CMSC 330 Ruby and Ruby Regular Expressions – Solutions Fall 2012

1. a. Shifted from efficiency to ease-of-programming

   b.
   - Naturalness of application - text processing is easier in Ruby
   - Cost of use Small Ruby programs are simpler/quicker to write

   c. Interpretation and compilation. Ruby is interpreted.

2. a.
```
def printit(x)
  print(x)
end

printit(2)
```
   x is the formal paramameter, 2 is the actual parameter

   b.
```
1.upto(10) {|i|
  puts(i)
}

(1..10).each() {|i|
  puts(i)
}
```

```
for i in (1..10)
  puts(i)
end

i = 1
while (i <= 10)
  puts(i)
  i += 1
end
```

```
i = 0
until ((i += 1) > 10)
  puts(i)
end
```

   Of course other versions are possible as well.

   c.
   - Explicit- declarations must specify the type of each variable used
   - Implicit- the first use of a variable declares it and determines its type

   d. Helps prevent subtle errors, catches more type errors at compile time

   e.
```
class Teacher

  @@totalStudents = 0

  def initialize
    @students = 0
    @@totalStudents += @students
  end

end
```

   f.
```
x = "a" ; y = x
```

   g.
```
x == y
```

3. a. Strings that contain two 3s.

b. Strings that contain an uppercase letter character.

c. Strings that contain zero or more uppercase letter character, followed by a digit character.

d. Strings that contain a `0` as their last character.

e. Strings containing a period character.

f.
```
4
6
nil
nil
7
1
```

g.
```
c
b
a
a
b
c
```

h.
```
6
nil
7
6
```

i. `f`

j. An array of strings where each string is a line from the file `filename`.

k. A string for the first command–line argument

l.
```
def m()
  2.times() { yield }
end

m() { puts("Running")}  # prints "Running Running"
```

4.
```
# version that reads entire file into an array
file = File.new(ARGV[0], "r")
lines = file.readlines()
lines.each{ |line|
  # need to get rid of the newline, otherwise it would be a character
  # that doesn't match the r.e. below
  line.chomp!()
  if (line !~ /[^A-Za-z0-9\_]+/) then
    puts(line)
  end
}
```