Name: ________________________________ Date: ___________________

Pledge: ____

Closed book: no textbook, no electronic devices, one sheet of paper with notes. Read each question carefully before answering! Write your answers on the test paper and turn in your notes.

Question 1 (5 points)
Consider the following code:

```
>>> L = [ 'This', 'is', 'a', 'list' ]
>>> M = [ 'not', 'possibly', 'definitely' ]
>>> N = L[ :2 ] + [ M[ 1 ] ] + L[ 2: ]
```

What is the value of N after these statements have executed?
```
[ 'This', 'is', 'possibly', 'a', 'list' ]
```

Assess: [state]
Rubric: (1 point for list brackets, 1 point for strings in quotes, 2 points for correct values, 1 point for no additional values)

Question 2 (5 points)
Consider the following code:

```
>>> L = [ 'www', 'stevens', 'edu', 'sit', 'registrar' ]
>>> M = range( len(L) )
>>> print L[ M[ 3 ] ]
```

What is printed on the screen after these statements have executed?
```
sit or 'sit'
```

Assess: [state]
Rubric: (3 points for sit or 'sit', 2 points for no additional values)

Question 3 (5 points)
Consider the following code:

```
>>> L = ['www', 'stevens', 'edu']
>>> M = ['www', 'nyu', 'edu'] + L
>>> N = L [ :1 ] + M[ 4: ]
```

What is the value of N after these statements have executed?
```
['www', 'stevens', 'edu']
```

Assess: none
Rubric: (1 point for list brackets, 1 point for strings in quotes, 2 points for correct values, 1 point for no additional values)

Question 4 (20 points)
Write a trace of the call `mystery(3, 2, 6)` for the function definition below. That is, starting with `mystery(3, 2, 6)`, show each call that is made, with the argument values, in a downward growing stack. In addition, draw a reverse arrow labeled with the value returned, for each call.

```
def mystery(n, a, d):
    if n == 0:
        return a
    return d + mystery(n - 1, a, d + 1)
```

```
mystery(3, 2, 6)
   ➔ 6 + mystery(2, 2, 7) = 6 + 17 = 23
mystery(2, 2, 7)
   ➔ 7 + mystery(1, 2, 8) = 7 + 10 = 17
mystery(1, 2, 8)
   ➔ 8 + mystery(0, 2, 9) = 8 + 2 = 10
mystery(0, 2, 9)
   ➔ 2
```

Assess: [execution]
Rubric: (5 points for first function call, 5 points for returning 2 in base case, 5 points for going up stack, 5 points for correct result)

Question 5 (15 points)
Consider the following code. What does it print? (Hint: you might want to do a trace for yourself.)

```
def confuse(x, lst):
    if lst == []:
        return False
    if x == lst[-1]:
        return True
    return confuse(x, lst[:-1])

L = range(1, 6)
M = map(lambda x : 2*x + 1, L)
print M
print confuse(4, M)
```

```
[3, 5, 7, 9, 11]
False
```

Assess: [execution]
Rubric: (3 points for M being a list, 2 points for M containing 5 values, 5 points for M having the right numbers, 5 points for False in the second line)

Question 6 (25 points)
Implement the following function using recursion:

```
def prefixes(L):
    '''Assume L is a list.
    Return the prefixes of L, in increasing order by length.
    For example, prefixes([5, 1, 42, 3]) should return
    [[], [5], [5, 1], [5, 1, 42], [5, 1, 42, 3]]

    Hint: As the example shows, we consider [] to be a prefix of any list,
    so prefixes([]) should return [[]].'''

    if L == []:
        return [[]]
    return prefixes(L[:-1]) + [L]
```

Assess: [coding]
Rubric:
(5 points for syntax,
 2 points for correct condition in if statement,
 3 points for correct return value in base case,
 5 points for design using recursion, i.e. recursive call to prefixes with any list slice,
 5 points for concatenating [L],
 5 points for recursive call argument being L [:-1])

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 7 (20 points)
' Implement this function using recursion.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
def keep_excitement_rec(lst):
    '''Assume lst is a list of non-empty strings.
    Return a list of strings with only those words ending in an exclamation
    mark.'''
    if lst == []:
        return []
    if lst[0][-1] == '!':
        return [lst[0]] + keep_excitement_rec(lst[1:])
    return keep_excitement_rec(lst[1:])
```

Rubric:
(3 points for correct if statement in base case,
 2 points for correct return statement in base case,
 3 points for correct if statement leading to recursive call,
 2 points for returning `[lst[0]]`,)
 5 points for each `keep_excitement_rec(lst[1:])`)

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' Question 8 (10 points)
' Implement this function using the Python's built-in 'filter' and 'lambda'.
' DO NOT USE recursion.
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
def keep_excitement_fil(lst):
    '''Assume lst is a list of non-empty strings.
    Return a list of strings with only those words ending in an exclamation
    mark.'''
    return filter(lambda x: x[-1] == '!', lst)
```

Rubric:
(5 points for correct use of lambda,
 5 points for correct use of filter)