Chapter 5 Loops

1. `count < 100 is always True at Point A. count < 100 is always False at Point C. count < 100 is sometimes True or sometimes False at Point B.`

2.
```
It would be wrong if it is initialized to a value between 0 and 100, because
it could be the number you attempt to guess.
```
When the initial guess value and random number are equal, the loop will never be executed.

3. (a) Infinite number of times.
   (b) Infinite number of times.

   (c) The loop body is executed nine times. The printout is `2, 4, 6, 8` on separate lines.

4. (a) and (b) are infinite loops, (c) has an indentation error.

5.
   max is 5
   number 0

6.
   sum is 14
   count is 4

7.
   Yes. The advantages of `for` loops are simplicity and readability. Compilers can produce more efficient code for the `for` loop than for the corresponding `while` loop.

8. `while` loop:

```
sum = 0
i= 0
while i <= 1000:
    sum += i
    i += 1
```

9. `Can you always convert a while loop into a for loop?`
   Not in Python. For example, you cannot convert the while loop in Listing 5.3, GuessNumber.py, to a for loop.

```
sum = 0
for i in range(1, 10000):
    if sum < 10000:
        sum = sum + i
```

10.

(A)
`n times`

(B)
`n times`

(C)
`n-5 times`

(D)
`The ceiling of (n-5)/3 times`

11.

Tip for tracing programs:
Draw a table to see how variables change in the program. Consider (a) for example.

i	j	output
1	0	0
1	1	
2	0	0
2	1	1
2	2	
3	0	0
3	1	1
3	2	2
3	3	
4	0	0
4	1	1
4	2	2
4	3	3
4	4	

(A).

```
0 0 1 0 1 2 0 1 2 3
```

(B).

```
****
****
```

```
2 ****
3 2 ****
4 3 2 ****
```

(C).

```
1xxx2xxx4xxx8xxx16xxx
1xxx2xxx4xxx8xxx
 1xxx2xxx4xxx
 1xxx2xxx
1xxx
```

(D).

```
1G
1G3G
1G3G5G
1G3G5G7G
1G3G5G7G9G
```

12. No. Try n1 = 3 and n2 =3.

13. The keyword `break` is used to exit the current loop. The program in (A) will terminate. The output is *Balance is 1*.

The keyword `continue` causes the rest of the loop body to be skipped for the current iteration. The `while` loop will not terminate in (B).

14. If a continue statement is executed inside a for loop, the rest of the iteration is skipped, then the action-after-each-iteration is performed and the loop-continuation-condition is checked. If a continue statement is executed inside a while loop, the rest of the iteration is skipped, then the loop-continuation-condition is checked.

Here is the fix:

```
i = 0

while i < 4:
    if i % 3 == 0:
        i += 1
        continue
    sum += i
    i += 1
```

15.

TestBreak.py

```
sum = 0
number = 0

while number < 20 and sum < 100:
    number += 1
    sum += number

print("The number is " + str(number))
print("The sum is " + str(sum))
```

TestContinue.py

```
sum = 0
number = 0

while (number < 20):
    number += 1
    if (number != 10 and number != 11):
        sum += number

print("The sum is " + str(sum))
```

16.

(A)

`print(j)`

```
1
2
1
2
2
3
```

(B)

for j in range (1,4):

```
1
2
1
2
2
3
```