Chapter 5

Miscellaneous Topics I

This chapter consists of a several common techniques and some other useful information.

5.1 Counting

Very often we want our programs to count how many times something happens. For instance, a video game may need to keep track of how many turns a player has used, or a math program may want to count how many numbers have a special property. The key to counting is to use a variable to keep the count.

Example 1 This program gets 10 numbers from the user and counts how many of those numbers are greater than 10.

```
count = 0
for i in range(10):
    num = eval(input('Enter a number: '))
    if num>10:
        count=count+1
print('There are', count, 'numbers greater than 10.')
```

Think of the `count` variable as if we are keeping a tally on a piece of paper. Every time we get a number larger than 10, we add 1 to our tally. In the program, this is accomplished by the line `count=count+1`. The first line of the program, `count=0`, is important. Without it, the Python interpreter would get to the `count=count+1` line and spit out an error saying something about not knowing what `count` is. This is because the first time the program gets to this line, it tries to do what it says: take the old value of `count`, add 1 to it, and store the result in `count`. But the first time the program gets there, there is no old value of `count` to use, so the Python interpreter doesn't know what to do. To avoid the error, we need to define `count`, and that is what the first line does. We set it to 0 to indicate that at the start of the program no numbers greater than 10 have been found.

Counting is an extremely common thing. The two things involved are:

1. `count=0` — Start the count at 0.
2. `count=count+1` — Increase the count by 1.

Example 2 This modification of the previous example counts how many of the numbers the user enters are greater than 10 and also how many are equal to 0. To count two things we use two count variables.

```
count1 = 0
count2 = 0
for i in range(10):
    num = eval(input('Enter a number: '))
    if num>10:
        count1=count1+1
    if num==0:
        count2=count2+1
print('There are', count1, 'numbers greater than 10.')
print('There are', count2, 'zeroes.')
```

Example 3 Next we have a slightly trickier example. This program counts how many of the squares from 1^2 to 100^2 end in a 4.

```
count = 0
for i in range(1,101):
    if (i**2)%10==4:
        count = count + 1
print(count)
```

A few notes here: First, because of the aforementioned quirk of the `range` function, we need to use `range(1,101)` to loop through the numbers 1 through 100. The looping variable `i` takes on those values, so the squares from 1^2 to 100^2 are represented by `i**2`. Next, to check if a number ends in 4, a nice mathematical trick is to check if it leaves a remainder of 4 when divided by 10. The modulo operator, `%`, is used to get the remainder.

5.2 Summing

Closely related to counting is summing, where we want to add up a bunch of numbers.

Example 1 This program will add up the numbers from 1 to 100. The way this works is that each time we encounter a new number, we add it to our running total, s.

```
s = 0
for i in range(1,101):
    s = s + i
print('The sum is', s)
```

Example 2 This program that will ask the user for 10 numbers and then computes their average.

```
s = 0
for i in range(10):
    num = eval(input('Enter a number: '))
    s = s + num
print('The average is', s/10)
```

Example 3 A common use for summing is keeping score in a game. Near the beginning of the game we would set the score variable equal to 0. Then when we want to add to the score we would do something like below:

```
score = score + 10
```

5.3 Swapping

Quite often we will want to swap the values of two variables, x and y. It would be tempting to try the following:

```
x = y
y = x
```

But this will not work. Suppose x is 3 and y is 5. The first line will set x to 5, which is good, but then the second line will set y to 5 also because x is now 5. The trick is to use a third variable to save the value of x:

```
hold = x
x = y
y = hold
```

In many programming languages, this is the usual way to swap variables. Python, however, provides a nice shortcut:

```
x,y = y,x
```

We will learn later exactly why this works. For now, feel free to use whichever method you prefer. The latter method, however, has the advantage of being shorter and easier to understand.

5.4 Flag variables

A flag variable can be used to let one part of your program know when something happens in another part of the program. Here is an example that determines if a number is prime.

```
num = eval(input('Enter number: '))

flag = 0
for i in range(2,num):
    if num%i==0:
        flag = 1

if flag==1:
    print('Not prime')
else:
    print('Prime')
```

Recall that a number is prime if it has no divisors other than 1 and itself. The way the program above works is `flag` starts off at 0. We then loop from 2 to `num-1`. If one of those values turns out to be a divisor, then `flag` gets set to 1. Once the loop is finished, we check to see if the flag got set or not. If it did, we know there was a divisor, and `num` isn't prime. Otherwise, the number must be prime.

