Chapter 1

Chapter 1 Exercises (Getting Started)

1. Use several print statements to print out a triangle that looks exactly like the one below.

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

2. Use several print statements to print out the letter A exactly like the one below.

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

3. Write a line of code that computes and prints out the result of the computation $\frac{14 \cdot 12}{33 \cdot 144 - 187}$.

4. Write a program that asks the user to enter a distance in kilometers and then prints out how far that distance is in miles. There are 0.621371 miles in one kilometer.

5. Write a program that asks the user to enter their name. Then print out the user's name three times on the same line.

6. Write a program that asks the user to enter two numbers (you'll probably want to use separate `input` statements for that). Then print out the result of adding those two numbers.

7. The Body Mass Index, BMI, is calculated as

$$\text{BMI} = \frac{703w}{h^2},$$

where w is the person's weight in pounds and h is the person's height in inches. Write a program that asks the user for their height their weight and prints out their BMI. [Note: one way to compute h^2 is as $h \cdot h$.]

8. Write a program that asks the user to enter five numbers (use five `input` statements). Then print out those numbers all on the same line, with each number separated from the others by exactly three spaces. Use the `sep` optional argument to the print statement to do this.

9. Write a program that asks the user to enter a number. Store that number in a variable. Add 2 to that number, store the result in the same variable, and then print out the value of that variable.

Chapter 2

Chapter 2 Exercises (For Loops)

Note: The programs in this section print out a lot of stuff. By default, when multiple `print` statements are used, each one will print on a separate line. However, by adding an optional argument like `end=''` or `end=''`, we can force things to stay on the same line. For instance, instead of `print('Hello')` we can use `print('Hello',end='')`. This will make the output of some of the programs easier to read and test. Note also that an empty `print()` statement will cancel out the effect of the `end` optional argument.

1. Write a program that asks the user to enter a word and then prints that word 25 times (each on separate lines).
2. Write a program that asks the user to enter a word and then prints that word 200 times, each on the same line.
3. Write a program that uses a for loop to print the numbers 5, 6, 7, 8, 9, . . . 89, 90, all on the same line separated by spaces.
4. Write a program that uses a for loop to print 2, 6, 10, 14, 18, . . . , 98, 102, all on the same line separated by spaces.
5. Write a program that uses a for loop to print 29, 28, 27, 26, 26, . . . , 5, 4, all on the same line separated by spaces.
6. Write a program that uses a for loop to print the output below:

```
1. A
2. A
3. A
4. A
5. A
```

7. Write a program that uses a for loop to print out the first 20 perfect squares, all on the same line. The first few are 1, 4, 9, 16, 25, ...
8. Write a program that uses for loops to print out 40 A's followed by 50 B's, all on the same line.
9. Write a program that uses a for loop to print out ABCABCABC. . . , where ABC repeats 100 times. Print this all on the same line.
10. Write a program that uses exactly three for loops to print the output below. It has 10 A's, followed by 5 copies of BCD, followed by one E, followed by 15 F's.

```
AAAAAAAAAABCDBCDBCDBCDBCDEFFFFFFFFFFFFFFF
```

11. Write a program that asks the user to enter a number and then prints out the letter A that many times, all on the same line.

12. Write a program that uses a loop and an `input` statement to ask the user to enter 10 numbers. After each number is entered, print out the square of that number.

13. Write a program that asks the user to enter a height and then draws a box like the one below that is 10 asterisks wide and as tall as specified by the user.

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

14. Write a program that asks the user to enter a size and then draws the letter C like the one below. It's width and height should be equal and the size specified by the user.

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

Chapter 3

Chapter 3 Exercises (Numbers)

1. Write a program that asks the user for two numbers x and y and then prints out the result of x^y.

2. Write a program that does the following computation in Python: $\frac{9+3}{8-2}$. The result should come out to 2. Be careful about order of operations.

3. Some board games require you to reduce the number of cards you are holding by half, rounded down. For instance, if you have 10 cards, you would reduce to 5 and if you had 11 cards you would also reduce to 5. With 12 cards you would reduce to 6. Write a program that asks the user to enter how many cards they have and print out what their hand would reduce to under this rule.

4. The distance between two numbers on the number line is the absolute value of the difference of the two numbers. For instance, the distance between 3 and 7.2 is $|3 - 7.2| = 4.2$. Write a program that asks the user for two numbers and prints out the distance between them.

5. Write a program that asks the user to enter a positive number and then prints out the square root of that number rounded to 2 decimal places.

6. Write a program that prints out the numbers from 1 to 20 and their square roots, rounded to 4 decimal places, with the square root of the number being on the same line as the number.

7. If you have a right triangle that is x units wide and y units tall, then using `atan2(y,x)` from the `math` module finds one of the triangle's angles. It returns the result in radians. The `degrees()` function can convert the result to degrees. Write a program that asks the user to enter a width and a height and prints out the angle returned by `atan2`, but in degrees and rounded to one decimal place.

8. In 24-hour time, hours run from 0 o'clock (midnight) to 23 o'clock (aka 11 pm). Write a program that asks the user for the current hour and for how many hours in the future they want to go. Have the program print out what the hour will be that many hours in the future. For instance, if it's 13 o'clock now, 27 hours from now it will be 16 o'clock. [Hint: Use the mod operator.]

9. Write a program that asks the user for a height in inches and prints out how many feet and inches that is. There are 12 inches in one foot. For instance, 40 inches is 3 feet and 4 inches. [Hint: use the `//` operator and the `%` operator to get each part.]

10. Write a program that generates two random numbers from 1 to 10, and prints out the two numbers and their sum.

11. Write a program that generates and prints 100 random numbers from 50 to 60, all on the same line, separated by spaces.

12. Write a program that asks the user to enter a letter. Then it generates a random number between 1 and 10 and prints out the letter that many times.

13. Write a program that asks the user to enter a positive integer. Then generate a random number between that number and 10 more than that number and print the letter A that many times on the same line.

14. Write a program that generates and prints 10 random zeroes and ones all on the the same line with no spaces between them. Then on the next line do the same but with 11 random zeroes and ones. On the next line do the same but with 12 random zeroes and ones. In total there should be 50 lines, each with one more random value than the previous line. The first few lines might look like below. [Hint: You will want to use one for loop nested within another.]

```
1000110101
11100100101
110101000100
0010010001000
```

15. This problem is about finding the day of the week of any date since about 1583. Ask the user to enter the year y, month m, and day d separately. The following formulas give the day of the week:

$$\begin{aligned}
p &= \left\lfloor \frac{14-m}{12} \right\rfloor \\
q &= y - p \\
r &= q + \left\lfloor \frac{q}{4} \right\rfloor - \left\lfloor \frac{q}{100} \right\rfloor + \left\lfloor \frac{q}{400} \right\rfloor \\
s &= m + 12p - 2 \\
t &= \left(d + r + \left\lfloor \frac{31s}{12} \right\rfloor \right) \bmod 7
\end{aligned}$$

The $\lfloor\ \rfloor$ brackets indicate the floor function. In Python, you can do this simply by doing integer division. For instance, p can be calculated just by doing `(14-m)//12`.

The day of the week is given by t, with $t = 0$ corresponding to Sunday, $t = 1$ corresponding to Monday, etc.

Chapter 4

Chapter 4 Exercises (If Statements)

1. Write a program that asks the user to enter a programming language. If they enter *python*, then print out the message “This program was written in Python.” Otherwise nothing should happen (no else statement is necessary).
2. Write a program that asks the user to enter a number. If the number is negative, print out a message telling them they can’t enter a negative. Otherwise print out the square root of that number.
3. Ask the user to enter a temperature in Celsius. If the temperature is less than or equal to 0, print “Warning: Low temperature.” If the temperature is between 0 and 35, print ”Temperature is okay.” Otherwise, print out “Warning: High temperature.”
4. Ask the user to enter a number of days. If the user enters 28 or 29, the program should print out ”Feb”. If they enter 30, the program should print out “Apr, Jun, Sep, Nov”. If they enter 31, the program should print out “Jan, Mar, May, Jul, Aug, Oct, Dec”. For any other entry, the program should print out “Error”.
5. Ask the user for a length in feet. Then ask them which unit they want to convert to (inches, centimeters, or meters) and print out the converted value. There are 12 inches in a foot, 30.48 cm in a foot, and .3048 meters in a foot. If the unit chosen is not one of the above, then print out an error message.
6. Ask the user to enter a number from 20 through 99. If the number is in the 20s, print out “veinte”, which is Spanish for twenty. If the number is in the 30s, print out “treinta”, Spanish for thirty. Do a similar thing for all the other ranges. The Spanish number names for 40 and beyond are *cuarenta*, *cincuenta*, *sesenta*, *setenta*, *ochenta*, and *noventa*. If the number is not in the range from 20 through 99, print out an error message.
7. Write a program that asks the user to enter a number. If the number is *not* 0, 2, 5 or between 10 and 15 or between 20 and 25, then the word “Okay” should be printed. Otherwise, nothing should happen.
8. Mods are useful for telling if one number is divisible by another. Write a program that asks the user for a number and then prints out whether that number is divisible by 7.
9. Write a program that prints out all the numbers from 1 to 100 that are divisible by 3 or 7 but not divisible by both.
10. Write a program that generates 100 random zeros and ones. Whenever a 0 is generated, the program should print out a question mark and whenever a 1 is generated an asterisk should be printed. Have everything be printed on the same line.
11. Write a program that prints 100 random letters that can be either A, B, C, D, or E, all on the same line.
12. Write a program that prints 100 random zeros and ones on the same line, but such that zeros are twices as likely as ones to appear.

13. This is a very simple billing program. Ask the user for a starting hour and ending hour, both given in 24-hour format (e.g., 1 pm is 13, 2 pm is 14, etc.). The charge to use the service is $5.50 per hour. Print out the user's total bill. You can assume that the service will be used for at least 1 hour and never more than 23 hours. Be careful to take care of the case that the starting hour is before midnight and the ending time is after midnight.

