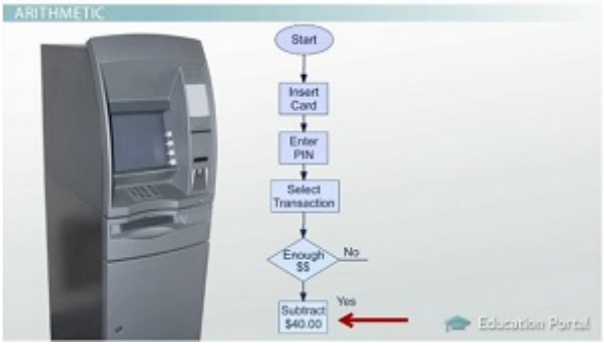
There are five basic programming elements, or operations: input, output, arithmetic, conditional, and looping. Every program uses at least two of these. This lesson will describe each one to you and show you examples in simple code.

Five Basic Programming Elements

Programming is somewhat like working with building blocks. Given enough children's toy blocks (and enough time and ingenuity), you can build just about anything with only a few kinds of blocks. The five basic elements in programming are:

1. **input**: getting data and commands into the computer
2. **output**: getting your results out of the computer
3. **arithmetic**: performing mathematical calculations on your data
4. **conditional**: testing to see if a condition is true or false
5. **looping**: cycling through a set of instructions until some condition is met

Let's look at an ATM. You walk or drive up to it, swipe your ATM card, tell it how much money you want, and, after chugging for a moment or two, it spits out cash and a receipt. In this simple example, we have input, arithmetic, and output. Below is a simple flowchart for getting $40 in 'Fast Cash' from a teller machine. We'll follow it through this lesson to show how the five operations can be mapped.



Programming elements flow chart

Input

Examples of input devices

Input can come from just about anywhere: a keyboard, a touchscreen, a text file, and another program are just a few examples. Input is one of the two elements that are used by every program because every program needs some data to work with. When you use an ATM, the input comes from two things: your ATM card and the information you enter on the keypad. Your ATM card has some customer identification on the magnetic strip on the back. It tells the ATM who you are so it can get your account information. But before it will do that, you have to 'authenticate' yourself. The bank wants to make sure it isn't someone who's not you trying to access your account. You authenticate yourself by entering your PIN, which is (or should be) known only to you. Once the bank is happy that it's you, it will ask you what you want to do. In this case, you just want $40, so you select 'Fast Cash,' which will take the money out of your checking. And, the input part of your transaction is over. In a flowchart, input leads to the first process. In this example, it is arithmetic.

Arithmetic

Computers can perform all kinds of mathematical operations and functions, from the simple addition or subtraction needed to update your checking account balance after a withdrawal or deposit, to the complex calculus needed to put a satellite into orbit. Not every program needs to do calculations on the data that's entered, but it may still need to do some in order to control what is happening inside the program itself. We'll see that in a minute. In our example, the ATM will check your balance and, if you have enough money, subtract $40 from it. And, that's the arithmetic part - the subtraction. So, this part of our flowchart is a process showing subtraction.

Output

Output is the result that your program gives you. That is the whole purpose of writing a program: to ask a question and get the answer! Output can take many forms - text or graphics, either printed or on a screen, a sound - just about any form that can be interpreted and understood by a human being or another program. In the case of our teller machine, the output is one of the most widely understood things in human society: cash. Back to our example, the machine dispenses the $40 and prints a receipt. The receipt shows you how much you withdrew and what your new balance is. And, that's the output. So, our flowchart goes from the subtraction to the output process, then our program ends.

Looping

Quite often, your program has to repeat an operation a number of times before the program can continue. The simplest example is adding up a column of numbers. Since a computer can only add two numbers at a time, it has to add the first two numbers, then add the next number to the total, then add the next one and the next one until there are no more numbers to be added. There are a number of different types of loops, which are used based on how the input or calculations need to be handled. All of them are either event-controlled or counter-controlled. The control part is important - if there is no control, the loop can go on forever, or until you stop the program. **Event-controlled loops** can be stopped by an external event. That event could be user input, perhaps in response to a prompt like

'Any More?' or by reaching the end of an input file. **Counter-controlled loops** are stopped when a counter in the loop reaches a predetermined value. The loop may be counting down to zero or up to a maximum value. Think ticket sales for a ball game or concert.

Conditional

Loops need to be controlled. The loop ends when a predetermined condition is met. A programming condition looks at a program statement and figures out if it is true or false. Here is an example of an event-controlled loop:

while read name

do

echo $name

done < namelist
