\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CSC121 PYTHON Programming**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Lesson 01 Design Computer Programs to Solve Problems

# **Objectives**

In this lesson, students will learn:

- To design computer programs to solve a problems

- To identify input and output items of a program

- To determine the steps in a program’s algorithm

- To test a program’s algorithm

# **1.1 Solving Problems**

You are learning to write computer programs in this course. Have you ever wondered why people write computer programs?

Of course, it is fun to write programs, but people also write programs for a practical reason. Many programs are written to solve problems. For example, a power company may need a program to calculate how much to charge a customer, a department store may need a program to schedule worker shifts, a marketing manager may need a program to analyze consumer shopping patterns, etc. In fact, many complex problems, such as weather forecast, are too hard to solve without using a computer program.

There are many problems in this world, but not every single problem can be solved by a computer program. What type of problems are we going to write programs to solve in this course? Most of the problems we are going to solve require some arithmetic calculations and logical processing. The following is an example:

*A prepaid phone card charges the following rate for any call made within the country: a connection fee of 10 cents per call, plus a time charge of 2 cents per minute. Write a program to calculate the total charge for a call.*

You will be able to write programs to solve simple problems like this within a couple of weeks. After that you will learn to write programs to solve problems that are larger and more complex.

# **1.2 Designing a Computer Program: Input, Output and Algorithm**

Before we actually write a program, we need to understand the specifics of the problem and figure out a way to solve it. We need to ask ourselves three questions:

1. What outcome we want the program to generate?
2. What information must the user of the program provide when the program is running in order to get the desired outcome?
3. What are the steps the program needs to take to generate the desired outcome?

Typically, a program will display the outcome after it has finished its work. The displayed outcome is called the **output** of the program. For many programs we are going to write, the output is the final result of some computational and logical processing steps the program has performed. When we design a program, we must first decide exactly what we want the program to deliver to the user. Therefore, the first thing to do is to determine the output.

The information the user needs to provide is the **input** of the program. Most computational and logical operations require some data. Very often the program asks the user to enter data into the program when the program begins to run so the computational and logical processing steps in the rest of the program will have the necessary data to work on.

Once we identify the input and output of the program, we need to design a set of steps for the program to generate the desired output. This set of steps is called the **algorithm** of the program. A complete algorithm typically includes three kinds of steps: *input steps* to specify what we ask the user to provide, *processing steps* that perform computational and logical operations on the data, and *output steps* to specify what to display to the user.

Let’s design a program for the phone card problem described earlier.

|  |  |
| --- | --- |
| Output | total charge of a call |
| Input | length of the call |
| Algorithm | 1. Input length of the call 2. Calculate time charge = 0.02 \* length of the call 3. Calculate total charge = time charge + 0.10 4. Display total charge |

The output of this program is pretty obvious. Since the goal of the program is to determine the total charge of a call, obviously it is the output of the program. In other words, when the program has finished its work, we want the program to show us how much money the customer needs to pay for the call.

The input is not hard to figure out. Since the total charge depends on the length of the call, we must have that entered into the program by the user. Otherwise, we do not have enough information to determine the total charge. Other than that, we don’t need the user to enter any other information.

There are four steps in our algorithm. The program gets length of the call from the user in the first step so by the time the program performs the calculations in step 2 and step 3, it has the necessary data.

It takes two steps to determine total charge of the call. In step 2 of the algorithm we multiply length of the call by 0.02 to calculate time charge because the rate is $0.02 per minute. The time charge calculated in step 2 is used in step 3 to calculate total charge. The total charge is the total of time charge and the $0.10 connection fee.

The total charge is displayed in step 4. Without this step, the program is useless because the user cannot see the total charge calculated by the program.

In the example above, the steps in the algorithm are expressed as pseudocode, which is a combination of simplified mathematical and English expressions. Pseudocode is a type of informal program code that is used to express the logic of an algorithm. It is not real program code and it cannot be executed by a computer.

# **1.3 Gross Pay Example**

Let’s look at another example.