14. Write a program that prints out on separate lines the following 200 items: `file0.jpg`, `file1.jpg`, `...`, `file199.jpg`, except that it should not print anything when the number ends in 8, like `file8.jpg` ,`file18.jpg`, etc. [Hint: `if i % 10` is not 8, then `i` doesn't end in 8.]

15. The Collatz conjecture is a well-known math problem. Start with any number x. If it is even, divide it by 2. If it is odd, compute $3x + 1$. Then repeat the process with the next number, the one after that, etc. Collatz conjectured that no matter where you start, you will always eventually get to the number 1.

    For instance, starting with $x = 3$, since 3 is odd, we compute $3x + 1$ to get 10. Then since 10 is even, we divide by 2 to get 5. Then since 5 is odd, we do $3x + 1$ to get 16. Then since 16 is even, we divide by 2 to get 8. Continuing, we then get 4, 2, and 1, at which point the sequence starts to repeat.

    Write a program that asks the user for a starting value and then prints out the first 30 terms of the Collatz sequence, all on the same line.

16. One way to solve math equations is by trial and error. Computers can help with that. The equation $21x^2 - x^3 + 21904 = 0$ has an integer solution between 1 and 100. Use a for loop and an if statement to find it.

17. Write a program to play the following game. The program should randomly generate 10 addition problems with three numbers ranging from 20 to 50 (for example $23 + 50 + 37$). For each problem, the user guesses the answer. If they get the answer exactly right, print "Right!" If their answer is within 5 of the correct answer, print "Close!" Otherwise, print "Wrong."

Chapter 5

Chapter 5 Exercises (Miscellaneous Topics I)

1. Write a program that generates 50 random numbers from 1 through 9. Print out all 50 numbers on the same line. Then print out how many fives are generated.
2. Write a program that adds up all the numbers from 1 through 100 and prints the result.
3. Write a program that adds up the sines (use the `sin()` function from the `math` module) of all the numbers from 1 through 100 and prints the result.
4. Write a program that adds up $1 + 3 + 5 + 7 + \cdots + 999$.
5. Write a program that uses a loop to ask the user to enter 10 numbers. Print out how many numbers are positive and add up all the numbers.
6. Write a program that randomly generates 10 numbers from 1 to 5 and asks the user to guess each number. For each guess print out whether it is right or wrong. At the end, print out how many the user gets right and how many they get wrong.
7. Write a program that randomly generates 10 numbers from 1 to 10 and asks the user to guess each number. For each guess print out whether it is right, wrong, or close (within in 1 of the right answer). Also keep score, where players score 5 for a correct guess, 2 for a close guess, and -1 for a wrong guess. Print the accumulated score after each guess.
8. One way to estimate probabilities is to run what is called a *computer simulation*. Here we will estimate the probability of rolling doubles with two dice (where both dice come out to the same value). To do this, run a loop 10,000 times in which random numbers are generated representing the dice and a count is kept of how many times doubles appear. Print out the final percentage of rolls that are doubles.
9. In the game Yahtzee, players roll five dice. A Yahtzee is when all five dice are the same. Write a program that simulates rolling five 10,000 times and counts how many Yahtzees occur. Print out what percentage of the rolls come out to be Yahtzees.
10. Write a program that asks the user for four values — w, x, y, and z. Find the smaller of x and y, the smaller of y and z, and swap their two values. Then print out the values of w, x, y, and z.
11. Write a program that generates 20 random numbers from 1 through 10. Print all of the generated numbers on the same line. Just once, after all the numbers have been printed, print out whether or not the number 10 was generated.
12. Write a program that generates 20 random numbers from 1 through 10. Print all of the generated numbers on the same line. Just once, after all the numbers have been printed, print out whether or not the same number was ever generated twice in a row. [Hint: use a variable to keep track of the most recently generated number.]

13. Write a program that asks the user to enter 10 numbers between 0 and 100. After all the numbers are entered, of all the numbers that the user enters that are between 50 and 100, print out the smallest one. If none are in that range, then print out a message indicating that fact.

14. Write a program that asks the user to enter 10 numbers. Then print out the two smallest numbers entered (they might have the same value).

15. Consider the following sequence: 1, 1, 3, 7, 17, 41, 99, 239, 577, 1393, ... The first two terms are 1 and each term in the sequence is obtained from the previous two terms by taking twice the previous term and adding it to the term before that. For example, $3 = 2 \cdot 1 + 1$, $7 = 2 \cdot 3 + 1$, and $17 = 2 \cdot 7 + 3$. In equation form, each term a_n is given by $a_n = 2a_{n-1} + a_{n-2}$. Write a program that asks the user for a value, n, and prints out the first n terms of the sequence.

16. Consider the following sequence: 1, 2, 3, 2.0, 2.333, 2.444, 2.259, 2.345, ... The first three terms are 1, 2, 3 and each additional term in the sequence is obtained from the average of the previous three terms. For example, the fourth term is $(1+2+3)/3 = 2$, and the fifth term is $(2+3+2)/3 = 2.333$. In equation form, each term a_n is given by $a_n = (a_{n-1} + a_{n-2} + a_{n-3})/3$. Write a program that asks the user for a value, n, and prints out the first n terms of the sequence.

Chapter 6

Chapter 6 Exercises (Strings)

1. Create the following Python string: `Didn't Hamlet say "To be or not to be"?`
2. Create a multiline string that has the letter A on the first line, the letter B on the second line, a single backslash (`\`) on the third line, and the letters C and D separated by a tab (`\t`) on the fourth line.
3. Write a program that asks the user to enter a string of at least six characters. Then print out the following:
   (a) The length of the string
   (b) The second character of the string
   (c) The last character of the string
   (d) The first five characters of the string
   (e) The last two characters of the string
   (f) The string backwards
   (g) Every character of the string except the last one
   (h) Every character of the string except the first and last
   (i) If the string contains a lowercase *a*, print out the index of the first *a*. Otherwise say that there is no lowercase *a*
   (j) The string in all caps
   (k) The string with every space replaced with an underscore
4. Write a program that asks the user to enter two strings. Then output the first string repeated 10 times and output the concatenation of the two (original) strings.
5. Write a program that asks the user to enter a number (but just read in that value as a string). Print out whether that number is an integer or a floating point number. Floating point numbers contain decimal points, while integers don't.
6. Write a program that asks the user to enter a programming language. If the language entered is Python, then print out a message saying that they entered Python. Otherwise don't print out anything. The program should work regardless of which letters of Python are capitalized (it should recognize, Python, python, PYTHON, and even PyTHoN).
7. Write a program that asks the user to enter a string. If the string contains any periods, commas, or semicolons, then print out the message "There is some punctuation." Otherwise, print out a count of how many spaces there are in the string.
8. Write a program that asks the user to enter a sentence, removes all the spaces from the sentence, converts the remainder to uppercase, and prints out the result.

9. Write a program that asks the user to enter a string, replaces all the spaces with asterisks, then replaces every fifth character (starting with index 0) with an exclamation point, and finally concatenates three copies of that result together. For example, `this is a test` would become `!his * !s * a * !est!his * !s * a * !est!his * !s * a * !est`.

10. Write a program that asks the user to enter a string. If the string is at least five characters long, then create a new string that consists of the first five characters of the string along with three asterisks at the end. Otherwise add enough exclamation points (!) to the end of the string in order to get the length up to five.

11. Write a program that asks the user to enter a string. Then loop through the string and print out all the characters of the string with a space inserted after each character. For instance, if the user enters `ABCDE`, the output would be `A B C D E`.

12. Write a program that asks the user to enter a string. Then loop through the string and create a new string that consists of all the characters of the original with spaces inserted after each character. For instance, if the user enters `ABCDE`, the new string would be `'A B C D E '`. Make sure this new string is stored in a variable. Print out the value of the new string.

13. Write a program that asks the user to enter a string. Print out whether the string contains more capital As than capital Bs, the same amount, or more capital Bs.

14. Write a program that asks the user to enter a string and prints out how many letters (uppercase and lowercase) are in the string. [Hint: It's possible to use the string `count()` method to do this, but it might be easier to use the counting technique from Chapter 5 along with the `isalpha()` method.]

15. Write a program that asks the user to enter a string. Print out whether the string contains more capital letters than lowercase letters, the same amount, or more lowercase than capitals.

16. Write a program that asks the user to enter a string of lowercase letters. Then print out all the lowercase letters that do not appear in the string.

17. Write a program that asks the user to enter two strings. Print out whether or not the first string contains any characters that are not in the second string.

18. Write a program that asks the user to enter two strings. If the strings have different lengths, then print out a message indicating such. Otherwise, print out whether or not the strings differ in exactly one position. For instance, ABCDE and ABCXE differ in exactly one position (index 3).

19. Write a program that prints 10 random lowercase letters, all on the same line.

20. Write a program that asks the user to enter a string. Then create a new string that consists of 10 randomly selected characters from the user's string. Print the new string.

21. Write a program that asks the user to enter a string of lowercase letters. Then for each lowercase letter, if the letter is not in the string, print out a message indicating that and otherwise print out the index of the first occurrence of that letter in the string.

22. Write a program that asks the user to enter an email address. Assume for this problem that email addresses are of the form `username@somedomain.com`, where there is a user name and a domain name separate by the `@` symbol. Print out the user name and domain name on separate lines.

23. Write a program that does the following: It first asks the person to enter their first and last names (together on the same line). For simplicity, assume their first and last names are one word each. Then ask them for their gender (male/female). Finally ask them if they prefer formal or informal address. Uppercase/lowercase should not matter when they make their selections. Then print a line that says hello to them, using Mr. or Ms. <last name> if they ask for formal address and just their first name otherwise. Here are two sample runs:

```
Name: Don Knuth                                Name: Grace Hopper
Gender: male                                   Gender: Female
Formal/Informal: informal                      Formal/informal: Formal
Hello, Don.                                    Hello, Ms. Hopper.
```

24. Write a program that ask the user to enter a string that consists of multiple words. Then print out the first letter of each word, all on the same line.

