Programming in Perl
Lecture 3 Examples

Formated print statement:

```
#!/usr/interp/perl -w

$string = "Hello world";
$integer = 785;
$float = 134.675;
printf("%20s %d %10.2f\n", $string, $integer, $float);
```

Output:

```
         Hello world 785     134.68<newline>
```

Regular Expressions:

Predefined Character Classes:

```
                          Equivalent     Negated   Equivalent
Name         Construct      Class       Construct    Class
----         ---------   ----------     ---------  ----------
Digit           \d       [0-9]             \D      [^0-9]
Word char       \w       [a-zA-Z0-9_]      \W      [^a-zA-Z0-9_]
Space char      \s       [ \r\t\n\f]       \S      [^ \r\t\n\f]
```

Multipliers:

```
Construct              Allowed Range
---------              -------------
{n, m}                 Pattern must occur AT LEAST n times
                       but NO MORE than m times
{n,}                   Pattern must occur AT LEAST n times
{n}                    Pattern must occur EXACTLY n times
*                      0 or more times (same as {0,})
+                      1 or more times (same as {1,})
?                      0 or 1 times (same as {0,1})
```

Use of multipliers:

```
/a{4,6}/ # matches 4, 5, or 6 a's
/a{4,}/  # matches 4 or more a's
/a{4}/   # matches exactly 4 a's
/a{0,4}/ # matches 0 to 4 a's (4 or fewer)

/foo{3}/ # matches foooo (or longer)
         # note that once foooo is matched, the line is matched,
         # so fooooo is also ok
/(foo){3}/ # matches foofoofoo
```

Greedy vs. Lazy Regex Evaluation:

```
Given

  "barbarbarfoobarfoobarfoo"

  /\w*foo/ # matches entire barbarbarfoobarfoobarfoo
           # greedy evaluation by default; \w* goes
           # to the end of the string, then it has to
           # backtrack to check for 'f' followed by 'o'
           # followed by 'o'
Given

  "barbarbar!foobarfoobarfoo"

  /\w*foo/ # matches foobarfoobarfoo portion
           # the \w* was happy with the first
           # part of the string until it hit the !
           # then it had to restart processing
Given

  "a xxx c xxxxx c xxx d"
  /a.*c.*d/ # the first .* matches up to the
            # the SECOND 'c' ; leftmost is greediest

Given

  "barbarbarfoobarfoobarfoo"

  /\w*?foo/ # matches barbarbarfoo
            # forces lazy evaluation; \w* matches
            # only to the first 'f', then it starts
            # checking for an 'o' followed by
            # another 'o'

Given

  "a xxx c xxxxx c xxx d"
  /a.*?c.*d/ # the first .*? now matches up to the
             # the FIRST 'c'; the rest is picked
             # up by the next .*
```

Parentheses as memory:

```
$_ = "apples pears peaches plums";
/(\w*)\s+(\w*)\s+(\w*)/;
# matches 0 or more word characters followed by
# 1 or more space characters, etc.
print ("$1\n$2\n$3\n");
```

Output:

```
apples<newline>
pears<newline>
peaches<newline>
```

```
$_ = "apples pears peaches plums";
/(\w*)(\w*)(\w*)/;
# matches 0 or more word characters followed by
# 0 or more word characters, etc.
print ("$1\n$2\n$3\n");
```

Output:

```
apples<newline>
<newline>
<newline>
```

Reusing a previously matched pattern:

```
$_= "axxyxxb";
if (/a.{5}b/) {
  print("any five: $&\n");
}
# condition will evaluate to false
if (/a(.)\1{4}b/) {
  print("same five: $&\n");
}
```

```
$_ = "<B> bold text </B>";
if (/<([a-zA-Z0-9]*?)>.*?<\/\1>/) {
  print("bold: $&\n");
}
$_ = "<i> italic text </i>";
if (/<([a-zA-Z0-9]*?)>.*?<\/\1>/) {
  print("italic: $&\n");
}
```