a random base pair

```
$seq = make_sequence(bp=>"agtc",len=>40);

print $seq;
$seq =~ s/a/make_sequence(bp=>”agtc”,len=>1)/eg;
print $seq;

gtcccttgacaccatactggccggatacgtgagcccacga
gtcccttggcgccattctggccgggttcgtgagcccgcgc
```

- /e is very powerful, but be diligent in its use
  - you are creating and evaluating Perl code at run time
  - some obvious security issues come to mind, if the code depends on user input

Example of /e

- a common use of /e is to use sprintf to reformat the matched string

```
# replace all numbers with decimals with 3-decimal counterparts
$seq =~ s/(\d+\.\d+)/sprintf(“%.3f”,$1)/eg;
```

- if you’re working for a dictatorship, you could use this censoring one-liner

```
# replace 40 characters on left/right of a keyword
# with [censored NNN characters] message
$seq =~ s/(.{40}government.{40})/sprintf(“[censored %d characters]”,length($1))/eg;
```

Transliteration with tr///

- a quick and dirty replacement can be made with the transliteration operator, which replaces one set of characters with another
  - tr/SEARCHLIST/REPLACEMENTLIST/

```
$seq = make_sequence(bp=>"agtc",len=>40);

print $seq;
$seq =~ tr/atgc/1234/;
print $seq;

ttgagtgatcagcgtgctcccgtaatggtcagaaaaacag
2231323124134323424443211233241311111413
```

- in this example, a→1 t→2 g→3 c→4

Transliteration with /d - deletion

- you can use tr to delete characters
  - /d deletes found but unreplaced characters

```
$seq = make_sequence(bp=>"agtc",len=>40);

print $seq;
$seq =~ tr/at//d;
print $seq;

ccgcgttgcgatgcttgattgaatttcagacccggcctgt
ccgcggcggcggcgcccggccg

print $seq;
$seq =~ tr/gcat/12/d;
print $seq;
ggtcctccaacaggagtttacgttaatgattgtgcaaagg
112222211121111211
```

Transliteration with /s - squashing

- /s squashes repeated transliterated characters into a single instance
  - helpful to collapse spaces

```
$x = "1223334444";

$x =~ tr/1234/abcd/     # abbcccdddd
$x =~ tr/1234/abcd/s    # abcd

$y = "1 22  333   4444";

$y =~ tr/ /_/s          # 1_22_333_4444
$y =~ tr/ / /s          # 1 22 333 4444
$y =~ tr/ //s           # 1 22 333 4444    same as above
```

- if you do not provide a replacement list, then tr will squash repeats without altering rest of string

```
$x = "1 22  333   4444";

$x =~ tr/0-9//s         # 1 2  3   4
$x =~ tr/0-9 //s        # 1 2 3 4
```

Transliteration returns number of replacements

- number of transliterations made is returned
  - use this to count replacements, or characters

```
$x = "1 22  333   4444";

$cnt = $x =~ tr/1234/abcd/    # $x → abbcccdddd    $cnt → 10
$cnt = $x =~ tr/0-9//         # $x unchanged       $cnt → 10

$y = "encyclopaedia";

$cnt = $y =~ tr/aeiou//       # $y unchanged    $cnt → 6

# /c complements the search list - i.e., replace all non-vowel characters
$cnt = $y =~ tr/aeiou//c      # $y unchanged    $cnt → 7
```

1.0.8.1.7

Introduction to Perl
Session 7

- you now know
  - context of match operator
  - replacing text with `s///`
  - use of transliteration `tr///`