*A store pays its hourly workers every two weeks. Write a program to calculate and display the gross pay for an hourly worker.*

Let’s design a program to solve this problem. We start with identifying the output. The problem description says “calculate and display the gross pay for an hourly worker”. So that will be the output.

|  |  |
| --- | --- |
| Output | gross pay for an hourly worker |
| Input |  |
| Algorithm |  |

Next, we identify the input. To calculate gross pay, we need to know the hourly pay rate for the worker, and the number of hours the worker has worked during this pay period.

|  |  |
| --- | --- |
| Output | gross pay for an hourly worker |
| Input | hours worked  hourly pay rate |
| Algorithm | 1: Input hours worked  2: Input hourly pay rate  3: Calculate gross pay = hours worked \* pay rate  4: Display gross pay |

This completes the program design. We will code this program in a future lesson.

# **1.4 Air Conditioner Example**

*The power of an air conditioner is measured in British Thermal Units (BTU). The higher the BTU, the more heat the air conditioner can bring away. When people buy an air conditioner, they need to know how many BTU they need to keep the room cool. Write a program to estimate how many BTU we need when we install a window air conditioner in a room. This number is determined by the volume of the room. The rule of thumb is that we need 3.5 BTU per cubic foot. The program should ask the user to enter the length, width and height of the room. It should calculate and display the number of BTU needed for the air conditioner.*

Let’s design a program to solve this problem. We start with identifying the output. The problem description says “calculate and display the number of BTU needed for the air conditioner”. So that will be the output.

|  |  |
| --- | --- |
| Output | BTU needed |
| Input |  |
| Algorithm |  |

Next, we identify the input. We need the user to provide the length, width and height of the room so we can calculate the volume and use it to determine how many BTU we need. The program should not ask the user to enter volume of the room directly because the problem description clearly tells us to “ask the user to enter the length, width and height of the room”.

|  |  |
| --- | --- |
| Output | BTU needed |
| Input | room length  room width  room height |
| Algorithm |  |

Now, let’s think about the algorithm. We ask the user to enter room length, width and height when the program begins to run. We use the data to calculate room volume. Next, use the rate of 3.5 BTU per cubic foot to calculate the number of BTU needed. Finally, we display BTU needed to the user.

|  |  |
| --- | --- |
| Output | BTU needed |
| Input | room length  room width  room height |
| Algorithm | 1. Input room length, room width and room height 2. Calculate room volume = room length \* room width \* room height 3. Calculate BTU needed = room volume \* 3.5 4. Display BTU needed |

# **1.5 Credit Card Reward Example**

*A credit card company has a new reward program. A cardholder earns points for all expenses charged to his account. Every dollar spent on gasoline will earn the cardholder 4 points. Every dollar spent on grocery will earn 3 points. Every dollar spent on all other things will earn 2 points. Write a program to do the following. Ask the user to enter three items: amount spent on gasoline, amount spent on grocery, and amount spent on other items. The program will calculate and display two items: total expenses and number of points earned.*

Let’s design a solution for this problem. We start with identifying the output. The problem description says “calculate and display two items: total expenses and number of points earned”. Therefore, there are two output items.

|  |  |
| --- | --- |
| Output | total expenses  number of points earned |
| Input |  |
| Algorithm |  |

Next, we identify the input items. We need the user to enter amount spent on gasoline, grocery and other items, respectively.

|  |  |
| --- | --- |
| Output | total expenses  number of points earned |
| Input | amount spent on gasoline  amount spent on grocery  amount spent on other items |
| Algorithm |  |

Now let’s think about the algorithm. Obviously we need the user to enter amount of expenses of each type when the program begins to run. Then we can calculate the points earned for each type of expenses separately. Next, we calculate the total expenses and the total number of points earned. Finally, we display these results to the user.

|  |  |
| --- | --- |
| Output | total expenses  number of points earned |
| Input | amount spent on gasoline  amount spent on grocery  amount spent on other items |
| Algorithm | 1. Input amount spent on gasoline, amount spent on grocery, and amount spent on other items 2. Calculate points earned from gasoline = amount spent on gasoline \* 4 3. Calculate points earned from grocery = amount spent on grocery \* 3 4. Calculate points earned from other items = amount spent on other items \* 2 5. Calculate total expenses = amount spent on gasoline + amount spent on grocery + amount spent on other items 6. Calculate total points earned = points earned from gasoline + points earned from grocery + points earned from other items 7. Display total expenses and total points earned |

# **1.6 Common Mistakes in Designing Algorithms**

There are a few common mistakes when we design algorithms.