25. Write a program that asks the user to enter a string. Then print out (all on the same line) the indices of the string that contain letters. For instance, if the user enters `A B?*C`, the program would output 0, 2, and 5 since the only letters are at those indices.

26. Write a program that asks the user to enter an integer n. Then produce output that changes one letter at a time from a string of all As to a string of all Bs, starting at the back of the string. Shown below is the output with $n = 4$.

```
AAAA
AAAB
AABB
ABBB
BBBB
```

27. IP addresses are important in computer networking. They consist of four numbers, each from 0 to 255, separated by dots. For instance, one IP address of www.google.com is 74.125.22.99. IP addresses of the following forms are considered special local addresses: 10.*.*.* and 192.168.*.*. The stars can stand for any values from 0 to 255. Write a program that asks the user to enter an IP address and prints out whether it is in one of these two forms. You can assume that the user enters a valid address. [Hint: Consider the `startsWith` method. For example, `if s.startswith('ab')` tells whether or not `s` starts with the letters `ab`.]

28. Write a program that asks the user to enter their name in lowercase and replaces each character of their name with the letter immediately following it in the alphabet (with *a* following *z*). [Hint: To avoid using 26 if statements, a string containing all the letters of the alphabet along with the `index` method might be helpful, or else look up the `ord()` and `chr()` functions.]

Chapter 7

Chapter 7 Exercises (Lists)

1. Write a program that asks the user to enter a list of at least five integers. Do the following:
   (a) Print out the total number of items in the list.
   (b) Print out the fourth item (index 3) in the list.
   (c) Print out the last three items in the list.
   (d) Print out all the items in the list except the first two.
   (e) Print out the list in reverse order.
   (f) Print out the largest and smallest values in the list.
   (g) Print out the sum of all the values in the list.
   (h) If the list contains a zero, print out the index of the first zero in the list, and otherwise print out a message saying there are no zeroes.
   (i) Sort the list and print out the list after sorting.
   (j) Delete the first item from the (now sorted) list and print out the new list.
   (k) Change the second-to-last item in the list to 9876 and print out the new list.
   (l) Append the value -500 to the end of the list and print out the new list.
2. Using the `*` operator, write a single line of code that creates a list consisting of 100 zeroes. On the next line, print out the list.
3. Write a program that creates the list `L = [1,2,3,4,5]`.
   (a) Attempt to make a copy of the list by using `copy=L`. Then set the first item in `L` to 999, and print out `copy`.
   (b) Set `L` back to `[1,2,3,4,5]`, make a proper copy of `L`, set the first item in `L` to 9, and print out `copy`.
4. Write a program that creates and prints a list of the square roots of the integers from 1 to 100, each rounded to 4 decimal places.
5. Write a program that generates and prints a list of 10 random numbers from 1 through 20.
6. Write a program that generates a list of 20 random zeroes and ones. Print out the list and then print out how many zeroes there are.
7. Write a program that generates and prints a list of 20 random numbers from 1 through 1000. Then print out how many even numbers are in the list.
8. Write a program that asks the user to enter a list of numbers from 1 to 100. Create a new list that contains just the entries from the random list that are greater than 50.

9. Write a program that asks the user to enter a list of numbers. Then print out the smallest thing in the list and the first index at which it appears in the list.

10. Write a program that asks the user to enter a list of numbers with some of the values greater than 10. Print out the smallest item in the list that is larger than 10.

11. Write a program that asks the user to enter a list of numbers. Then replace all the even numbers in the list with zeroes and print out the list and replace each odd number in the list with two ones. For instance, if the first few things in the list are `[33,2,14,197,26, ...`, then the list would become `[1,1, 0, 0, 1, 1, 0, ...`. Print out the new list. [Hint: It might be easier to build up a new list rather than trying to directly modify the list itself.]

12. Write a program that asks the user to enter a list of at least three numbers. Then print the following:
    (a) The average of the items in the list
    (b) The three smallest things in the list.
    (c) The average of all the items in the list except the smallest.

13. Write a program that asks the user for a month number and the print out the month name associated with that number (where 1 is January, 2 is February, etc.). Do this in two ways:
    (a) using 12 if statements
    (b) using lists

14. Write a program that creates and prints the following list that has one 1, two 2s, three 3s, etc., finishing with ten 10s: `[1,2, 2, 3, 3, 3, 4, 4, 4, 4, ...,10,10,10,10,10,10,10,10,10,10]`

15. Write a program that asks the user to enter two lists of the same length. Then create a new list where the entry at each index i in the new list is the larger of the entries at index i of the user's two lists. For instance, if the two lists are `[1,25,3]` and `[18,2,30]`, then new list would be `[18,25,30]`.

16. Write a program that asks the user to enter a list. Then create a new list of "cumulative sums" from the user's list. The ith entry of the new list is the sum of the first i entries of the user's list. For instance, if the user enters `[1,3, 5, 7]`, the generated list would be `[1,4, 9, 16]` (where $4 = 1 + 3$, $9 = 1 + 3 + 5$, and $16 = 1 + 3 + 5 + 7$).

17. Write a program that asks the user for an integer `n` and then generates and prints a list of the first `n` triangular numbers. The triangular numbers are 1, $3 = 1 + 2$, $6 = 1 + 2 + 3$, $10 = 1 + 2 + 3 + 4$, etc.

18. Write a program that asks the user to enter two lists of the same size. Then put the lists together into a new list by alternately taking items from the two lists, like in a riffle shuffle. For instance, if one list is `[1,2, 3]` and the other is `[7,8, 9]`, the new list would be `[1,7, 2, 8, 3, 9]`.

19. Write a program that rotates all the elements in a list one element to the left, with the first element rotating to the end. For instance, if the list is `[1,2,3,4]`, it would rotate into `[2,3,4,1]`.

20. Write a program that generates a list of 20 random 0s, 1s, and 2s. Then create a new list which consists of all the indices at which a consecutive repeat occurs (where the item at the current index equals the item at the index right before it). For instance, if the list starts out with `[0,2, 2, 1, 0, 1, 1,...`, there are repeats at indices 2 and 6.

21. Write a program that asks the user to enter a list of at least six items. Then create a new list that consists of the first five things of the user's list, but in reverse order, and then has the rest of the user's list unchanged. For instance, if the user enters `[2,4,7,8,9,14,15,16]`, then the new list should be `[9,8,7,4,2,14,15,16]`.

22. Write a program that asks the user to enter a string of lowercase letters and creates a list containing counts of how many times each letter appears in the string. The first index is how many a's are in the string, the second is how many b's, etc.

23. Write a program that asks the user for how many students are in a class. The program should then generate and print a list of "grades" for those students that are all random numbers from 90 to 100, except that a random one of the items in the list should be 0.

24. Write a program that generates a list of 100 random numbers from 1 to 50. Print out the list and then write some code that changes the first two occurrences of 50 to -999. If there is just one 50 in the list, then just change it. If there are no 50s in the list, then print a message saying so.

25. In the dice game Yahtzee you roll 5 dice. A large straight is when the dice come out as 1-2-3-4-5 or 2-3-4-5-6. Write a program that simulates rolling five dice 10,000 times, and count how many times a large straight happens. [Hint: the dice are five random numbers. Put them into a list and sort them. That makes it easy to compare if they are 1-2-3-4-5 or 2-3-4-5-6.]

26. Write a program that asks the user to enter a list. Then shuffle the list using the following algorithm: Pick a random element of the list and swap it with the element at index 0. Then pick a random element from among the entries at index 1 through the end of the list and swap it with the element at index 1. Then pick a random element from among the entries at index 2 through the end of the list and swap it with the element at index 2. Continue this until the end of the list is reached. Print out the shuffled list.

27. Write a program that asks the user to enter a list of strings and a list of indices. Then create a new list that consists of all the strings from the user's list except for those at the indices from the list list of indices. For instance, if the list of strings is `['a','bc','d','e','fg']` and the list of indices is `[2,4]`, then the new list would be `['a','bc','e']`.

Chapter 8

Chapter 8 Exercises (More with Lists)

1. Write a program that asks the user to enter a list of at least four things. Then do the following:
    (a) Use the `choice()` function to print out random item from the list.
    (b) Use the `sample()` function to print out three random items from the list.
    (c) Use the `shuffle()` function to reorder the list and then print out the list.
2. Write a program that creates a random string of length 100, where each entry has an equal chance of being either an H or or T. The idea is that the string represents the result of 100 coin flips. Use the `choice()` function to do this.
3. Write a program that creates a string of consisting of 1000 random As, Bs, Cs, and Ds, where A has a 60% chance of being chosen, B has a 30% chance, C has a 8% chance, and D had a 2% chance. [Hint: Create a list or string that has 60 As, 30 Bs, 8 Cs, and two Ds, and then repeatedly use the choice function.]
4. Write a program that asks the user for a length and then generates a random password of that length. The password's characters can be lowercase and uppercase letters, digits, and any of these special characters: `!@#$%^&*()`
5. Write a program that contains a list of at least 10 verbs, a list of at least 10 nouns, and a list of at least 10 adjectives. Then have the program randomly generate and print a sentence of the form "The <*adjective*> <*noun 1*> <*verb*> the <*noun 2*>." All the words except "the" should be randomly pulled from the appropriate lists. It is okay if the two nouns are the same.
6. Repeat the previous problem, but make sure that the two nouns are different.
7. Write a program that creates a list of the 52 cards in a standard deck. Represent the cards as strings with the value followed by the suit. The possible values are `Two`, `Three`, . . . , `Ten`, `Jack`, `Queen`, `King`, and `Ace`. The possible suits are `Clubs`, `Diamonds`, `Hearts`, and `Spaces`. A typical card would be a string like `SevenofClubs`. Then shuffle the cards and print out the first five cards in the shuffled list. [Hint: Rather than manually typing in 52 card names, you can use a for loop nested inside another for loop to create the list.]
8. Write a program that asks the user to enter a word that's at least three letters long. Then choose exactly three indices in that word to replace with asterisks and print out the result. For instance, if the word is `python`, then a possible output could be `p*t**n`.
9. Write a program that takes a string and then one-by-one randomly replaces all of its characters by asterisks, printing out each step. For instance, if the word is `'list'`, then a possible output is `l*st`, `l**t`, `***t`, `****`.
10. Create a list that contains the numbers 1 through 20 in order. Then have the program randomly pick three pairs of elements in the list and swap them. Print out the resulting list. An element may be swapped more than once, but it may not be swapped with itself. A possible output is

`[7,2,3,4,5,6,1,8,12,10,11,9,13,14,15,18,17,16,19,20]`. The swaps in this example are `1,7`, then `9,12`, and finally `16,18`.

