Name:_____

1. Consider the following program to print even numbers up to some user specified amount.

```
stopValue = input("Enter a stopping value: ")
counter = 0
while counter != stopValue:
    print counter
    counter = counter + 2
print "Done"
```

a) In the box, show a few lines of the programs output assuming the user enters 8 for a stopping value.

b) What would happen if the user enters 8 for a stopping value?

c) What modification would improve this program?

2. An *infinite loop* is one that would loop forever. (FYI, at a Python shell ctrl-c (^c) can be used to kill the running program.) Most infinite loops are caused by programmer error, but sometimes they are intentional. The following code uses an infinite loop and a *break* statement that immediately causes control to exit the loop.

```
total = 0
counter = 0
while True:     # an infinite loop
    score = input("Enter a score (or -99 to exit): ")
    if score < 0:
        break
    total += score
    counter += 1
print "Average is", float(total)/counter
```

Draw a flow chart for this code.

Name:_____

3. A *continue* statement also alters execution of a loop body. When a *continue* statement is encountered, the next iteration of the loop (including the `while` condition check) is immediately executed. Armed with this limited knowledge, trace the following program to determine its output.

```
number = -4
while number < 3:
    print number
    number = number + 1
    if number <= 0:
        continue
    print "Positive: ", number

print "done"
```

4. Most languages have a `for` loop to aid in writing "count-controlled" loops where the number of iterations is known in advance. In Python, the `for` loop iterates once for each item in some sequence type (i.e, a list, tuple, string). A simple example printing each number in the list [1, 3, 9, 7] is:

```
for value in [1, 3, 9, 7]:
    print value
```

```
1
3
9
7
```

a) What do you guess would be printed by the following program?

```
for character in 'house':
    print character
print 'done'
```

5. Often the `for` loop iterates over a list generated by the built-in `range` function which has the syntax of: `range([start,] end, [, step])`, where [ ] are used to denote optional parameters. Some examples:

- `range(5)` generates the list `[0, 1, 2, 3, 4]`
- `range(2,7)` generates the list `[2, 3, 4, 5, 6]`
- `range(10,2)` generates the list `[]`
- `range(10,2,-1)` generates the list `[10, 9, 8, 7, 6, 5, 4, 3]`

Write range function calls to generate the following lists:

a) odd numbers from 3 to 13 (inclusive)

b) multiples of 10 from 0 to 100 (inclusive)

c) the numbers from 0 to 100 (inclusive)

6. Rewrite the following program using a `for` loop.

```
numberOfScores = input("Enter # of scores: ")
counter = 0
total = 0
while counter < numberOfScores:
    score = input("Enter a score: ")
    total = total + score
    counter = counter + 1

print "Average is ", float(total)/counter
```