Assignment 1

Problem 1

(a)

The IP address consists of four numbers separated by a dot. For each number, there could be one to three digits.

Regular expression:

[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}

(b)

For each of the four numbers, they should be in the range of 0 to 255. The regex for each number can be created using the same pattern.

1. I consider numbers in the format of 25X, then X should be in [0-5].

2. I consider numbers greater or equal to 200 but less than 250, then it should be 2[0-4][0-9].

3. I consider numbers greater or equal to 100 but less than 200, then it should be 1[0-9][0-9].

4. I consider numbers greater or equal to 0 but less than 100, then it should be [1-9]?[0-9].

5. Repeat the pattern appended with a literal dot for three times, then add the pattern itself one more time.

Regular expression:

((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])

Problem 2

(a)

I assume the year of dates are from 1000 to 2999, which could be modified depending on special needs. The month of dates should be considered in two types. January to September start with zero, while others are from 10 to 12. Similarly, there are three cases for the day of the dates: 01 to 09, 10 to 29, 30 to 31. I do not consider a stricter restriction on the corresponding semantic check for month and its number of days.

Regular expression:

([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))

(b)

The example does not show the second part of the time. This is a regex without matching second.

([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))T([0-1][0-9]|[2][0-3]):([0-5][0-9])

If second should be considered, then this is the modified regex.

([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))T([0-1][0-9]|[2][0-3]):([0-5][0-9]):([0-5][0-9])

Problem 3

(a)

I assume there may or may not be leading zeros in both days and months. I also assume the year of dates are from 1000 to 2999, which could be modified depending on special needs.

Regular expression:

(0[1-9]|[1-9]|1[0-2])\/(0[1-9]|[1-9]|[12]\d|3[01])\/[12]\d{3}

(b)

I assume there may or may not be leading zeros in both days and months. I also assume the year of dates are from 1000 to 2999, which could be modified depending on special needs.

Regular expression:

(0[1-9]|[1-9]|[12]\d|3[01])\.(0[1-9]|[1-9]|1[0-2])\.[12]\d{3}

(c)

I assume there may or may not be leading zeros in days. I also assume the year of dates are from 1000 to 2999, which could be modified depending on special needs. Especially, there should be a white space between the month and day, and between the comma and year.

Besides, I add letter "i" to the matching principle to match ignore cases in regexr.com. In python, we can use re.IGNORECASE as the flag for case insensitive matching.

Regular expression:

/(January|February|March|April|May|June|July|August|September|October|November|December) (0[1-9]|[1-9]|[12]\d|3[01]), [12]\d{3}/ig

Problem 4

General Assumptions:

- The email address should start with an alphanumeric character.
- As the email address standard states, for the local part before @, it can consist of a series of letters, digits and certain symbols, including one or more dots. However, **dots may not appear consecutively or at the start or end of the email address.**
- The special characters can appear multiple times in the local part. It is reasonable based on general rules of email address – RFC5322.
- For the local part, the four special characters (-_+%) can appear just before the @ sign.
- For the first section of the domain part, alphanumeric and dash characters can appear multiple times.
- For the domain part after the first dot, only alphanumeric characters can appear as the penultimate section with a dot (except for dash).
- For the last section of the domain part, only more than two letters can appear.
- I did not add the start or end anchor into the regex. The implement method needs to be considered in real Python program.

1. The first solution (may use re.match() in Python to find):

/[A-Za-z0-9]([-._+%]?[A-Za-z0-9])*[-_+%]?@[A-Za-z0-9-]+\.([A-Za-z0-9]+\.)*[a-zA-Z]{2,}/ig

For the local part, it assumes any special characters (-._+%) cannot appear consecutively. Also, any different special characters cannot appear adjacently. This restriction is strict in real life.

2. The second solution (may use re.match() in Python to find):

/[A-Za-z0-9](\.?[A-Za-z0-9-_+%])*@[A-Za-z0-9-]+\.([A-Za-z0-9]+\.)*[a-zA-Z]{2,}/ig

For the local part, it assumes characters "-", "_", "+", "%" can appear more than once, even consecutively, which is less strict in real life.

Problem 5

Please refer to Assignment1_Problem5_HuiLyu.py file.

(I assume the year of dates are from 1000 to 2999, which could be modified depending on special needs.)

I run the program in Spyder (Python 3.5) in IPython console.

The code is appended below.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Feb 23 16:46:17 2017

@author: HuiLyu
"""

from datetime import datetime
from datetime import date
import re

# Three types of non-standard dates:
# (0[1-9]|[1-9]|1[0-2])\/(0[1-9]|[1-9]|[12]\d|3[01])\/[12]\d{3}
# (0[1-9]|[1-9]|[12]\d|3[01])\.(0[1-9]|[1-9]|1[0-2])\.[12]\d{3}
# (January|February|March|April|May|June|July|August|September|October|November|December) (0[1-9]|[1-9]|[12]\d|3[01]), [12]\d{3}
```

```
def reformat1(string):
    match = re.search(r'(0[1-9]|[1-9]|1[0-2])\/(0[1-9]|[1-9]|[12]\d|3[01])\/([12]\d{3})', string)
    if match != None:
        month = match.group(1)
        day = match.group(2)
        year = match.group(3)
        standard_date = date(int(year), int(month), int(day)).isoformat()
        return standard_date
    else:
        return -1

def reformat2(string):
    match = re.search(r'(0[1-9]|[1-9]|[12]\d|3[01])\.(0[1-9]|[1-9]|1[0-2])\.([12]\d{3})', string)
    if match != None:
        day = match.group(1)
        month = match.group(2)
        year = match.group(3)
        standard_date = date(int(year), int(month), int(day)).isoformat()
        return standard_date
    else:
        return -1

def reformat3(string):
    try:
        standard_date = datetime.strptime(string, '%B %d, %Y')
        return standard_date.date()
    except ValueError:
        return -1
```

```
def standardization():
    string = input("Please input the date:")
    result1 = reformat1(string)
    if result1 != -1:
        print("The standardized date is {}".format(result1))
    else:
        result2 = reformat2(string)
        if result2 != -1:
            print("The standardized date is {}".format(result2))
        else:
            result3 = reformat3(string)
            if result3 != -1:
                print("The standardized date is {}".format(result3))
            else:
                print("Failed to reformat. Please try again.")

standardization()
```