11. Write a program that asks the user to enter a bunch of words separated by spaces. Then use the `split()` method to create a list of words the user enters.

12. When sending emails to multiple people, addresses are usually separated by semicolons. Write a program that asks the user to enter a list of email addresses (as a list of strings) and then use the `join()` method to join the emails into a single string with semicolons separating them.

13. Write a program that asks the user to enter a date in a form `month/day/year` and then uses the `split()` method to print out the month, day and year all on separate lines.

14. Write a program that asks the user to enter a bunch of words separated by spaces, all in lowercase. Then capitalize the first letter of each word and print out the resulting string. [Hint: Use `split()` and `join()`.]

15. Here is a nice way to create a list of all the rearrangements of the string `abcd`:

    ```
    from itertools import permutations
    for x in permutations('abcd'):
        print(x)
    ```

    The output looks a little messy. Instead, we would like everything outputted as strings, like `abcd`, `abdc`, etc. Use the `join()` function to accomplish this.

16. Ask the user to enter several sentences. Then print out the last word of every sentence. For this problem, assume that sentences always end with a period and that there are no periods elsewhere in a sentence.

17. Write Python list comprehensions to solve each of the following.

    (a) Write a program that creates and prints a list of the square roots of the integers from 1 to 100, each rounded to two decimal places.

    (b) Write a program that asks the user to enter a list of integers. Then create and print a list that consists of each entry of the user's list with 1 added to it. For instance, if the entry is is `[3,4, 5]`, the new list should be `[4,5, 6]`.

    (c) Write a program that asks the user to enter a list of integers. Then create and print a list containing only the even integers from the user's list.

    (d) Write a program that asks the user to enter a list of integers. Then create and print a new list that consists of all entries of the user's list, except with negatives replaced by 0. For instance, if the entry is `[2,-4,5,-6,-7]`, the new list should be `[2,0, 5, 0, 0]`. [Hint: use `max(x,0)`.]

    (e) Write a program that asks the user to enter a string. Then create and print a list that contains only the letters from the user's string (and no other character besides letters). [Hint: use the `isalpha()` method.]

    (f) Write a program that asks the user to enter a list of lists. Then create and print a list that contains the last elements of each of the lists in the master list. For instance, if the user enters `[[1,2,3],[4,8],[15,13]]`, the created list would be `[3,8,13]`.

    (g) Write a program that asks the user to enter a string of lowercase letters. Then create and print a list of 26 items, consisting of the frequencies of each letter in alphabetical order. For the string `helloworld`, the list would be `[0,0,0,1,1,0,0,1,0,0,0,3,0,0,2,0,0,1,0,0,0,0,1,0,0,0]`. [Hint: use the `count()` method.]

    (h) Write a program that asks the user to enter two strings of equal length. Then create and print a list of all the indices in which the strings differ. For instance, given `python?` and `Python!`, the list would be `[0,6]`.

18. Write a program that asks the user to enter a list of test scores. Then create and print a list of lists, where the first list contains all the As (90 or above), the second all the Bs (80–89), etc. (where Cs are 70–79, Ds are 60–69, and Fs are 0–59).

19. Write a program that creates a 6×6 list of random integers from 0 to 9. Then do the following:
    (a) Print out the list.
    (b) Print out a count of how many zeros are in the list.
    (c) Print the smallest thing in the first row (row 0).
    (d) Print the largest thing in the third column (column 2).
    (e) Print the sum of the entries in the last row.
    (f) Create and print a new one-dimensional list from the two-dimensional list such that all the entries in row 0 of the original are followed by all the entries in row 1, which are followed by all the entries in row 2, etc. See the example below:

    ```
    2 4 9 1 1 0
    5 7 5 4 2 1
    6 6 3 5 9 2
    7 1 2 0 9 4
    7 0 2 1 8 7
    7 0 0 0 3 7

    [2,4,9,1,1,0,5,7,5,4,2,1,6,6,3,5,9,2,7,1,2,0,9,4,7,0,2,1,8,7,7,0,0,0,3,7]
    ```

    (g) Create a new two-dimensional list by swapping the rows and columns of the original. That is, row 0 of the original should become column 0 of the new one, row 1 of the original should become column 1 of the new one, etc.

20. The following is useful as part of a program to play *Battleship*. Write a program that creates and prints a 10×10 list that consists of random zeroes and ones. Then ask the user to enter a row and column number. If the entry in the list at that row and column is a one, the program should print `Hit` and otherwise it should print `Miss`.

21. Write a program that first creates and prints a 6×6 list of random zeroes, ones, and twos. Then ask the user to enter a starting row, ending row, starting column, and ending column, and print out the sum of all the entries in the list within the range specified by the user.

22. Write a program that generates and prints a 15×15 list of random zeros, ones, and twos, and then adds up all the entries on or below the main diagonal. These are all entries (r, c), where the column number c is less than or equal to the row number r.

23. Write a program that first creates and prints a 6×6 list of integers from 0 to 5. Then write some code that determines if there are two consecutive zeros in the list, either horizontally or vertically. Try to do this with loops rather than with dozens of if statements.

24. Ask the user for an integer n. Then create and print a $n \times n$ list that consists of spaces everywhere except that the first row, last row, first column, last column, main diagonal, and off diagonal should all be # symbols. Below is an example of what the list should look like with $n = 10$. Remember that you are creating an list, not just printing characters.

    ```
    ##########
    ##      ##
    # #    # #
    #  #  #  #
    #   ##   #
    #   ##   #
    #  #  #  #
    # #    # #
    ##      ##
    ##########
    ```

25. Write a program that creates and prints a 10×10 list that has exactly 10 zeroes and 90 ones, all randomly ordered. [Hint: One way to do this problem is to create a list consisting of 10 zeros and 90 ones, shuffle the list, and then add the elements of that list one-by-one to the 10×10 list, where row r, column c of the 10×10 list is mapped to index i of the shuffled list by `r=i//10`, `c=i%10`.

26. Write a program that creates and prints a 10×10 list of random zeroes and ones, where for each entry, when it is generated, there is a 10% chance that it will be a zero and a 90% chance it will be a one. Unlike the problem above, there need not be exactly 10 zeroes and 90 ones in the finished product.

Chapter 9

Chapter 9 Exercises (While Loops)

1. Using a while loop, write a program that prints out the integers 2, 5, 8, 11, 14, 17, and 20 in that order, all on the same line.

2. Using a while loop, write a program that counts down from 100, stopping at 1, printing all the numbers on the same line.

3. Write a program that repeatedly asks a user to enter positive integers. The user stops the program by entering a negative. After this, print the sum of all the numbers the user enters, excluding the final negative.

4. Write a program that asks the user to enter numbers from 1 to 10. The program should stop when the user enters a 5. After the loop is done, the program should print out a count of how many numbers were entered and print out yes or no, depending on whether the user entered any numbers less than 3.

5. Write a program that asks the user to enter a string of at least five characters. If the user enters a string that is not long enough, then a message should be printed and they should be asked again for the string. Keep doing this until they enter a long enough string and then print out the fifth character (index 4) of that string.

6. Write a program that asks the user to enter a string of at least five characters. If the user enters a string that is not long enough, then a message should be printed and they should be asked again for the string. Keep doing this until they enter a long enough string. Then ask the user to enter an index in that string. If the user enters a negative or a number greater than or equal to the length of the string, then tell them to enter a valid index. Keep doing this until they enter a valid index and ask for the index again. Then print out the character of the user's string at the index that they entered.

7. Write a program that repeatedly asks the user to enter a string. For each string, print out whether or not it is a palindrome (reads the same backwards as forwards). The user indicates they are done by pressing enter without entering a string (which means their input will be the empty string).

8. Write a program that asks the user to enter a positive number between 20 and 100. Then repeatedly subtract random numbers in the range from 1 to 10 from that value until the number becomes negative. After each subtraction, print out the new value of the number. For instance, if the user enters 25, the program might print out something like 25, 18, 16, 7, 3, -2.

9. The Collatz conjecture is a well-known math problem. Start with any number x. If it is even, divide it by 2. If it is odd, compute $3x + 1$. Then repeat the process with the next number, the one after that, etc. Collatz conjectured that no matter where you start, you will always eventually get to the number 1.

   For instance, starting with $x = 3$, since 3 is odd, we compute $3x + 1$ to get 10. Then since 10 is even, we divide by 2 to get 5. Then since 5 is odd, we do $3x + 1$ to get 16. Then since 16 is even, we divide by 2 to get 8. Continuing, we then get 4, 2, and 1, at which point the sequence starts to repeat.

Write a program that asks the user for a starting value and then prints out the all the terms of the Collatz sequence, stopping when the a 1 is reached.

10. Write a program that builds up a list, by adding random numbers from 1 to 10 to the list, stopping when the fifth 10 is generated. Print out the list at the end.

11. Write a program that creates and prints a list of 50 random numbers from 1 to 5 with the property that the same number never appears twice or more in a row. [Hint: One way to do this is to use a while loop to make sure the current number being generated is not equal to the previous number.]

12. Write a program that asks the user to enter a list of numbers. Replace the first negative number by 0, leave the rest of the list alone, and print the new list. If there are no negatives, then the list doesn't change. [Hint: make sure as part of your while loop that you check to make sure the index doesn't go past the end of the list.]