1. **Forgetting input steps in algorithm**: The computer will not read any input unless it is told to do so. The following algorithm is faulty because it does not have input step.

|  |  |
| --- | --- |
| Algorithm  (missing input step) | 1. Calculate room volume = room length \* room width \* room height 2. Calculate BTU needed = room volume \* 3.5 3. Display BTU needed |

1. **Forgetting output steps in algorithm**: The computer will not display any output unless it is told to do so. The following algorithm is faulty because it does not have output step.

|  |  |
| --- | --- |
| Algorithm  (missing output step) | 1. Input room length, room width and room height 2. Calculate room volume = room length \* room width \* room height 3. Calculate BTU needed = room volume \* 3.5 |

1. **Incomplete calculation steps in algorithm**: We must specify **what** to calculate and **how** to calculate it in every calculation step. The following algorithm is faulty because its calculation steps are unclear

|  |  |
| --- | --- |
| Algorithm  (incomplete calculation steps) | 1. Input room length, room width and room height 2. room length \* room width \* room height 3. Calculate BTU needed 4. Display BTU needed |

Step 2 does not specify what to calculate, while step 3 does not specify how to calculate BTU needed. The computer cannot finish its works with incomplete steps.

1. **Steps in wrong order**: The order of the steps is very important. When you write a calculation step, you need to ensure that you have all the necessary data entered in the program. When you write an output step to display something, you need to ensure that the item you are going to display is already calculated properly.

|  |  |
| --- | --- |
| Algorithm  (steps in wrong order) | 1. Calculate room volume = room length \* room width \* room height 2. Input room length, room width and room height 3. Display BTU needed 4. Calculate BTU needed = room volume \* 3.5 |

The steps in this algorithm are in wrong order. It calculates room volume before it gets room length, width and height. It also tries to display BTU before it is calculated.

*Pay attention to these mistakes when you design computer programs.*

# **1.7 Testing Your Algorithms**

Once you have designed an algorithm, you should make up some data to test the algorithm. This will help you to discover errors in your steps.

When a program is running, some memory cells in the computer are used to store data. Inside the program, we use something called **variables** to represent these memory cells. Each variable in a program is associated with a cell in the computer’s memory. Each memory cell can store one piece of data. In a computer program, we need variables to store input items entered by the user, as well as results of calculations. The data item stored in a variable is called the **value** of the variable. Program code can do two things to a variable: read and write. To write to a variable means to store a new value in a variable. If the variable already has a value there, the new value will overwrite the old value. To read a variable means to retrieve the value of a variable, typically for displaying or other processing purpose. When we retrieve the value of a variable, we do not remove the value from the variable. We simply look it up. The value remains in the variable until it is overwritten by a write operation.

To test an algorithm, we need to execute the steps by hand. Most steps deal with data. Therefore, when we execute the steps, we need to create variables and read or write to these variables whenever necessary. We also need to use a table to keep track of the value changes in the variables. Let’s look at a few examples.

## **Example 1**

Earlier we saw the algorithm for the pay calculating program:

Step 1: Input hours worked

Step 2: Input hourly pay rate

Step 3: Calculate gross pay = hours worked \* pay rate

Step 4: Display gross pay

Let’s execute this algorithm by hand and keep track of the variables.

Step 1: Input hours worked

The user enters hours worked into the program. We need to store the number entered by the user in a variable so it can be used later. Let’s assume that the hours worked is 20.

|  |  |
| --- | --- |
| Step | Hours worked |
| Input hours worked | 20 |

Step 2: Input hourly pay rate

The user enters hourly pay rate into the program. We need to store the number entered by the user in a variable so it can be used later. Let’s assume that the hourly pay rate is 8.5. Since no change is made to hours worked, it remains 20.

|  |  |  |
| --- | --- | --- |
| Step | Hours worked | Hourly pay rate |
| Input hours worked | 20 |  |
| Input hourly pay rate | 20 | 8.5 |

Step 3: Calculate gross pay = hours worked \* pay rate

The computer uses the values stored in the variables hours worked and hourly pay rate to calculate gross pay. Since no change is made to hours worked and hourly pay rate in this step, they remain 20 and 8.5, respectively. We need a new variable to store gross pay. Since 20 \* 8.5 = 170, the computer will store 170 in gross pay.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Hours worked | Hourly pay rate | Gross pay |
| Input hours worked | 20 |  |  |
| Input hourly pay rate | 20 | 8.5 |  |
| Calculate gross pay = hours worked \* pay rate | 20 | 8.5 | 170 |

