1. (8 points) **Input/Output**. Provide the *exact output* of the program shown below.

```
print 'A'
for i in range(5,10):
        print i

print 'B'
for i in range(4):
        print 'i:',i
        for j in range(i):
                print i*j

print 'C'
x, y, z = 0, 1, 2
for i in range(2):
        x, y, z = y, z, x
print x, y, z

print 'D'
x, y, z = 0, 1, 2
for i in range(99):
        x, y, z = y, z, x
print x, y, z
```

Standard output (use two columns as needed):

| | |
|---|---|
| A | 2 |
| 5 | i: 3 |
| 6 | 0 |
| 7 | 3 |
| 8 | 6 |
| 9 | C |
| B | 2 0 1 |
| i: 0 | D |
| i: 1 | 0 1 2 |
| 0 | |
| i: 2 | |
| 0 | |

2. (8 points) **Input/Output**. Provide the *exact output* of the program shown below.

```
print 'a -----'

def f(x):
        return x*x > x

for a in filter( f, [x * 0.5 for x in range(-3,3)] ):
        print a

print 'b -----'

L = [
        lambda x: -x,
        lambda x: x + x,
        lambda y: y**2,
]

for z in L:
        L0 = map(z, range(5))
        print L0[0]
        print L0[-1]
```

Standard output:

a -----

-1.5

-1.0

-0.5

b -----

0

-4

0

8

0

16

3. (8 points) For each of the Python regular expressions below, list all strings which:

1. match the regular expression, using `re.search`,
2. are of length 5 or less and
3. contain only the characters '0' and '1'.

**List the strings in length order, starting with the shortest strings.**

| *Regular expression* | *Matching strings* |
|---|---|
| `'^(01*)$'` | `'0', '01', '011', '0111', '01111'` |
| `'^(0\|1*)$'` | '', '0', '1', '11', '111', '1111', '11111' |
| `'^(00\|11*)$'` | '1', '00', '11', '111', '1111', '11111' |
| `'^((00\|11)*)$'` | '', '00', '11', '0000', '0011', '1100', '1111' |
| `'^((11\|000)*)$'` | '', '11', '000', '1111', '00011', '11000' |

4. (8 marks) Input/Output. Provide the *exact output* of the Python code shown below:

```
import re

L = [
        [ '(aa|bb)',     'baab' ],
        [ '^(aa|bb)',    'baab' ],

        [ '(aa)(b)',     'bab' ],
        [ '(aa)?(b)',    'bab' ],

        [ '(a+)(a+)',    'baaab' ],
        [ '(a+?)(a+)',   'baaab' ], ]
def list_match(L):
        for x in L:
                m = re.search(x[0], x[1])
                if m:
                        print 'Y'
                        for i in range(m.lastindex):
                                print m.group(i+1)
                else:
                        print 'N'
list_match(L)
```

Standard output:

Y

aa

N

N

Y

None

b

Y

aa

a

Y

a

aa

5. (8 points) **Input/Output.**

(a) Provide the *exact output* of the program shown below.

```
def f(L):
        if L[1] == [ ]:
                return L[0]
        else:
                return f(L[1])

print f([ 1, [], [] ])

print f([ 3, [1,[],[]], [5,[],[]] ])

print f([ 5, [4, [3,[],[]], []], [7,[],[]] ])
```

Standard output:

1

1

3

6. (10 points) **Implementation to specification.** Provide a correct implementation for the `address` function specified below. Be sure that your code is as short and simple as possible.

```
# Purpose:
#       Extract the city, province and postal code from address A
#       if A is legal return a list [c,p,pc] where
#               c: city (any proper noun acceptable)
#               pc: postal code (6 characters, alternating letter/digit)
#               p: province (2 upper case characters)
#       else
#               return None
# Preconditions:
#       none
# Examples:
#       address('Victoria, BC V8W 3P6') returns ['Victoria','BC','V8W 3P6']
#       address('Victoria,  BC  V8W 3P6') returns ['Victoria','BC','V8W
3P6']
#       address('Victoria,BC V8W3P6') returns ['Victoria','BC','V8W 3P6']
#       address('victoria, BC V8W 3P6') returns None
#       address('Victoria, Bc V8W 3P6') returns None
#       address('Victoria, BCV8W 3P6') returns None
def address(A):
```

upper case

```
    city = '([A-Z][A-Za-z]+)'
    province = '([A-Z][A-Z])'
    postal_code = '([A-Z][0-9][A-Z] *[0-9][A-Z][0-9])'
    m = re.search(city + ',*' + province + ' +' + postal_code,A)
    if m:
            return [m.group(1),m.group(2),m.group(3)]
    else:
            return None
```

Python

**BONUS** (5 points) **Implementation to specification.** Provide a correct ~~C~~ implementation for the phone function specified below. Be sure that your code is as short and simple as possible.

```
import re

# Purpose
#       Extract a telephone number from s.
#       A legal number can be of one of three forms:
#               9-1-604-123-4567
#               9-250-123-4567
#               4567
#       In the first 2 cases, the '-' characters are optional.
#       If the line contains a legal number and nothing else
#               return a list with the values of the 5 fields of the number
#       else
#               return None
# Preconditions
#       s is a string
# Examples
#       phone('9-1-604-123-4567') returns
#               ['9', '1', '604', '123', '4567']
#       phone('916041234567') returns
#               ['9', '1', '604', '123', '4567']
#       phone('9-250-123-4567') returns
#               ['9', None, '250', '123', '4567']
#       phone('92501234567') returns
#               ['9', None, '250', '123', '4567']
#       phone('4567') returns
#               [None, None, None, None, '4567']
#       phone('9-9-604-123-4567') returns None
def phone(s):


  # outside call
  m = re.search('^(9)-?(1)?-?([0-9]{3})-?([0-9]{3})-?([0-9]{4})$',s)
  if m:
          return [m.group(1),m.group(2),m.group(3),m.group(4),m.group(5)]

  # internal call
  m = re.search('^([0-9]{4})$',s)
  if m:
          return [None,None,None,None,m.group(1)]

  return None
```