Abstraction is a way that we can simplify complex code into simple code. Similar to creating functions in Python or C#, we replace a big chunk of code with a simple line to call the function. Considering the main function, it can be more organized, as the content of the function is outside of it, leaving the key or essential part of the code in the main function.

Abstraction brings many benefits. It can simplify our main code. In terms of application, creating a function or a class can be one of its many forms. Since we are condensing multiple lines of code into 1 or a few lines, it is a form of abstraction.

Consider the code below:

**def** read\_dictionary(filename, key\_column\_index):

    """Read the contents of a CSV file into a compound

    dictionary and return the dictionary.

    Parameters

        filename: the name of the CSV file to read.

        key\_column\_index: the index of the column

            to use as the keys in the dictionary.

    Return: a compound dictionary that contains

        the contents of the CSV file.

    """

*# Create an empty dictionary that will*

*# store the data from the CSV file.*

    dictionary = {}

    with open(filename, "rt") as csv\_file:

        reader = csv.reader(csv\_file)

        next(reader)

        for row\_list in reader:

            if len(row\_list) != 0:

                key = row\_list[key\_column\_index]

                dictionary[key] = row\_list[1]

    return dictionary

**def** main():

    student\_dictionary = read\_dictionary("students.csv", 0)

pass

This is an excerpt of a code from a CSE 111 assignment. This function is for reading a CSV file and adding the data collected into a dictionary. With this function, to perform the above action only requires 1 line of code in the main function. Without it, the main function would be much longer. While the total lines of code may seem the same, the function can be reused and thus, perform the attributes of abstraction.

Also consider the code below:

**public** **class** Resume() {

**public** string \_name;

**public** List<Job> \_jobs = new List<Job>();

**public** void Display() {

        Console.WriteLine($"Name: {\_name}");

        Console.WriteLine("Jobs:");

        foreach (Job job in \_jobs) {

            job.DisplayJobDetails();

        }

    }

}

This is an excerpt of a code from this week’s learning activity. Such benefits due to the application of abstraction, can also be shown when using classes, where multiple variables are declared in a class, like a group. We can assign the group of variables for different users, and reuse them, just like functions. Without the need to declare a list or new variables for each user, we can simply call the class and name the variables according to the structure in the class.