Step 4: Display gross pay

The computer looks up the value stored in the variable gross pay, which is 170, and displays it on the computer screen. Since no change is made to any of the variables, their values remain the same.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Hours worked | Hourly pay rate | Gross pay |
| Input hours worked | 20 |  |  |
| Input hourly pay rate | 20 | 8.5 |  |
| Calculate gross pay = hours worked \* pay rate | 20 | 8.5 | 170 |
| Display gross pay | 20 | 8.5 | 170 |

## **Example 2**

Earlier we also saw this pseudocode for the phone card program:

Step 1: Input the length of the call

Step 2: Calculate time charge = length of call \* 0.02

Step 3: Calculate total charge = 0.10 + time charge

Step 4: Display total charge

Let’s execute this algorithm by hand and keep track of the variables.

Step 1: Input the length of the call

The user enters the length of the call into the program. We need to store the number entered by the user in a variable so we can use it later. Let’s assume that the length of the call is 7 minutes.

|  |  |
| --- | --- |
| Step | length of call |
| Input the length of the call | 7 |

Step 2: Calculate time charge = length of call \* 0.02

The computer uses the value 7 stored in the variable length of call to calculate time charge. Since no change is made to length of call in this step, it remains 7. We need a new variable to store time charge. Since 7 \* 0.02 = 0.14, the computer will store 0.14 in time charge.

|  |  |  |
| --- | --- | --- |
| Step | length of call | time charge |
| Input the length of the call | 7 |  |
| Calculate time charge = length of call \* 2 | 7 | 0.14 |

Step 3: Calculate total charge = 0.10 + time charge

The computer uses the value 0.14 stored in the variable time charge to calculate total charge. Since no change is made to length of call and time charge in this step, they will remain 7 and 0.14, respectively. We need a new variable to store total charge. Since 0.14 + 0.10 = 0.24, the computer will store 0.24 in total charge.

|  |  |  |  |
| --- | --- | --- | --- |
| Step | length of call | time charge | total charge |
| Input the length of the call | 7 |  |  |
| Calculate time charge = length of call \* 0.02 | 7 | 0.14 |  |
| Calculate total charge = 0.10 + time charge | 7 | 0.14 | 0.24 |

Step 4: Display total charge

The computer looks up the value of total charge, which is 0.24, and displays it on the computer screen. Since no change is made to any of the variables, their values remain the same

|  |  |  |  |
| --- | --- | --- | --- |
| Step | length of call | time charge | total charge |
| Input the length of the call | 7 |  |  |
| Calculate time charge = length of call \* 0.02 | 7 | 0.14 |  |
| Calculate total charge = 0.10 + time charge | 7 | 0.14 | 0.24 |
| Display total charge | 7 | 0.14 | 0.24 |

## **Example 3**

Earlier we also saw this pseudocode for the air conditioner program:

Step 1: Input the length of the room

Step 2: Input the width of the room

Step 3: Input the height of the room

Step 4: Calculate volume = length \* width \* height

Step 5: Calculate BTU needed = volume \* 3.5

Step 6: Display BTU needed

Let’s execute this algorithm by hand and keep track of the variables. This time we only show the final table that has all the steps and variables in it since you probably have understood the process of executing a program by hand. Let’s make up a test case in which the length, width and height of the room is 15, 11 and 10, respectively.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Step | Room length | Room width | Room height | Room volume | BTU needed |
| Input room length | 15 |  |  |  |  |
| Input room width | 15 | 11 |  |  |  |
| Input room height | 15 | 11 | 10 |  |  |
| Calculate volume = length \* width \* height | 15 | 11 | 10 | 1650 |  |
| Calculate BTU needed = volume \* 3.5 | 15 | 11 | 10 | 1650 | 5775 |
| Display BTU needed | 15 | 11 | 10 | 1650 | 5775 |

# **1.8 Conclusion**

In this lesson, we learned how to design algorithms for simple programs. A typically algorithm consists of three types of steps: input steps, processing steps and output steps. Input steps get necessary data into the program. Processing steps perform calculations (and other types of processing that we will learn later) and store results in variables. Output steps display results of calculations. The order of steps is important. We cannot perform a calculation without the necessary data, nor display output without it being calculated first. We also learned how to execute the steps of an algorithm by hand and observe how values of variables change during the steps. In the next lesson, we will extend what we learned in this lesson by transforming algorithms into real Python programs.