The exercises that appear in this chapter should all be completed using loops. In some cases the exercise specifies what type of loop to use. In other cases you must make this decision yourself. Some of the exercises can be completed easily with both for loops and while loops. Other exercises are much better suited to one type of loop than the other. In addition, some of the exercises require multiple loops. When multiple loops are involved, one loop might need to be nested inside the other. Carefully consider your choice of loops as you design your solution to each problem.

1. **Average**: In this exercise you will create a program that computes the average of a collection of values entered by the user. The user will enter 0 as a sentinel value to indicate that no further values will be provided. Your program should display an appropriate error message if the first value entered by the user is 0.

   > Hint: Read the values in a loop and sum all the values in a variable and keep another variable as the counter. Because the 0 marks the end of the input it should not be included in the average.

2. **Temperature Conversion Table**: Write a program that displays a temperature conversion table for degrees Celsius and degrees Fahrenheit. The table should include rows for all temperatures between 0 and 100 degrees Celsius that are multiples of 10 degrees Celsius. Include appropriate headings on your columns. The formula for converting between degrees Celsius and degrees Fahrenheit can be found on the internet.
3. **Compute the Perimeter of a Polygon**: Write a program that computes the perimeter of a polygon. Begin by reading the number of sides and then read the length of each side using a loop. Display the perimeter at the end.
4. **Admission Price**: A particular zoo determines the price of admission based on the age of the guest. Guests 2 years of age and less are admitted without charge. Children between 3 and 12 years of age cost $14.00. Seniors aged 65 years and over cost $18.00. Admission for all other guests is $23.00. Create a program that begins by reading the ages of all of the guests in a group from the user, with one age entered on each line. The user will enter a blank line to indicate that there are no more guests in the group. Then your program should display the admission cost for the group with an appropriate message. The cost should be displayed using two decimal places.
5. **Approximate π**: The value of π can be approximated by the following infinite series:

   $$\pi \approx 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \frac{4}{10 \times 11 \times 12} - \cdots$$

   Write a program that displays 15 approximations of π. The first approximation should make use of only the first term from the infinite series. Each additional approximation displayed by your program should include one more term in the series, making it a better approximation of π than any of the approximations displayed previously.

6. **Multiplication Table**: In this exercise you will create a program that displays a multiplication table that shows the products of all combinations of integers from 1 times 1 up to and including 10 times 10. Your multiplication table should include a row of labels across the top of it containing the numbers 1 through 10. It should also include labels down the left side consisting of the numbers 1 through 10. The expected output from the program is shown below:

```
    1   2   3   4   5   6   7   8   9  10
1   1   2   3   4   5   6   7   8   9  10
2   2   4   6   8  10  12  14  16  18  20
3   3   6   9  12  15  18  21  24  27  30
4   4   8  12  16  20  24  28  32  36  40
5   5  10  15  20  25  30  35  40  45  50
6   6  12  18  24  30  36  42  48  54  60
7   7  14  21  28  35  42  49  56  63  70
8   8  16  24  32  40  48  56  64  72  80
9   9  18  27  36  45  54  63  72  81  90
10 10  20  30  40  50  60  70  80  90 100
```

When completing this exercise you will probably find it helpful to be able to print out a value without moving down to the next line. This can be accomplished by added `end=""` as the last parameter to your print statement. For example, `print("A")` will display the letter A and then move down to the next line. The statement `print("A", end="")` will display the letter A without moving down to the next line, causing the next print statement to display its result on the same line as the letter A.

7. **Greatest Common Divisor**: The greatest common divisor of two positive integers, *n* and *m*, is the largest number, *d*, which divides evenly into both *n* and *m*. There are several algorithms that can be used to solve this problem, including:

   Initialize *d* to the smaller of *m* and *n*.
   While *d* does not evenly divide *m* or *d* does not evenly divide *n* **do**
      Decrease the value of *d* by 1
   Report *d* as the greatest common divisor of *n* and *m*

   Write a program that reads two positive integers from the user and uses this algorithm to determine and report their greatest common divisor.

8. **Prime Factors**: The prime factorization of an integer, n, can be determined using the following steps:

   Initialize *factor* to two
   While *factor* is less than or equal to *n* **do**
      If *n* is evenly divisible by *factor* **then**
         Conclude that *factor* is a factor of *n*
         Divide *n* by *factor* using integer division
      Else
         Increase *factor* by one

   Write a program that reads an integer from the user. If the value entered by the user is less than 2 then your program should display an appropriate error message. Otherwise your program should display the prime numbers that can be multiplied together to compute n, with one factor appearing on each line. For example:

   ```
   Enter an integer (2 or greater): 72
   The prime factors of 72 are:
   2
   2
   2
   3
   3
   ```

9. **Maximum Integer**: This exercise examines the process of identifying the maximum value in a collection of integers. Each of the integers will be randomly selected from the numbers between 1 and 100. The collection of integers may contain duplicate values, and some of the integers between 1 and 100 may not be present (as a hint here, this is how Python will generate integers, more at the end of the exercise).

Take a moment and think about how you would handle this problem on paper. Many people would check each integer in sequence and ask themselves if the number that they are currently considering is larger than the largest number that they have seen previously. If it is, then they forget the previous maximum number and remember the current number as the new maximum number. This is a reasonable approach, and will result in the correct answer when the process is performed carefully. If you were performing this task, how many times would you expect to need to update the maximum value and remember a new number?

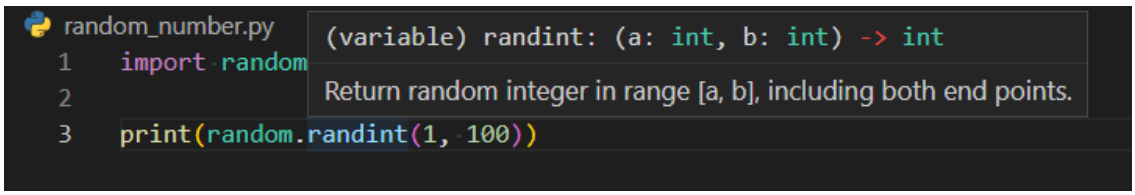
While we can answer the question posed at the end of the previous paragraph using probability theory, we are going to explore it by simulating the situation. Create a program that begins by selecting a random integer between 1 and 100. Save this integer as the maximum number encountered so far. After the initial integer has been selected, generate 99 additional random integers between 1 and 100. Check each integer as it is generated to see if it is larger than the maximum number encountered so far. If it is then your program should update the maximum number encountered and count the fact that you performed an update. Display each integer after you generate it. Include a notation with those integers which represent a new maximum.

After you have displayed 100 integers your program should display the maximum value encountered, along with the number of times the maximum value was updated during the process. Partial output for the program is shown below, with **...** representing the remaining integers that your program will display. Run your program several times. Is the number of updates performed on the maximum value what you expected?

```
30
74 <== Update
58
17
40
37
13
34
46
52
80 <== Update
37
97 <== Update
45
55
73
...

The maximum value found was 100
The maximum value was updated 5 times
```

> To generate random numbers between 1 and 100 you can use the following bit of code. **Remember that the import is necessary only once, at the beginning of the script**. The instruction that generates the random integer takes both end points as a closed interval, so both ends are taken in consideration. `random.randint` can be used multiple times in the program.
>
> ```
> import random
>
> foo = random.randint(1, 100)
> ```


random_number.py
(variable) randint: (a: int, b: int) -> int
Return random integer in range [a, b], including both end points.
1 import random
2
3 print(random.randint(1, 100))