1. **Run timing**[1]: System administrators often use Python to perform a variety of tasks, including producing reports from user inputs and files. It's not unusual to report how often a particular error message has occurred, or which IP addresses have accessed a server most recently, or which usernames are most likely to have incorrect passwords. Learning how to accumulate information over time and produce some basic reports (including average times) is thus useful and important. Moreover, knowing how to work with floating-point values, and the differences between them and integers, is important. For this exercise, then, we'll assume that you run 10 km each day as part of your exercise regime. You want to know how long, on average, that run takes. Write a program (run_timing) that asks how long it took for you to run 10 km. The program continues to ask how long (in minutes) it took for additional runs, until the user presses Enter. At that point, the program exits—but only after calculating and displaying the average time that the 10 km runs took. For example, here's what the output would look like if the user entered three data points:

   ```
   Enter 10 km run time: 15
   Enter 10 km run time: 20
   Enter 10 km run time: 10
   Enter 10 km run time: <enter>
   Average of 15.0, over 3 runs
   ```

2. **Ordinal Suffix**[2]: While cardinal numbers refer to the size of a group of objects like "four apples" or "1,000 tickets", ordinal numbers refer to the place of an object in an ordered sequence like "first place" or "30th birthday". This exercise involves numbers but is more an exercise in processing text than doing math. In English, ordinal numerals have suffixes such as the "th" in "30th" or "nd" in "2nd". Write an ordinal suffix program with an integer input named `number` and output a string of the number with its ordinal suffix. For example, ordinal suffix for 42 should return the string '42nd'. You may use Python's `str()` function to convert the integer argument to a string.
3. **Smallest and Biggest**: Write a program that reads a sequence of integers from the user and displays at the end the biggest and the smallest number that they have introduced.
4. **Buy 8 Get 1 Free**[2]: Let's say a coffee shop punches holes into a customer's card each time they buy a coffee. After the card has eight hole punches, the customer can use the card to get their 9th cup of coffee for free. In this exercise, you'll translate this into a simple calculation to see how much a given quantity of coffees costs while considering this buy-8-get-1-free system. Write a program that has a inputs named `number_of_coffees` and `price_per_coffee`. Given this information, the program displays the total cost of the coffee order. This is not a simple multiplication of cost and quantity, however, because the coffee shop has an offer where you get one free coffee for every eight coffees you buy. For example, buying eight coffees for $2.50 each costs $20 (or 8 × 2.5). But buying nine coffees also costs $20, since the first eight makes the ninth coffee free. Buying ten coffees calculates as follows: $20 for the first eight coffees, a free ninth coffee, and $2.50 for the tenth coffee for a total of $22.50.
5. **Rock, paper, scissors**[2]: Rock, paper, scissors is a popular hand game for two players. The two players simultaneously choose one of the three possible moves and determine the winner of the game: rock beats scissors, paper beats rock, and scissors beats paper. This exercise involves determining a game's outcome given the moves of the two players. Write a program that takes in the inputs for both players and then displays the winner out of the two

or tie if that is the case. Handle the situations when the input is neither rock, paper or scissors and display a message that the game cannot go on until a right option is chosen.

---

1. *Reuven M. Lerner, Python Workout,*
2. *Al Sweigart, Python Programming Exercises, Gently Explained.*

Optionally, you can try adding functions in the exercises above.