13. Write a program that asks the user to enter a list of numbers. Replace the first three negative numbers by 0, leave the rest of the list alone, and print the new list. If there are less than three negatives, then all the negatives will become 0, and if there are no negatives, then the list doesn't change.

14. Write a program that asks the user to enter a string that consists of some letters and some non-letters. Print out the substring that consists of all the characters of the string from the first character up through, but not including, the first non-letter. For instance, if the string is `abc#defg`, the output would be `abc`. The output might end up being the empty string if the first character is not a letter.

15. Write the following game. The player starts with 100 points. The program then picks a random number from 1 to 10 and asks the player to guess it. If they guess it right, they gain 100 points. If their guess is less than the number, then they lose 10 points. If their guess is greater, they lose 20 points. After the guess, print out the result of the guess and the player's amount of points. The computer then picks a new number and play continues in the same way until the player gets to 200 or more points (in which case they win) or 0 or less points (in which case they lose). Print out a message at the end indicating whether the player won or lost.

16. Write the following game. The player is given a random multiplication problem to answer. Make it so that the random numbers range from 2 to 99. For each problem, the player gets one guess. If they get it right, the number of points added to their score is the sum of the two numbers in the problem (e.g., if the problem is `7 x 12`, they would earn 19 points). If they get it wrong, their score doesn't change, but they lose a guess. After each problem, print out whether they get it right and print out their score and how many guesses are remaining. The player starts with three guesses. The game ends when the player either wins by reaching a score of 200 or loses by running out of guesses.

17. Consider the following game. The player starts out with $50. They have to pick a number from 1 to 10. The computer then picks a random number from 1 to 10. If the player's number matches the computer's or if the player's guess is not in this range, then the player owes the computer $50 and the game ends. If not, the player's money doubles to $100 and they keep playing.

    This time the player and the computer pick numbers from 1 to 9, and if their numbers match or if the player's guess is not in this range, then the player owes the computer all the money ($100). Otherwise, the player's money doubles to $200 and the game continues, this time with numbers from 1 to 8. This process continues, with the range of numbers shrinking by 1 each turn, until the last turn with numbers only ranging from 1 to 2.

    If the player's and computer's numbers ever match or if their guess is not in the proper range, the game is over and the player has to pay the computer whatever the money amount has gotten to. At any point, instead of guessing a number, the player can type in 0 to quit the game and leave with their current winnings.

    Write a program that plays this game, printing out after each guess whether the player gets it right or wrong, and how much money they have. Also print out the result of the end of the game.

18. Write a program that asks the user to enter a string that consists of some letters and some non-letters. Then randomly remove the non-letters one at a time from the string until there are no more letters. After each letter is removed, print out the string.

19. Write a program that asks the user for an integer `n` and then generates and prints a list of the first `n` prime numbers.

20. Write a program that asks the user to enter a positive integer. Then generate and print random numbers from 1 to `n` until a repeat is generated. Please also label the numbers in the order they are generated. The output should look something like this (stopping at 12 because of the repeat).

```
1. 7
2. 12
3. 15
4. 12
```

21. Here is an interesting question: Suppose a country has a policy where couples must continue having kids until they have a boy, at which point they can have no more kids. What will the long term percentage of boys and girls come out to? Ignore the possibility of twins and other complications. Find an approximate answer to this question by simulating 10,000 couples having kids according to this rule. It should count the total number of boys and girls born and return the percentage of girls.

22. Consider a while loop that generates random numbers from 1 to 10 until a 6 comes up. Sometimes a 6 will come up right away and sometimes it will take quite a while. For instance, if we print out the numbers that the loop generates, we might see the sequence `2,3,1,3,6` or the sequence `2,1,9,9,2,2,10,8,9,6`. We are interested in how long it could possibly take a 6 to come up. Write a program that simulates this by running this experiment 10,000 times times, and reports how long the longest sequence of numbers turns out to be from those 10,000 trials.

23. People often try to collect all the items from a set. Suppose we have a set of 100 items that we are trying to collect. The items are sold in packages of 20 items, with all 20 items in each package being different. Write a program that simulates how long it will take until we have collected all 100 items. Do this by randomly generating packages of 20 and adding them to a collection until that collection has all 100 different items.

24. Write a program that asks the user to enter two lists. Then sort both of those lists and "merge" them. Merging two lists means that we build up a new list by looping through each of the lists, comparing the current items in the two lists and adding the smaller item to the new list. For instance, if the two lists are `[1,4, 5, 8, 10]` and `[2,3, 7]`, then the merged list is `[1,2, 3, 4, 5, 7, 8, 10]`, which comes from first taking 1 from the left list, then taking 2 and 3 from the right list, then taking 5 from the left list, then taking 7 from the right list, and finally taking 8 and 10 from the left list.

Chapter 10

Chapter 10 Exercises (Miscellaneous Topics II)

1. Ask the user to enter a course code, like MATH 247 or CMSCI 125. The code consists of an alphabetic department code followed by a space and then a course number. If the course number is not in the range from 100 to 499, then output that the user's entry is invalid.

2. Write a program that asks the user to enter several numbers separated by semicolons. Then print out the sum of all the numbers.

3. Write a program that asks the user to enter a number (an integer or a float) like `1,000,000.123` that uses commas to clarify the groups of the number. Print out the square root of that number.

4. Write a program uses a loop to create a list comprised of the strings `'a1'`, `'b2'`, `'c3'`, up through `'z26'`. Then print the list.

5. Write a program that asks the user to enter a sentence that contains some positive integers. Then write a program that reads through the string and replaces each number n in the sentence with $n+1$. For instance:

   input: `I bought 4 apples and 17 oranges.` input: `My favorite number is 8.`
   output: `I bought 5 apples and 18 oranges.` output: `My favorite number is 9.`

6. Write a program that loops over the integers from 100 to 120. For each number, randomly rearrange its digits integer to create a new number and print the new number along with the original. [Hint: Use `str()`, `list()`, `shuffle()` and `join()`.]

7. Write a program that asks the user for a length and then generates a random password of that length. The password's characters can be lowercase and uppercase letters, digits, and any of these special characters: `!@#$%^&*()`. The password is required to have at least one uppercase letter, at least one lowercase letter, at least one digit, and at least one of those special characters.

8. Write a program that randomly prints 10 valid dates. Do this in two different ways:

   (a) Randomly choose a month and then randomly choose a valid day in that month. Do this 10 times. It is possible that a date might repeat.

   (b) Create a list that contains all 365 days of the year, then use the `sample()` function or shuffle the list and take the first 10 things. This guarantees there will be no repeats.

9. Write a program that generates a list of 10 random multiplication problems, which are strings of a form like `2 x 9` or `10 x 7`, where the two numbers can range from 2 to 12. At the same time, generate a list of answers to those problems. Then loop through the list, asking the user to enter answers to the problems. Print out whether each answer is right or wrong, and at the end print out how many the user got correct.

10. Write a program that generates and prints a list of 20 random multiplication problems, each a string like `'2 x 11'`, with the property that the random numbers being multiplied can range from 2 through 12, but can't be 10. Do this in two ways:

    (a) Use `randint(2,12)` to generate the random numbers and use a while loop to keep generating until a number that is not 10 is generated.

    (b) Use the `choice()` function on an appropriate list.

11. Write a program that asks the user to enter a date in the format month/day. The program should print out whether the date is real. For instance, `14/10` is not a real date, and neither is `2/30`. Do this by using a list that contains the number of days in each month.

12. Write a program that asks the user for an angle in degrees-minutes-seconds form, such as `49d33'22''` or `153d4'22''`, where the input will starts with the number of degrees (0 to 360), followed by the letter `d`, followed by the number of minutes (0 to 60), followed by an apostrophe, followed by the number of seconds (0 to 60), followed by two apostrophes.

    The program should convert that angle to decimal form, rounded to two decimal places. For instance, `49d33'22` should become 49.56 and `153d4'22''` should become 153.07. To convert from degrees/minutes/seconds to the decimal form, use $d + \frac{m}{60} + \frac{s}{3600}$, where d, m, and s are the numbers of degrees, minutes, and seconds, respectively.

13. Write a program that asks the user to enter a list of costs of products. Then add a 6% tax to every cost and print out the new costs, each on separate lines, right-justified and displayed to exactly two decimal places (e.g., 2.4 should be 2.40). For the right-justification, assume costs will never be greater than $999.

14. Write a program that prints out a table of the sines and cosines of the angles from 0 to 360 going up in increments of 30 degrees. The angles should be right justified. The sines and cosines should be displayed to exactly 3 decimal places. There should be several spaces between the angles, the sines, and the cosines. All the decimal points should be lined up. The start of the output is shown below. [Hint: the sine and cosine functions operate on radians, not degrees, so use the `radians` function from the `math` module to convert first.]

```
  0    0.0000   1.0000
 30    0.5000   0.8660
```

15. Here are some lists to use with this problem:

```
Names = ['Alice', 'Bob', 'Caroline']
Birthdays = ['2/7/86', '11/12/66', '8/17/72']
Salaries = [72000, 144300, 43200]
```

    Read through all these lists and print out all the information for each person on the same line with the 12 spots allocated for the name, with the birthdate reformatted so that there is a leading zeroes for single digit months and days, and format the salaries with commas in the numbers as separators. For both adding zeroes and commas, use the appropriate formatting code instead of string operations.

16. Write a program that uses nested loops to produce a 9×9 table with entries of the form (i, j) where i and j run from 1 to 9. The output is partially shown below.

```
(1,1) (1,2) ... (1,9)
(2,1) (2,2) ... (2,9)
...
(9,1) (9,2) ... (9,9)
```

17. (a) Modify the program above so that it does not show anything above the diagonal; that is, none of the items (i, j) with j greater than i are shown.

    (b) Modify the program above so that it does not show anything below the diagonal; that is, none of the items (i, j) with i greater than j are shown.

