# Module 5

This Module will help you understand **what computer programming is** and what the **basic concepts/structures of computer programming** are. You will gain a basic understanding of **programming strategies** and you will also explore **designing an algorithm**.

# Module Sections

* 1. What is computer programming?
  2. Programming Strategies/Algorithm Design

# 5.1 What is computer programming

Recall back to Module 1 where we discussed hardware and software. We learned that software is what tells hardware what to do by providing instructions. The instructions solve a problem of some sort. Now you may say that needing to send a print job to the printer is not a problem but it actually is. Anything that we desire to do requires a set of instructions or steps to follow to get to the end result. This makes everything we do a problem with a set of instructions to gain an end result. A program is that set of instructions created to solve a problem (complete a desired task) and therefore programming is the creation of those instructions. With respect to computers, the set of instructions are for the computer to carry out and accomplish an end result. In order to do this there are some basic concepts/structures to use

There are three common concepts and structures for computer programming. First you have to have a **sequence of instruction**. This means, you have the right instructions in the correct order. This is very important in any set of instructions for a computer. Think about the sequence of getting dressed. Do you put your undergarments on first and then your outer garments? Of course you do! If the sequence was followed out of order, say, you put your coat on and then your pants and decide that now you need to put on your shirt and undergarments, you are in a bit of a debacle. Following the correct sequence is essential to success in reaching the desired end result! On occasion there are instructions where the sequence isn’t as important. Where this occurs, we lean on a more predictable order or standard based on norms relative to culture. Think about the writing of an address on a piece of mail in the United States. We learn in school to address an item in a certain sequence. However, the postal service workers use the exact reverse of this order to get the item delivered.

Next we talk about **conditional structures**. This is where decisions dictate the next step followed, such as considering if something is true or false and then performing an action based on the answer. These decisions help determine separate outcomes based on the decision made. This is why they are called conditional structures. The condition determines the route of the instructions. These are typically called if statements in computer programming and follow the order of IF…THEN…ELSE. If we do this, then this happened, otherwise if we don’t then this happens.

Finally there are **looping structures**. These differ from the conditional because there are not multiple different outcomes; the statement requires a repeat of the same instructions until a condition occurs or repeat of the same instructions to maintain an existing condition. Most often these statements in computer programming are simply to do something a specified number of times and stop or an unspecified number of times until a condition is met and then stop.

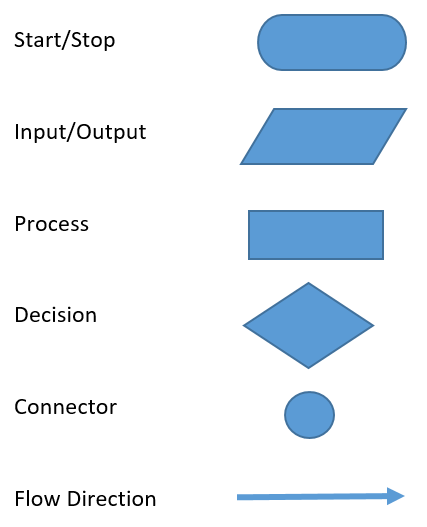
# 5.2 Programming Strategies/Algorithm Design

Computer programs can range in size and complexity. For simple tasks like turning on a light, there are very few instructions and really no need for a strategy. However, for more in-depth tasks you shouldn’t think of starting to code without using a strategy to work through the logic of the program. We consider what the problem is and design an algorithm to work through and solve the problem. An algorithm is the plan we write out to solve our problem.

When we solve any problem we most likely try to look at different pathways to get to an end result. Then we try to process which of those pathways is the best route to take. Sometimes there may be more than one way that seems the best route. Designing an algorithm will help us narrow down our choices and determine if there really is a best solution.

To create an algorithm we must know what our desired end result will be and we must consider all of the potential issues that we will meet while working towards that end result. The basic things to consider and ensure you understand the problem is to decide what the inputs and outputs will be, are there any decisions to be made or any steps that will need to be repeated, and what order will these steps need to be in. Once you have all of this, then you are ready to write out the algorithm. We will discuss two common algorithm design methods that work very well and have been used for many years; flow charting and pseudocode.

Flowcharting is a method of mapping out the program based on the major concepts or structures it will need. This will help to visualize the logical flow of the program. A flow chart is a diagram of standard symbols that represent the step-by-step instructions to solve the problem. The most common flowchart symbols are shown below.



Pseudocode is basically the algorithm written out in standard wording somewhat like an outline. There are not defined standards for writing this out however certain notations and structures should be used anyway. The most common notations are:

1. Input
2. Output
3. While
4. For
5. Repeat-Until
6. If-Then-Else

Also, typically when writing steps that reside inside the last four items, you should ident the text.

These design methods seem time consuming and almost unnecessary, right? This is because our brains usually work autonomously without all of this explicit guidance. A computer on the other hand needs these well thought out and very specific instructions. There is a saying in the computing world that a computer is only as smart as the person using it. It may be safe to say that a computer program is only as smart as the person who wrote the code! Computers are not sentient beings, they need us humans to tell them exactly what to do!

Complete the tutorials found at the link below for more information.

<https://www.bbc.com/bitesize/guides/z3bq7ty/revision/1>

These tutorials are provided by the [British Broadcasting Corporation’s Bitesize](https://www.bbc.com/bitesize/) resource and cover the following topics.

Designing an Algorithm

Pseudocode

Flowcharts

**Work through the Designing an Algorithm module. Be sure to click on the hyperlinks if you need more information! Review all three pages and then take the test. The test is not graded, but will provide you a good feedback on your understanding of the content.**

# Module 5 Project

The goal of this project is to test your knowledge on the basics of programming structures. More specifically, you will design your own algorithm using both pseudocode and a flowchart.

Think of some daily activities that you perform. You are going to prepare a plan and design an algorithm that can be translated into a program that would tell a computer how to do the activity you have chosen.

1. In paragraph form, describe in detail the big picture, the final goal, and the individual stages of your algorithm. If there are issues to overcome, write out possible solutions for each. Also, describe your inputs, outputs, decisions, and repetitions in the algorithm. Think about sequence when writing this out.
2. Using the standard notation for pseudocode that you have learned, write out the algorithm for the daily activity you have selected. Be sure to indent where required.
3. Using your pseudocode from step two, create a flowchart of your algorithm. Utilize the correct symbols in your flowchart that you learned about in the module.
4. Share your work in the appropriate discussion forum.
5. **You must also provide respectful feedback to at least two (2) other students’ posts. This includes identifying what you may see as errors or maybe a better way to write their algorithm. Compliments are also always welcome however, all comments are to be at least two complete sentences and contain more than phrases such as “I agree.”**