**TAS – Software design and development – Part 2 transcript**

## Control structures revision: Sequence, selection and iteration

(Duration: 14 minutes 31 seconds)

You recall from the earlier video that we discussed that programming equals DO plus STORE or DO control structures in STORE data structures. We're going to have a more detailed look now at the control structures. So again, with our diagram, DO and STORE, DO represent the algorithms. And in particular, the use of sequence, selection and iteration. The three main ways of doing and we're going to look at those in more detail. So let's have a look at the first or simplest way of doing or control structures. The sequence. In the software design development course, you'll use two main ways to represent algorithms, a graphic way using flow charts and a text-based way using pseudo-code. The simplest type of control structure or way of doing is a sequence. And a sequence is simply a set of instructions or steps. And they're shown in a flow chart, by the use of a rectangle to symbolise a process.

So in this example, DO this, DO that, DO the other. Let's do a quick exercise to think through the logic and detail needed to create an algorithm. We want to make a algorithm for domestic servant robot that's making a cup of instant coffee. Take a few minutes now to write down the basic steps involved in this type of algorithm. You can pause the video here for a moment. What are the steps? Well very generally, and a big sweep of the idea or an overview of what we might need to have is, and let's create a version for this for version control. We want to boil water, step one. Step two, put the coffee in the cup and step three at the water to the cup. And you might've had other steps in your algorithm and they might be just as valid, but you could probably agree that this algorithm is not detailed enough for a robot to interpret properly. It needs to be refined.

So take a few minutes now to create a refinement of your existing algorithm and provide a little more detail to make it more explicit and less open to misinterpretation or understanding. Okay to boil water, we need to do a number of steps. We need to fill the kettle. To fill the kettle, we need to put the kettle under the tap. We need to turn on the tap. We need to wait until the kettle is full and then we need to turn off the tap. Then we switch on the kettle and then we wait until it's boiling. You can see by the coding of this refined algorithm, with the 1.1 to fill the kettle underneath the boil water, that we're breaking it down and we're making it easier to follow.

This ability to refine an algorithm is an important aspect in computational thinking. And certainly an important aspect of this course. You might want to share your algorithm with a colleague or peer and see whether or not you could interpret it or whether you think your robot could interpret that algorithm and correctly make an instant coffee.

Okay, let's move now to the second type of control structure, selection. This control structure selection, or if you like the powerful if, is a very useful control structure used in very much, if not all programming languages. This if statement gives the computer at least the appearance of being able to make a choice or a decision. It's a very powerful tool. The if statement or selection is also called, branching and the decisions are made based on a comparison operator or a condition, equal to, not equal to, less than or greater than. there are two main types of if statements. There's a binary selection and there's a multi-way selection, which is often seen in other languages as a case statement. The conditions are always represented in a flow chart as a diamond. And you'll be using if statements throughout any algorithm that you solve within the HSC. Okay, take a few minutes now to have a look back at your coffee making scenario and the robot and see where it would make sense to include an if or a condition. Let's have a look at one possible place to include a condition or an if statement. The domestic servant robot needs an algorithm to make the coffee. Let's have a look at when he puts the coffee or the robot puts the coffee into the cup. What does the robot do it there's no coffee? Have a look at 2.1. If there's no coffee to take the jar off the shelf, they might want to see if the jar is empty. Then 2.1.2, if the jar is empty, get a new jar from the cupboard. 2.1.3, remove the lid from the jar. There'll be other places within this algorithm where you could include an if statement, but there's a good example of how in refining the algorithm, we can include a condition. Again, check your answers with the neighbour or peer and see what they had in the solution that they came up with.

And finally, the last of the control structures that we need for this course is iteration. Have a quick look at these two flow charts. You can pause the video for a few minutes and describe in words, the similarities and the differences between these two flowcharts. Did you answer that both of them represent loops or iteration and that one has the condition or the diamond at the start and the other has the condition or the diamond at the end of the processes? You might want to consider why you would do that. In these two flow charts or these two algorithms, these two loops are called a pre-test loop, when the condition comes before the process and a post-test loop, where the condition comes after the processes. Why would you choose to use one over the other? Well, one example might be that you've got a lot of data or a number of processes to complete. If the condition doesn't need to be entered, it wouldn't make sense to enter the first loop. Let's have a look here, wouldn't make sense to go and do all of these processes if you didn't need to enter this condition. In this example, for the post-test loop, you have to do these processes first. So in some circumstances, the pre-test, which is often called a while loop, while true. So the while loop, while this condition is true or pre-test is more efficient than the post-test and we'll have a look at an example of that. An unplugged example, you may want to think about or use is for example, to say, is it raining? Is it raining? Carry an umbrella or carry an umbrella until it stops raining. One comes before the process and the other comes after the process.

Here's a definition in some sample pseudo code from NESA and you can use here, we'll see how this loop works with the desk check on the bottom right-hand side. So the definition of iteration from the NESA glossary, a repetition of a process in a computer programme where each repeated cycle builds towards a desired result. An example of an iterative process with co-designed to add the numbers from one to nine in shown below. And there you'll see a little bit of pseudo code with a variable called sum set to zero, a variable called number that will go from one to nine and the variable sum will take the value of sum and number and put it into sum. One of the interesting things about people that first start to learn how to code, is the confusion they may come across from understanding that always the right-hand side of the equation goes into the left-hand side. So it's important to think of your variables as containers and you can see from this iteration or loop that we're combining here, the DO and the STORE to create the programme, but also a very useful tool to get in the habit of testing and checking all your algorithms is to be able to disc check, right? The variable numbers at the top of the table and pursue every line of code according to exactly what it said in each line. And that way you can get in the habit of testing your own algorithms, you can check out your neighbour’s algorithm and as we'll be doing shortly, looking at doing some examples from an HSC examination.

Here's a very simple example to understand the power of loops. Here we've got five lights, each with their own switch. To turn all the lights on, we have to write this code somehow or somewhere, switch the first light on, switch the second light on, switch the third light on, et cetera. So we're repeating the same instruction. Is there a better way? Take a minute to rewrite this code more efficiently. You can pause the video and have a go at writing the code more efficiently, you know, with a more efficient algorithm. One way that can help students visualise variables is to use a cup or a cylinder to represent a storage container or a variable, and you can see in this loop that the variables A is used for the counting and so we could write that code as for count A one to five, switch light A on. A is the variable or if you like, the STORE from our DO plus STORE equation. They were fairly trivial to examples, but what if you had a thousand lights or a million lights, how much typing would be saved if you used a loop, instead of writing out every line of code? The source code would be shorter and much easier to read. When we have to change our code to more or less lights, we simply modify one single number. So remember that loops are used to repeat certain instructions repeatedly.

End of transcript.