18. Write a program that uses nested loops to print out all integer solutions to the equation $37x+14y = 11$ with x and y both between -100 and 100.

19. Write a program that asks the user to enter a list of integers. Then use a while loop to read through the list and print the first index at which a repeat occurs and stop the loop. If there is no repeat in the list, then print a message indicating that. [Hint: Use the short-circuiting of the `and` operator to avoid an index out of bounds error.]

20. Write a program that creates and prints a 6×6 list of random integers from 0 to 3. Then ask the user to enter a location in the list and print out the sum of all the neighbors of the value at that location. The neighbors of a cell are the up to eight cells that touch it horizontally, vertically, and diagonally. Be careful because cells at the edge of the list won't have all eight neighbors.

Chapter 11

Chapter 11 Exercises (Dictionaries)

1. Create a dictionary whose keys are the strings `'abc'`, `'def'`, `'ghi'`, `'jkl'`, and `'mno'` and whose corresponding values are 7, 11, 13, 17, and 19. Then write dictionary code that does the following:

   (a) Print the value in the dictionary associated with the key `'def'`.

   (b) Use the `keys()` method to print out all the keys.

   (c) Loop over the dictionary and print out all the keys and their associated values.

   (d) Use an if statement to check if the dictionary contains the key `'pqr'` and print out an appropriate message indicating whether it does or doesn't.

   (e) Change the value associated with the key `'abc'` to 23 and then print out all the values in the dictionary using the `values()` method.

2. Write a program that creates a dictionary for the first 10 items of the periodic table. The keys should be the element symbols `H`, `He`, `Li`, `Be`, `B`, `C`, `N`, `O`, `F`, `Ne`. The values should be the names of the elements: Hydrogen, Helium, Lithium, Beryllium, Boron, Carbon, Nitrogen, Oxygen, Fluorine, and Neon. Then ask the user to enter an abbreviation and print out its name.

3. To convert from inches to centimeters, the conversion is to multiply by 2.54. To convert from inches to meters, the conversion is to multiply by .0254. To convert from inches to feet, the conversion is to multiply by 1/12. To convert from inches to yards, the conversion is to multiply by 1/36. Write program that creates a dictionary whose keys are `cm`, `m`, `ft` and `yd` and whose values are these conversion factors. Then ask the user to enter a value in inches and the unit (abbreviation) that they want to convert to. Use the dictionary to do the conversion and print out the result.

4. Write a program that repeatedly asks the user to enter a word and create a dictionary whose keys are those words and whose values are counts of how many vowels are in the words. The user indicates they are done by pressing enter without entering anything (the input will therefore be an empty string). Print out the dictionary after it is done being created.

5. Write a program that uses a loop to create a dictionary whose keys are all the capital and lowercase letters and whose values are the ASCII codes of those letters. The ASCII code of a character `c` can be gotten from calling `ord(c)` using the built-in `ord()` function. When the dictionary is done being created, print it out.

6. Write a program that asks the user to enter a string (consisting of any characters). Then create and print a dictionary from that string whose keys are the characters of the string and whose values are how many times those characters appear in the string.

7. Write a program that asks the user for two strings. Then use maps like in the previous problem to print out whether the words contains the same characters with the same frequencies, though possibly in different orders. For instance, `AABCCC` and `CCCABA` both contain two As, a B, and three Cs, so the program would print out `Yes` for these two strings. But it would print out `No` for the strings `AAABC` and `ABC`, since the frequencies don't match up.

8. Write a program that asks the user to enter a date in the format of a month name followed by a day number. The program should print out whether the date is real. For instance, January 32 is not a real date, and neither is February 30. Assume that the user enters the full month name and that the month name is valid. To validate the day number, use a dictionary that contains the number of days in each month.

9. Write a program that asks the user to enter a person's name and then a list of course numbers that that person has taken. Do this five times. Create a map from this where the keys are the names and the values are the lists of courses that person has taken. Then ask the user for a course name and use the map to print out all the people who took that course.

10. Write a program that creates a dictionary whose keys are questions and whose corresponding values are answers to those questions. Then use that dictionary to play a simple quiz game, where the quiz questions are given in a random order. For each question, ask the player for an answer and tell them whether they got it right or not.

11. One use for dictionaries is something called a *sparse array*. These are arrays (lists) of numbers, where most of the entries are 0. To use a regular list when most of the entries are 0 would waste space. So instead we store only those values that are not 0 and we use a dictionary to do so. The keys are the list indices and the values are the values at that index. For instance, the list `[0,0,27,0,0,0,0,15,0,0,...]` would be stored with the dictionary `{2:27,7:15}`. Write a program that asks the user for three indices and three values to store in the sparse array at those indices. Then ask them to enter a index and print out the value at that index (which will be either a value from the dictionary or 0 if the index is not a key in the dictionary).

12. The substitution cipher is a simple way of encrypting a message, where each `a` is replaced with a certain letter, say `f`, each `b` is replaced with a certain letter, say `i`. The key for the cipher is a string indicating the replacements. For instance, `fiepltdkjbsuywqxanczrmvhg` says that each `a` is replaces with `f`, each `b` is replaced with `i`, each `c` is replaced with `e`, etc. We can generate a key by shuffling the letters of the alphabet. Then given a word, we can encode it with the substitution cipher using a dictionary whose keys are the letters of the alphabet and whose values are the letters they are mapped to in the key. A similar, but reversed, dictionary can be used for decryption.

    Write program that generates and prints a random key and uses that key to create a dictionary like the one described above. Then ask the user for a lowercase word. Encrypt and then decrypt that word using the substitution cipher dictionary and print out the encrypted and decrypted result.

Chapter 12

Chapter 12 Exercises (Text Files)

1. The file `expenses.txt` has a bunch of expenses, each an integer, one on each line. Write a program that reads the file and prints out how many expenses are over $2000. Print all of them on the same line.
2. The file `expenses.txt` has a bunch of expenses, each an integer, one on each line. Write a program that reads the file, applies a 10% tax to each expense, and outputs the new expenses in a file called `new_expenses.txt` with one expense on each line. Make sure the expenses are formatted like money typically is, with each value given to exactly two decimal places.
3. The file `secret_message.txt` has a secret message encoded into it. Specifically the first letters of each line, when taken all together, spell out the secret message. Write a program that reads the file and prints out the secret message.
4. Use `wordlist.txt` to do the following.
   (a) All words that contain the string `quot`.
   (b) The percentage of words that contain all the vowels (not including *y*).
   (c) The average word length.
   (d) All words of at least six letters that start and end with the same three letters.
   (e) The longest word that starts and ends with the same letter.
   (f) All words that contain four alphabetically consecutive letters (like abc, bcd, cde, def, . . .).
5. Write a program that reads through `wordlist.txt` and produces a new file `new_wordlist.txt` that contains all the words from `wordlist.txt` that are 5 or less letters long.
6. The files `first_names.txt` and `last_names.txt` are lists of common first and last names. Write a program that uses these files to print out 10 random full names, where a full name consists of a first and last name separated by space, like `Don Knuth`.
7. Each line of the file `population.txt` contains a country name and its population, separated by a tab. Write a program that uses the file to print out all the countries whose names start with *G* that have at least 500,000 people.
8. Write a program that reads a text file named `studentParagraphs.txt`. The file contains several paragraphs of text, each paragraph separated by two newline characters. At the start of each paragraph is a student name followed by a colon. The program should print out lines of the form *student name: grade*, where the grade for each student is based on the length of their paragraph. Here are the rules for grades:

length in characters	grade
200 or more	A
150-199	B
100-149	C
50-99	D
49 or less	F

9. Each line of the file `elements.txt` has an element number, its symbol, and its name, each separated by a space, followed by a hyphen and then another space. Write a program that reads this file and outputs a new file `new_elements.txt`, where each line of the file contains just the element symbol and number, separated by a comma, like below.

   ```
   H,1
   He,2
   Li,3
   ...
   ```

10. You are given a file called `baseball.txt`. A typical line of the file starts like below.

    ```
    Ichiro Suzuki     SEA     162     680     74   ...[more stats]
    ```

    Each entry is separated by a tab, `\t`. Splitting these entries at tabs yields a list where the first entry is the player's name, the second is their team, the eighth is the number of home runs, and the 14th is the average. Print out the name, team, home runs, and average of all the players that hit at least 20 home runs and have an average of at least .300.

11. The file `high_temperatures.txt` has the high temperature for each date of the year in a certain city. Write a program that reads the file and writes a new file `new_high_temperatures.txt`, where the temperatures are converted from Fahrenheit to Celsius and rounded to the nearest whole number, and each date is converted from the `month/day` format to give the full month name. For instance, the first line, which is `1/1 37`, should become `January 1 3`.

12. The file `countries.txt` has country names, one name on each line. Write a program that reads this file and creates a list of lists, where the first item in the list is a list of all the countries that start with *A*, the second item is a list of all the countries that start with *B*, etc. There should be 26 lists in total, some of which may be empty. Print the list.

13. Ask the user to enter a string of lowercase letters. Then use `wordlist.txt` to find all the words that can be made from the letters of the user's string, taking into account frequencies. [Hint: Creating a dictionary of letter frequencies might help.]

14. The file `countries.txt` contains the names of countries, one on each line. Ask the user for a country name and an integer n. If the country name is not in the file, print a message. Otherwise, print the country name that comes n lines before the user's country in the file, wrapping around to the back of the list if needed. [Hint: use a mod to wrap around.]

15. The files `football_teams.txt` and `basketball_teams.txt` contain all of the NFL and NBA team names. Read these files and use them to answer this question: What NFL team's nickname, when you remove a single letter, gives an NBA team's nickname? [Hint: systematically remove letters from NFL teams' nicknames to find the answer.]

16. The file `romeoandjuliet.txt` that contains the text of a well-known Shakespearean play. Write a program that reads through the file and creates a dictionary of the whose keys are the words of the file and whose values are the number of times those words occur. For the most accurate results, convert everything to lowercase and remove all punctuation except apostrophes and hyphens. Then use `sorted(list(d.items()),key=lambda x:-x[1])[:10]` to print out the ten most common words (replace `d` with whatever you call your dictionary.)

