A. Short Answer:

(1) What are the values of the following expressions?

\[
\begin{align*}
&[5,7,1,9,3,2,8][\text{len}(["0123")]) \quad \underline{\text{_________}} \\
&[6,9,2,5,4,3,1,8][(2+1):(10-13)] \quad \underline{\text{_________}} \\
&"\text{ study!\t}.\text{title()}.\text{strip()[:3]} \text{ in } "\text{study!}" \quad \underline{\text{_________}} \\
&[[1,0,1],[0,1,0],[2]][1][0] \quad \underline{\text{_________}} \\
&[[1,3,1,2],[2,4,1],[3,1,1]][[1,2,3][0]][[-1,-2,-3][-2]] \quad \underline{\text{_________}}
\end{align*}
\]

(2) Explain what the is operator does.

The is operator does identity comparison. That is, it returns true if two variables refer to the SAME list (/object/memory location). (Ie, if they refer to an alias).

(3) Define the term mutability as it relates to data structures.

A mutable data structure is one that can change without creating a new copy. E.g. Lists are mutable, strings are immutable.

(4) What is the purpose of the call stack?

The call stack is how Python keeps track of active function calls, and how it remembers where to return to when a function is done.
B. Comprehension:

(1) What does the code to the right print? Show your work.

Steps:
[3,4,3,0,1,2]
[3,4,3,0,1,2,3]
[3,4,3,2,1,2,3]
[4,2,1,2,3]
[4,2,2,1,2,3]  #<-output

Note: L and R are aliases for the same list, so only one list in the steps.

(2) What does the code to the right print?

D={'a':'b','c':'d','b':'e','e':'f'}
L = []
for k in D:
    if D[k] in D:
        L.append(k)
    else:
        L.append(D[k])
for i in range(len(L)-1):
    D[L[i]] = L[i+1]
print(D)

(3) What does the code to the right print?

grid=[]
for i in range(3):
    grid.append([])
    grid[i].append(i)
    for j in range(1,3):
        grid[i].append(grid[i][j-1]+j)
for x in grid:
    print(x)
What does the code to the right print?

```
[7, 2, 4]
[14, 2, 0]
[6, 0, 1]
[4, 4, 6]
```

C. Debugging:
The following code should implement the insertionSort algorithm. Circle any bugs in the provided code, and write out the corrected code in the space provided.

```
def insertionSort(L):
    L = L[:]
    out = []
    while len(L) > 0:
        item = L.pop(0)
        out.append(item)
        i = len(out)-1
        while out[i] < out[i-1]:
            temp = out[i-1]
            out[i-1] = out[i]
            out[i] = temp
        i -= 1
        if i == 0:
            break
    return out
L = [[6,8,2],[7,4,2],[2,1,0],[3,4,7]]
for i in range(0,len(L)):
    for j in range(0,len(L[i])):
        if L[i][j]%3==0:
            L[i][j] += 1
        elif i>0 and L[i][j] > L[i-1][j]:
            L[i][j] = L[i-1][j]
        elif i<len(L)-1 and L[i+1][j]<3:
            L[i][j] *= L[i+1][j]
        else:
            L[i][j] //= len(L[i])
for e in L:
    print(e)
```
D. Programming:

(1) Write a function called `capEnds()` that takes a string as argument and returns a string with the last letter of every word uppercased. You may not use the built-in `string.title()` method.

E.g. `capEnds("these are some words")` → "thesE arE somE wordS"

```python
def capEnds(s):
    out = ''
    for i in range(len(s)):
        if i == len(s)-1 or s[i+1]==' ':
            out+=s[i].upper()
        else:
            out+=s[i]
    return out
```

(2) Write a function called `userWords()` that repeatedly asks the user to enter words until they enter the word "stop!". Every word the user enters should be stored in a nested list arranged by word-length such that all the 3-letter words are stored in a list at index 3 (for example). Return the nested list when complete. You may assume no word is larger than 9 letters.

```python
def userWords():
    word = ""
    wordList = [[],[],[],[],[],[],[],[],[],[]]
    while word!="stop!":
        word = input("Enter a word (stop!) to stop: ")
        wordList[len(word)].append(word)
    return wordlist
```
(3) Write a function called `qWords()` that takes a 2D list of words as argument (e.g., such as the one returned from the previous problem), and returns a 1D list of all the words that start with the letter 'q' from the input list.

```python
def qWords(L):
    out = []
    for lis in L:
        for e in lis:
            if e[0].lower() == 'q':  # .lower() not mandatory
                out.append(e)
    return out
```

(4) Write a **recursive** function called `reverse()` that takes a list as argument and returns the list in reverse order. E.g. `reverse([5,3,2,4,1]) → [1,2,3,4,5]`

```python
def reverse(lis):
    if len(lis) == 0:
        return []
    else:
        return reverse(lis[1:]) + [lis[0]]
```
(5) Write a function called `copyEvenLines()` that takes two filenames as input and copies every other line from the first file into the second. That is, counting from 0, the 0th line, the 2nd line, the 4th etc... You may not assume that either file exists. Return True if successful, else False.

```python
def copyEvenLines(inFile, outFile):
    try:
        f = open(inFile, 'r')
        lines = f.readlines()
        f.close()

        f = open(outFile, 'w')
        for i in range(0, len(lines), 2):
            f.write(lines[i])
        f.close()
        return True
    except:
        return False
```

(6) Write a function called `mutateList()` that takes a nested list of integers as argument and performs the following mutations: any even integers get divided by 2, any odd integers less than 10 get multiplied by 10, and any row (inner list) with less than 5 elements should be padded with zeroes to exactly 5 elements.

E.g. `mutateList([[8,5,7,2],[13,12,9,17,5],[10,20]])` → `[[4,50,70,1,0],[13,6,90,17,50],[5,10,0,0,0]]`

```python
def mutateList(L):
    for i in range(0, len(L)):
        for j in range(0, len(L[i])):
            if L[i][j] % 2 == 0:
                L[i][j] /= 2
            elif L[i][j] < 10:
                L[i][j] *= 10
            if len(L[i]) < 5:
                for j in range(len(L[i]), 5):
                    L[i].append(0)