5.5 Maxes and mins

A common programming task is to find the largest or smallest value in a series of values. Here is an example where we ask the user to enter ten positive numbers and then we print the largest one.

```
largest = eval(input('Enter a positive number: '))
for i in range(9):
    num = eval(input('Enter a positive number: '))
    if num>largest:
        largest=num
print('Largest number:', largest)
```

The key here is the variable `largest` that keeps track of the largest number found so far. We start by setting it equal to the the user's first number. Then, every time we get a new number from the user, we check to see if the user's number is larger than the current largest value (which is stored in `largest`). If it is, then we set `largest` equal to the user's number.

If, instead, we want the smallest value, the only change necessary is that $>$ becomes $<$, though it would also be good to rename the variable `largest` to `smallest`.

Later on, when we get to lists, we will see a shorter way to find the largest and smallest values, but the technique above is useful to know since you may occasionally run into situations where the list way won't do everything you need it to do.

5.6 Comments

A comment is a message to someone reading your program. Comments are often used to describe what a section of code does or how it works, especially with tricky sections of code. Comments have no effect on your program.

Single-line comments For a single-line comment, use the # character.

```
# a slightly sneaky way to get two values at once
num1, num2 = eval(input('Enter two numbers separated by commas: '))
```

You can also put comments at the end of a line:

```
count = count + 2  # each divisor contributes two the count
```

Multi-line comments For comments that span several lines, you can use triple quotes.

```
""" Program name: Hello world
    Author: Brian Heinold
    Date: 1/9/11 """

print('Hello world')
```

One nice use for the triple quotes is to comment out parts of your code. Often you will want to modify your program but don't want to delete your old code in case your changes don't work. You could comment out the old code so that it is still there if you need it, and it will be ignored when your new program is run. Here is a simple example:

```
"""
print('This line and the next are inside a comment.')
print('These lines will not get executed.')
"""
print('This line is not in a comment and it will be executed.')
```

5.7 Simple debugging

Here are two simple techniques for figuring out why a program is not working:

1. Use the Python shell. After your program has run, you can type in the names of your program's variables to inspect their values and see which ones have the values you expect them to have and which don't. You can also use the Shell to type in small sections of your program and see if they are working.

2. Add print statements to your program. You can add these at any point in your program to see what the values of your variables are. You can also add a print statement to see if a point in your code is even being reached. For instance, if you think you might have an error in

a condition of an if statement, you can put a print statement into the if block to see if the condition is being triggered.

Here is an example from the part of the primes program from earlier in this chapter. We put a print statement into the for loop to see exactly when the flag variable is being set:

```
flag = 0
num = eval(input('Enter number: '))
for i in range(2,num):
    if num%i==0:
        flag = 1
    print(i, flag)
```

3. An empty input statement, like below, can be used to pause your program at a specific point:

```
input()
```

5.8 Example programs

It is a valuable skill is to be able to read code. In this section we will look in depth at some simple programs and try to understand how they work.

Example 1 The following program prints `Hello` a random number of times between 5 and 25.

```
from random import randint

rand_num = randint(5,25)
for i in range(rand_num):
    print('Hello')
```

The first line in the program is the import statement. This just needs to appear once, usually near the beginning of your program. The next line generates a random number between 5 and 25. Then, remember that to repeat something a specified number of times, we use a for loop. To repeat something 50 times, we would use `range(50)` in our for loop. To repeat something 100 times, we would use `range(100)`. To repeat something a random number of times, we can use `range(rand_num)`, where `rand_num` is a variable holding a random number. Although if we want, we can skip the variable and put the `randint` statement directly in the `range` function, as shown below.

```
from random import randint

for i in range(randint(5,25)):
    print('Hello')
```

Example 2 Compare the following two programs.

```
from random import randint          from random import randint

rand_num = randint(1,5)             for i in range(6):
for i in range(6):                      rand_num = randint(1,5)
    print('Hello'*rand_num)             print('Hello'*rand_num)
```

```
Hello Hello                         Hello Hello Hello
Hello Hello                         Hello
Hello Hello                         Hello Hello Hello Hello
Hello Hello                         Hello Hello Hello
Hello Hello                         Hello Hello
Hello Hello                         Hello
```

The only difference between the programs is in the placement of the `rand_num` statement. In the first program, it is located outside of the for loop, and this means that `rand_num` is set once at the beginning of the program and retains that same value for the life of the program. Thus every print statement will print `Hello` the same number of times. In the second program, the `rand_num` statement is within the loop. Right before each print statement, `rand_num` is assigned a new random number, and so the number of times `Hello` is printed will vary from line to line.

Example 3 Let us write a program that generates 10000 random numbers between 1 and 100 and counts how many of them are multiples of 12. Here are the things we will need:

- Because we are using random numbers, the first line of the program should import the `random` module.
- We will require a for loop to run 10000 times.
- Inside the loop, we will need to generate a random number, check to see if it is divisible by 12, and if so, add 1 to the count.
- Since we are counting, we will also need to set the count equal to 0 before we start counting.
- To check divisibility by 12, we use the modulo, `%`, operator.