17. One way to save the state of a game is to write the values of all the important variables to a file. Write a simple guess-a-number game, where the computer picks random numbers from 1 to 5, and the user has one try to guess the number. After each guess, the program should print out whether they got it right or not and print out the total number right and wrong. Also after each guess, the user has the option to save their progress and quit or to keep playing. To save their progress, write the amount right and wrong to a file called `guess_a_number.txt`. When the player starts the game, they should have the option to start a new game or continue an old one. If they start a new game, the number right and wrong should start at 0 and otherwise, they should be read from the file.

18. The file `continents.txt` contains a list of country names arranged by continent. Please look at the file to see how things are arranged. Using this file, write a quiz program that randomly selects a country and asks the user to guess what continent the country is on. Print out a message indicating whether or not they guessed correctly.

19. The file `multiple_choice.txt` contains multiple choice questions. Write a program that reads that file and create a new file, `new_multiple_choice.txt`, with the same questions but with the choices for each question reordered.For instance, shown on the left is what a typical question would look like and on the right is what it might look like after reordering the choices for each question.

    ```
    2. After Asia, the largest continent is     2. After Asia, the largest continent is
    (a) Africa                                  (a) North America
    (b) North America                           (b) Africa
    (c) South America                           (c) South America
    ```

    Note that some questions may have more choices than others, and assume that there is no limit on the number of questions in the file. Questions are separated from each other by a blank line.

20. The file `high_temperatures.txt` contains the average high temperatures for each day of the year in a certain city. Each line of the file consists of the date, written in the month/day format, followed by a space and the average high temperature for that date. Find all of the 30-day periods over which there is the biggest increase in the average high temperature (there are actually a few 30-day periods with the same maximum increase). [Hint: a mod might be helpful to consider the case of a 30-day period that starts in December and ends in January.]

21. The file `nfl1978-2013.csv` contains the results of every regular-season NFL game from 1978 to 2013. Open the file to see how it is arranged. Allow the user to enter two team names (you can assume they enter the names correctly) and have the program print out all of the games involving the two teams where one of the teams was shut out (scored no points).

22. The file `nfl_scores.txt` contains information on how many times each final score has happened in an NFL game. Please look at the file in a text editor to see how things are arranged. Using the file, print out all the final scores that have *never* happened, with the limitation that both teams' total points must be 0 or in the range from 2 to 29. For instance, the scores 4-2, 11-9, and 6-5. 20-1 and 77-55 have also never happened, but those fall outside of the specified range, so they wouldn't be printed.

Chapter 13

Chapter 13 Exercises (Functions)

1. Write a program called `triangular()` that takes an integer `n` and returns the nth triangular number. The nth triangular number is given by the formula $n(n+1)/2$.

2. Write a function called `get_bill_total()` that takes a bill amount and a sales tax percentage and returns the total bill amount, rounded to two decimal places.

3. Write a function called `distance()` that takes four numbers x_1, y_1, x_2, and y_2 and returns $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$.

4. Write a function called `print_blank_lines()` that takes an integer and prints out that many blank lines. It shouldn't return anything.

5. Write a function called `print_perfect_squares()` that takes an integer and prints out all the perfect squares from 1 up through that number. It shouldn't return anything.

6. Write a function called `add_up_range()` that takes a starting location, ending location and step and adds up all the integers in that range. For instance, `add_up_range(2,6,1)` would add up the integers $2+3+4+5$ and return 14. Have the three arguments behave just like they do in Python's **`range`** statement.

7. The Euler ϕ function, $\phi(n)$, tells how many positive integers less than or equal to n are relatively prime to n. (Two numbers are said to be relatively prime if they have no factors in common, i.e. if their gcd is 1). Write a function called $phi()$ that implements the Euler ϕ function. [Hint: the `fractions` module has a function called `gcd()` that might be useful.]

8. Below is described how to find the date of Easter in any year. Write a function called `easter()` that takes year and returns a string with the date of Easter in that year. Despite its intimidating appearance, this is not a hard problem. Note that $\lfloor x \rfloor$ is the *floor* function, which for positive numbers just drops the decimal part of the number. In Python $\lfloor x/y \rfloor$ is `x // y`.

   C = century (1900s $\rightarrow C = 19$)

   Y = year (all four digits)

   $m = (15 + C - \lfloor \frac{C}{4} \rfloor - \lfloor \frac{8C+13}{25} \rfloor) \bmod 30$

   $n = (4 + C - \lfloor \frac{C}{4} \rfloor) \bmod 7$

   $a = Y \bmod 4$

   $b = Y \bmod 7$

   $c = Y \bmod 19$

   $d = (19c + m) \bmod 30$

   $e = (2a + 4b + 6d + n) \bmod 7$

Easter is either March $(22 + d + e)$ or April $(d + e - 9)$. There is an exception if $d = 29$ and $e = 6$. In this case, Easter falls one week earlier on April 19. There is another exception if $d = 28$, $e = 6$, and $m = 2, 5, 10, 13, 16, 21, 24$, or 39. In this case, Easter falls one week earlier on April 18.

9. Write a function called `is_upper()` that takes a string of letters and returns whether or not every letter of the string is uppercase.

10. Write a function called `rand_lower_string()` that takes an integer parameter and returns a random string of that many lowercase letters.

11. Write a function called `contains_a_letter()` that takes a string and returns **`True`** if the string contains any letters (upper or lowercase) and **`False`** otherwise.

12. Write a function called `simplify_text()` that is given a string and converts the string to all lowercase, removes all common punctuation except for hyphens and apostrophes, and returns the new string.

13. On Mars, residents have a 25-hour day, where each hour is 60 minutes long. Northern hemisphere residents use a system where hours are called 0, 1, . . . 24. Southern hemisphere residents label their hours as 1A, 2A, 3A, 4A, 5A, 1B, 2B, . . . , 5E, where the day is broken up into 5 equal portions (morning (A), afternoon (B), evening (C), night (D), late night (E)). Write a function called `convert_time()` that takes a string `time` in northern hemisphere time and converts it into southern hemisphere time. Input times will have hours and minutes. As examples, `3:30` converts to `4A:30` and `19:45` converts to `5D:45`.

14. Write a function called `next_name()` that returns the next name in a sequence, where the names in the sequence are *A*, *B*, . . . , *Z*, *AA*, *AB*, . . . , *ZZ*, *AAA*, *AAB* . . . . For instance, `next_name('S')` returns `'T'`, `next_name('CD')` returns `'CE'`, `next_name('AZ')` returns `'BA'`, and `next_name('ZZZ')` returns `'AAAA'`.

15. Write a function called `reverse_only_letters()` that takes a string and creates returns a new string where all the positions of all the letters have been reversed, but all other characters have been left in position. For instance, if the argument is `ab*c&&de` then the returned string would be `ed*c&&ba`.

16. Write a function called `average()` that takes a list of numbers and returns the average of those numbers.

17. Write a function called `mid_range()` that takes an integer list and returns $(x + y)/2$, where x is the largest value in the list and `y` is the smallest value.

18. Write a function called `closest()` that takes a list of integers and another integer n and returns the integer in the list that is the closest to n. If two integers are tied for the closest, then return the smaller of the two.

19. Write a function called `changes_by_one()` that takes a list of integers and integers and returns a list of all the indices at which the current value is equal to 1 more than the previous value. For instance, if the list is `[1,2,5,5,10,11,12,15,16]`, it would return `[1,5,6,8]`.

20. Write a function called `string_sort()` that takes a string of lowercase letters and returns a string with all the same characters, but ordered alphabetically. [Hint: Convert the string to a list and use the list `sort()` method.]

21. On telephones, digits correspond to letters, according to the table below. Write a function called `telephone_match()` that determines if a numerical phone number `num` matches the string `s` according to the table below. For instance, if `num` is `468-3437` and `s` is `INTEGER`, then the function would return `true` because each letter in `s` matches the corresponding number in `num`.

2	ABC
3	DEF
4	GHI
5	JKL
6	MNO
7	PQRS
8	TUV
9	WXYZ

22. Write a function called `integer_lengths()` that takes a list of integers and returns the sum of all their lengths. For instance, given `[2,44,532,3,44,22]`, the sum of the lengths is $1+2+3+1+2+2=11$.

23. Write a function called `union()` that takes two lists and returns a new list that contains all the items that are in one, the other, or both of the lists. Make it so that the returned list contains no repeated copies of elements.

24. Write a function called `teams()` that takes an argument which is a list of names and prints out a randomized grouping of the names into teams of two. You can assume that there are an even number of names. The function shouldn't return anything. Here is sample output for the list `[A,B, C, D, E, F, G, H, I, J]`:

```
Team 1: D, J
Team 2: I, A
Team 3: B, C
Team 4: E, G
Team 5: F, H
```

25. Write a function called `triple_shuffle()` that takes three lists of the same size and shuffles them concurrently, so that they are shuffled in the exact same way. For instance, if the lists are `[1,2,3,4,5]`, `[11,22,33,44,55]`, and `[9,8,7,6,5]`, and the first list is shuffled into `[4,5,3,2,1]`, then the second would become `[44,55,33,22,11]` and the third would become `[6,5,7,8,9]`. The function should modify the lists and not return anything.

26. Write a function called `print_2d()` that takes a 2-dimensional list and prints the list nicely, with each row being printed on its own line.

27. Write a function called `random_2d_list()` that takes integers, m, n, x, and y and creates an $m \times n$ 2d-list whose elements are random numbers from x to y.

28. Write a function called `number_of_lines()` that takes a file name (a string) and returns how many lines are in the file.

29. Write a function called `find_words()` that takes a string given a string `s` of consonants, and returns a list of all the English words (from `wordlist.txt`) that have those consonants in that order, when vowels (a, e, i, o, u) are excluded. There should be no other consonants in the word besides the ones in the input string. For instance, given `bt`, it would return `bat`, `bit`, `boat`, and `abate`, among other words.

