

Regex Cont

***don't forget to fill out the conflict form if you have a conflict

***Lots of regex tips here: <https://cs1110.cs.virginia.edu/re-tips.html>

One more practice of the day (<http://www.cs.virginia.edu/~up3f/cs1110/practice-of-the-day/>):
(19)

```
# Answer the following questions
# (review basic concepts)

# 1. What would be the value of the variable nums after
#     the execution of the following code?

nums = [1, 2, 3, 4]
nums[3] = 10
# index 3 is the last one
# now nums = [1, 2, 3, 10]
print(nums)
# prints [1, 2, 3, 10]

# 2. What is the value of val in the following expression
#     if x = 4, y = 3, and z = 6

x = 4
y = 3
z = 6
val = x < y or z > x
# x < y is False
# z > x is True
# False or True is True
print(val)
# print True

# 3. True/False: If a whole paragraph is included in a single string,
#     the split() method can be used to obtain a list of the words
#     included in the paragraph
# True -- as long as every word is separated by a space. You might also want to
# strip it first

# 4. When executed, what is output by the following code fragment?

def func1(x, y):
    x[0] = 2
    y = "2"
x = [1]
y = "1"
```

```

func1(x, y)
# func1([1], "1")
# change [1] to [2]
# this happens globally
# change "1" to "2"
# this happens locally
print(x, y)
# print [2], "1"

# 5. What is the output of the following code?

greeting = "Hello, World!"
sub = greeting[:5] + greeting[7:len(greeting)-1]
# greeting[:5] is "Hello"
# greeting[7:len(greeting)-1] is "World"
# concatenated without a space
print(sub)
# print "HelloWorld"

# Final list of things we printed to check against:
# [1, 2, 3, 10]
# True
# [2] 1
# HelloWorld

# if you run it, you can see that this is what gets printed
# if anything doesn't make sense, watch it in Visualize Python

```

We can put repetition in regex statements inside curly brackets immediately following the character we expect to recur

If the number of occurrences could be within a range, {first_possibility, last_possibility}


```

import re                                     at least      at most

def find_all_phone_numbers(filename, regex):
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.search(line)
        if obj != None:
            print(obj, obj.group(), obj.start(), obj.end())
    infile.close()

regex = re.compile(r"[0-9]?[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]")
# this is a very simple regular expression, describing a sequence of a pattern
# it's also very long and repetitive--we can improve it
# let's use {n} for our repetitions
new_regex = re.compile(r"[0-9]?[0-9]{3}-[0-9]{4}")
# let's improve it even more
# put the ambiguity in the number of repetitions, rather than a single ambiguous
# occurrence followed by three guaranteed ones
newest_regex = re.compile(r"[0-9]{3,4}-[0-9]{4}")

```

 exactly 4 times

```
find_all_phone_numbers("simpsons_phone_book", newest_regex)
```

One more useful function distinction: **search()** vs **match()**:

```
import re

def search_phone_numbers(filename, regex):
    """
    returns the first occurrence of the pattern in each line
    """
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.search(line)
        if obj != None:
            print(obj, obj.group(), obj.start(), obj.end())
    infile.close()

def match_phone_numbers(filename, regex):
    """
    returns only if the line begins with the regular expression
    """
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.match(line)
        if obj != None:
            print(obj, obj.group(), obj.start(), obj.end())
    infile.close()

newest_regex = re.compile(r"[0-9]{3,4}-[0-9]{4}")
search_phone_numbers("simpsons_phone_book", newest_regex)
match_phone_numbers("simpsons_phone_book", newest_regex)
```

Run this file to see the difference (don't forget to download the phonebook)

An example of a regex including * (0 to many):

```
import re

def find_all_phone_numbers(filename, regex):
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.search(line)
        if obj != None:
            print(obj, obj.group(), obj.start(), obj.end())
    infile.close()

# a regex for anyone whose first name starts with "J" and last name is "Neu"
regex = re.compile(r"J.*Neu")
find_all_phone_numbers("simpsons_phone_book", regex)

# if we switch it to match, we only get names whose lines start with J
```

Unlike search and match, findall returns a single list of each object as a string, but not the extra info:

```
import re

def find_all_phone_numbers(filename, regex):
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.findall(line)
        for obj in objs:
            print(obj)
    infile.close()

# a regex for anyone whose first name starts with "J" and last name is "Neu"
regex = re.compile(r"J.*Neu")
find_all_phone_numbers("simpsons_phone_book", regex)
```

Finditer() is similar to findall(), but returns all of the info as a list of lists:

```
def finditer(filename, regex):
    """
    returns any time regex appears regardless of environment
    """
    infile = open(filename, 'r')
    for line in infile:
        obj = regex.finditer(line)
        for obj in objs:
            print(obj, obj.group(), obj.start(), obj.end())
    infile.close()
```

Back to regular expression syntax; a new way to write a regex, and grouping:

```
import re
regex = re.compile(r"[0-9]{3,4}-[0-9]{4}")
# [0-9] could also be written as /d for 'digit'
new_regex = re.compile(r"(\d{3,4})-(\d{4})")
# using parentheses defines groups, which are indexed and can later be called
# they start at 1, rather than 0
# so here, if we find a matched object obj
# the part that matches \d{3,4} is obj.group(1)
# and the part that matches \d{4} is obj.group(2)
```

Useful if we want to print only part of the match, or to change to formatting of the data when printing it

In terms of phone numbers, we don't have to write "555-5555"

Could write "555 5555" or "5555 555" or "5555-555" or a lot of things

Because .findall() returns strings, we can't use obj.group() with it