When we put this all together, we get the following:

```
from random import randint

count = 0
for i in range(10000):
    num = randint(1, 100)
    if num%12==0:
        count=count+1

print('Number of multiples of 12:', count)
```

Indentation matters

A common mistake is incorrect indentation. Suppose we take the above and indent the last line. The program will still run, but it won't run as expected.

```
from random import randint

count = 0
for i in range(10000):
    num = randint(1, 100)
    if num%12==0:
        count=count+1
    print('Number of multiples of 12:', count)
```

When we run it, it outputs a whole bunch of numbers. The reason for this is that by indenting the print statement, we have made it a part of the for loop, so the print statement will be executed 10,000 times.

Suppose we indent the print statement one step further, like below.

```
from random import randint

count = 0
for i in range(10000):
    num = randint(1, 100)
    if num%12==0:
        count=count+1
        print('Number of multiples of 12:', count)
```

Now, not only is it part of the for loop, but it is also part of the if statement. What will happen is every time we find a new multiple of 12, we will print the count. Neither this, nor the previous example, is what we want. We just want to print the count once at the end of the program, so we don't want the print statement indented at all.

5.9 Exercises

1. Write a program that counts how many of the squares of the numbers from 1 to 100 end in a 1.

2. Write a program that counts how many of the squares of the numbers from 1 to 100 end in a 4 and how many end in a 9.

3. Write a program that asks the user to enter a value n, and then computes $(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n})-\ln(n)$. The ln function is `log` in the `math` module.

4. Write a program to compute the sum $1-2+3-4+\cdots+1999-2000$.

5. Write a program that asks the user to enter a number and prints the sum of the divisors of that number. The sum of the divisors of a number is an important function in number theory.

6. A number is called a *perfect number* if it is equal to the sum of all of its divisors, not including the number itself. For instance, 6 is a perfect number because the divisors of 6 are 1, 2, 3, 6 and $6 = 1 + 2 + 3$. As another example, 28 is a perfect number because its divisors are 1, 2, 4, 7, 14, 28 and $28 = 1 + 2 + 4 + 7 + 14$. However, 15 is not a perfect number because its divisors are 1, 3, 5, 15 and $15 \neq 1 + 3 + 5$. Write a program that finds all four of the perfect numbers that are less than 10000.

7. An integer is called *squarefree* if it is not divisible by any perfect squares other than 1. For instance, 42 is squarefree because its divisors are 1, 2, 3, 6, 7, 21, and 42, and none of those numbers (except 1) is a perfect square. On the other hand, 45 is not squarefree because it is divisible by 9, which is a perfect square. Write a program that asks the user for an integer and tells them if it is squarefree or not.

8. Write a program that swaps the values of three variables x, y, and z, so that x gets the value of y, y gets the value of z, and z gets the value of x.

9. Write a program to count how many integers from 1 to 1000 are not perfect squares, perfect cubes, or perfect fifth powers.

10. Ask the user to enter 10 test scores. Write a program to do the following:
    (a) Print out the highest and lowest scores.
    (b) Print out the average of the scores.
    (c) Print out the second largest score.
    (d) If any of the scores is greater than 100, then after all the scores have been entered, print a message warning the user that a value over 100 has been entered.
    (e) Drop the two lowest scores and print out the average of the rest of them.

11. Write a program that computes the factorial of a number. The factorial, $n!$, of a number n is the product of all the integers between 1 and n, including n. For instance, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$. [Hint: Try using a multiplicative equivalent of the summing technique.]

12. Write a program that asks the user to guess a random number between 1 and 10. If they guess right, they get 10 points added to their score, and they lose 1 point for an incorrect guess. Give the user five numbers to guess and print their score after all the guessing is done.

13. In the last chapter there was an exercise that asked you to create a multiplication game for kids. Improve your program from that exercise to keep track of the number of right and wrong answers. At the end of the program, print a message that varies depending on how many questions the player got right.

14. This exercise is about the well-known Monty Hall problem. In the problem, you are a contestant on a game show. The host, Monty Hall, shows you three doors. Behind one of those doors is a prize, and behind the other two doors are goats. You pick a door. Monty Hall, who

knows behind which door the prize lies, then opens up one of the doors that doesn't contain the prize. There are now two doors left, and Monty gives you the opportunity to change your choice. Should you keep the same door, change doors, or does it not matter?

(a) Write a program that simulates playing this game 10000 times and calculates what percentage of the time you would win if you switch and what percentage of the time you would win by not switching.

(b) Try the above but with four doors instead of three. There is still only one prize, and Monty still opens up one door and then gives you the opportunity to switch.