30. Write a function called `words_in_word()` that takes a string as a parameter and returns a list of all groups of contiguous letters in that word that are real English words (using the file `wordlist.txt` to tell if a word is real). For instance, if `word` is `restaurant`, then the list returned should be
`[a,an,ant,aura,es,ran,rant,re,rest,restaurant,ta,tau]`.

Chapter 14

Chapter 14 Exercises (Object-Oriented Programming)

1. Write a class called `BankAccount` that has the following:
   - A field called `name` that stores the name of the account holder.
   - A field called `amount` that stores the amount of money in the account.
   - A field called `interest_rate` that stores the account's interest rate (as a percentage).
   - A constructor that just sets the values of the three fields above.
   - A method called `apply_interest()` that takes no arguments and applies the interest to the account. It should just modify the `amount` field and not return anything. For instance, if the account has $1000 in it and the interest rate is 3%, then the `amount` variable should be changed to $1030 ($1000 + 3% interest).

   Then test the class, by creating a new `BankAccount` object for a user named Juan De Hattatime who has $1000 at 3% interest. Then do the following:
   - Use the `apply_interest()` method to apply the interest to the account.
   - Print out how much money is now in the account after applying the interest.
   - Change the account's interest rate to 2%.
   - Use the `apply_interest()` method to apply the interest to the account again.
   - Print out how much money is now in the account after applying the interest again.

2. Write a class called `Item` that represents an item for sale. It should have the following:
   - Fields representing the name and price of the item
   - A constructor that sets those fields,
   - A `__str__()` method that returns a string containing the item name and price, with the price formatted to exactly 2 decimal places

   Test the class by creating a new item object and printing it out.

3. Write a class called `ShoppingCart` that might be used in an online store. It should have the following:
   - A list of `Item` objects that represents the items in the shopping cart
   - A constructor that creates an empty list of items (the constructor should take no arguments except `self`)
   - A method called `add()` that takes a name and a price and adds an `Item` object with that name and price to the shopping cart
   - A method called `total()` that takes no arguments and returns the total cost of the items in the cart

- A method called `remove_items()` that takes an item name (a string) and removes any `Item` objects with that name from the shopping cart. It shouldn't return anything.
- A `__str__()` method that returns a string containing info on all the items in the shopping cart

Then test out the shopping cart as follows: (1) create a shopping cart; (2) add several items to it; (3) print the cart's total cost (using the `total()` method); (4) remove one of the items types; (5) print out the cart.

4. Write a class called `RestaurantCheck`. It should have the following:

   - Fields called `check_number`, `sales_tax_percent`, `subtotal`, `table_number`, and `server_name` representing an identification for the check, the bill without tax added, the sales tax percentage, the table number, and the name of the server.
   - A constructor that sets the values of all four fields
   - A method called `calculate_total` that takes no arguments (besides `self`) and returns the total bill including sales tax.
   - A method called `print_check` that writes to a file called `check###.txt`, where `###` is the check number and writes information about the check to that file, formatted like below:

   ```
   Check Number: 443
   Sales tax: 6.0%
   Subtotal: $23.14
   Total: $24.53
   Table Number: 17
   Server: Sonic the Hedgehog
   ```

   Test the class by creating a `RestaurantCheck` object and calling the `print_check()` method.

5. Write a class called `Ticket` that has the following:

   - A field `cost` for the price of the ticket and a string field `time` for the start time of the event (assume times are in 24-hour format, like `'18:35'`)
   - A constructor that sets those variables
   - A `__str__()` method that returns a string of the form `Ticket(<cost>, <time>)`, where `<cost>` and `<time>` are replaced with the values of the `cost` and `time` fields.
   - A method called `is_evening_time()` that returns **`True`** or **`False`**, depending on whether the time falls in the range from 18:00 to 23:59.
   - A method called `bulk_discount()` that takes an integer n and returns the discount for buying n tickets. There should be a 10% discount for buying 5 to 9 tickets, and a 20% discount for buying 10 or more. Otherwise, there is no discount. Return these percentages as integers.

   Test the class by creating a `Ticket` item, printing it, calling the `is_evening_time()` method, and calling the `bulk_discount()` method.

6. Write a class called `MovieTicket` that inherits from the `Ticket` class of the previous problem. It should have the following (in addition to all that it gets from the `Ticket` class):

   - A string field called `movie_name`
   - A constructor that sets `movie_name` as well as `cost` and `time`
   - Override the `__str__()` method so that it returns a string of the form `MovieTicket(<cost>, <time>, <nam`
     where `<cost>`, `<time>`, and `<name>` are replaced with the values of the class's fields.
   - A method called `afternoon_discount()` that returns a discount of 10 (standing for 10%) if the ticket time falls in the range from 12:00 to 17:59 and 0 otherwise.

   Test the class by creating a `MovieTicket` item, printing it, and calling the `afternoon_discount()` and `is_evening_time()` methods.

7. Write a class called `Course` that has the following:

- A field `name` that is the name of the course, a field `capacity` that is the maximum number of students allowed in the course, and a list called `student_IDs` representing the students in the course by their ID numbers (stored as strings).
- A constructor that takes the name of the course and capacity and sets those fields accordingly. The constructor should also initialize `student_IDs` list to an empty list, but it should not take a list as a parameter. It should only have the course name and capacity as parameters.
- A method called `is_full()` that takes no arguments and returns **`True`** or **`False`** based on whether or not the course is full (i.e. if the number of students in the course is equal to or above the capacity).
- A method called `add_student()` that takes a student ID number and adds the student to the course by putting their ID number into the list. If the student is already in the course, they must *not* be added to the list, and if the course is full, the student must *not* be added to the course.

Test the class by creating a `Course` object, adding several students to the class, and calling the `is_full()` method. Print out the value of the `student_IDs` field to make sure everything comes out as expected.

8. Write a class called `Avatar` that represents a character in a role-playing game. The class should have the following:

   - Fields called `name`, `hit_points`, `attack_power`, and `defense_power`
   - A constructor that sets those fields
   - A method called `attack()` that returns a random integer from 1 though the value of `attack_power`
   - A method called `defend()` that takes an integer representing an attack amount and decreases `hit_points` by an amount equal to the difference of the attack amount and `defense_power` (using 0 if that comes out negative). Return the amount of damage taken.
   - A method called `is_alive()` that returns **`True`** or **`False`**, depending on whether `hit_points` is greater than 0 or not

   Then test the class by creating a program that creates containing at least three `Avatar` objects. Then repeatedly have two different random avatars be chosen and have one of them attack the other. Print out which avatars are chosen and the result of the attack. Whenever an avatar is no longer alive, a message should be printed and that avatar should be removed from the list. The program stops when there is only one avatar left. To slow down the rate at which things happen, you might want to import `time` and use `time.sleep(.5)` to pause the program for .5 seconds after each attack.

9. Write classes called `Fighter` and `Mage` that inherits from the `Avatar` class. The `Fighter` class should have an additional method called `special_power()` that with a probability of 1 in 5, returns an attack amount equal to twice the value of `attack_power` and otherwise returns a value of 0. The `Mage` class should have a method also called `special_power()` that adds 10 to the value of `hit_points`.

   Then use these two classes to create a game. One player is a Fighter and one is a Mage. Players take turns until one player is no longer alive. On each turn, the player can choose between attacking and using their special power. Show the result of each attack and continue the game until one player is defeated.

10. Write a class called `Timer` used to time things. It should have the following:

    - A field `initial_time` that holds information about the time.
    - A method `start()` that starts the timing by saving the value of `time.time()` to the `initial_time` field. It takes no parameters and returns nothing. You will need to import the `time` module to do this.
    - A method `elapsed_seconds()` that returns the amount of time in seconds that has passed since the timer was started.
    - A method `formatted_time()` that takes a time in seconds, rounds to the nearest whole second, and converts it to minutes and seconds, and returns that in a string with the minutes and seconds separated by a colon. For instance, `formatted_time(131)` would return `'2:11'`.

    Then test your class by doing the following:

- Create a timer and start it.
- Ask the user to type something.
- Print out the amount of time in seconds that took, using `elapsed_Seconds()`
- Then create another timer and start it.
- Ask the user to type some more text.
- Print out the amount of time in seconds that took using `elapsed_seconds()`
- Print out the total amount of time passed since the first timer was started, using `elapsed_seconds()` and formatting it with `formatted_time()`.

11. Write a class called `Calendar` with the following:

    - a field called `year` and a constructor that sets the year
    - a boolean method `is_leap_year()` that returns whether the year is a leap year. Leap years are years that are divisible by 4, except that years that are also divisible by 100 are not leap years unless they are divisible by 400.
    - a method `first_day(m)` that returns the day of the week of the first day of the month `m` (given as a number from 1 to 12) in the given year. To get the first day use the formula below, where y, m, and d are the year, month, and day. The result of the computation, as well as the returned value, is an integer from 0 to 6, with 0 meaning Sunday, 1 meaning Monday, etc.

$$\begin{aligned}
p &= \left\lfloor \frac{14-m}{12} \right\rfloor \\
q &= y - p \\
r &= q + \left\lfloor \frac{q}{4} \right\rfloor - \left\lfloor \frac{q}{100} \right\rfloor + \left\lfloor \frac{q}{400} \right\rfloor \\
s &= m + 12p - 2 \\
t &= \left(1 + r + \left\lfloor \frac{31s}{12} \right\rfloor\right) \bmod 7
\end{aligned}$$

    An expression like $\lfloor x/y \rfloor$ is done by `x // y`.

    - a method `print_calendar(m)` that prints out a calendar for that month (given as an integer from 1 to 12) in the given year. The calendar should look something like the one below, which would be for this month:

```
         1    2    3    4    5    6
   7     8    9   10   11   12   13
  14    15   16   17   18   19   20
  21    22   23   24   25   26   27
  28    29   30
```

    Test your class by creating a calendar object and calling all of its methods.