Current Skill Iterative processing

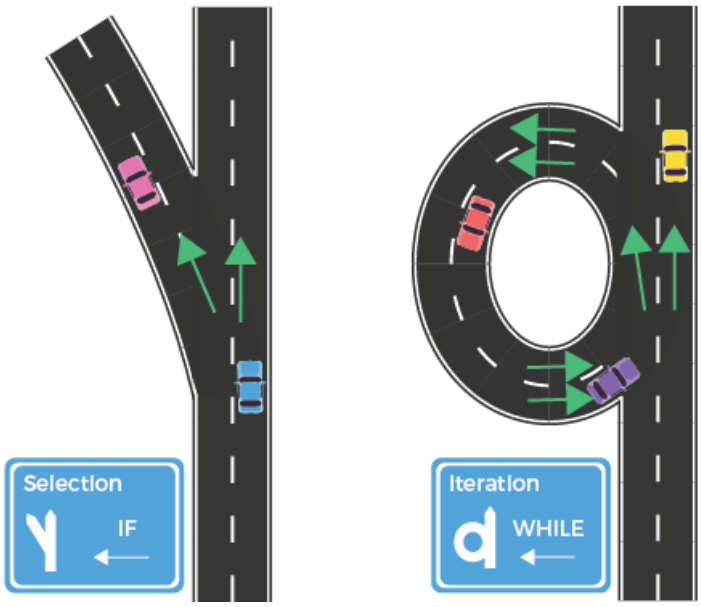
Iteration

Iteration in programming means **repeating** steps, or instructions, over and over again. This is often called a **'loop'**.

Algorithms consist of instructions that are carried out (performed) one after another. Sometimes an algorithm needs to repeat certain steps until told to stop or until a particular condition has been met.

Iteration allows algorithms to be simplified by stating that certain steps will repeat until told otherwise. This makes designing algorithms quicker and simpler because they don't need to include lots of unnecessary steps.

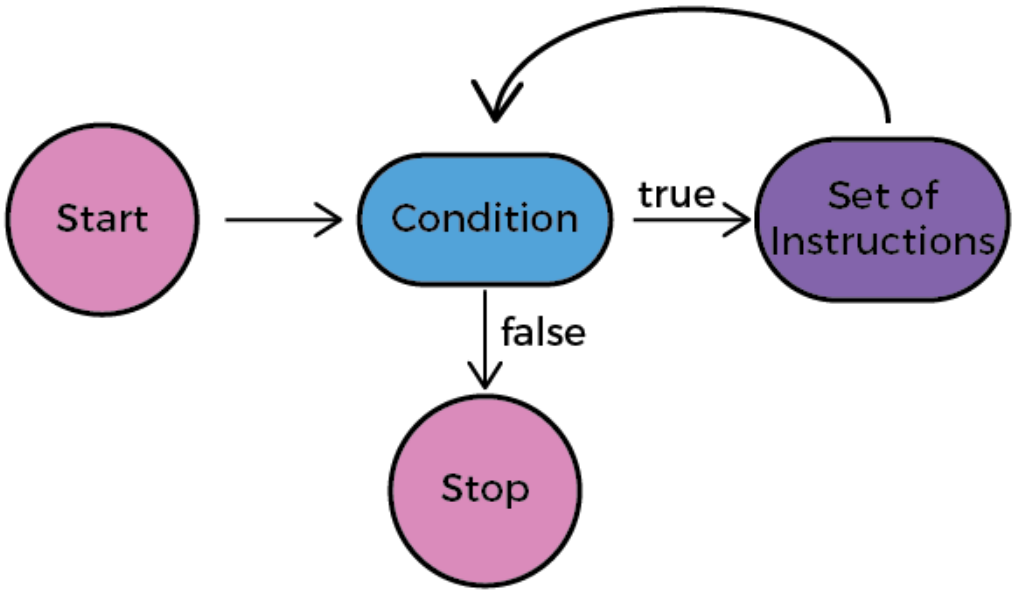
Using condition with iteration is very similar to selection as two routes are created. However, with iteration there is a loop back into the algorithm, rather than creating and keeping two separate routes as would occur with selection.



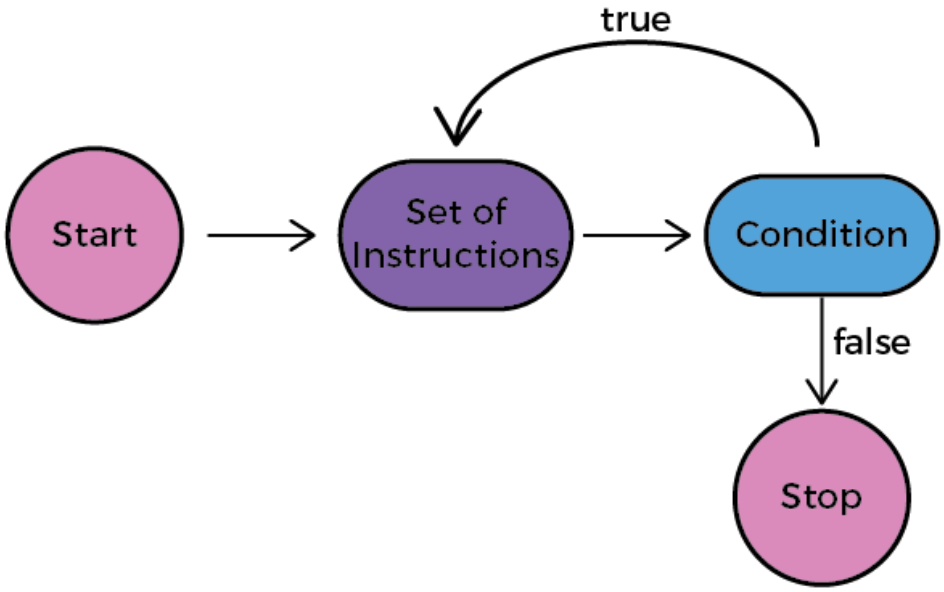
Iterative procession

Iteration

A WHILE-DO loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.



A REPEAT-UNTIL loop is similar to while loop with the only difference that it checks for the condition after executing the statements, and therefore is an example of Exit Control Loop. Here, the loop body will execute at least once.



FOR-DO loop

is a repetition control structure which allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

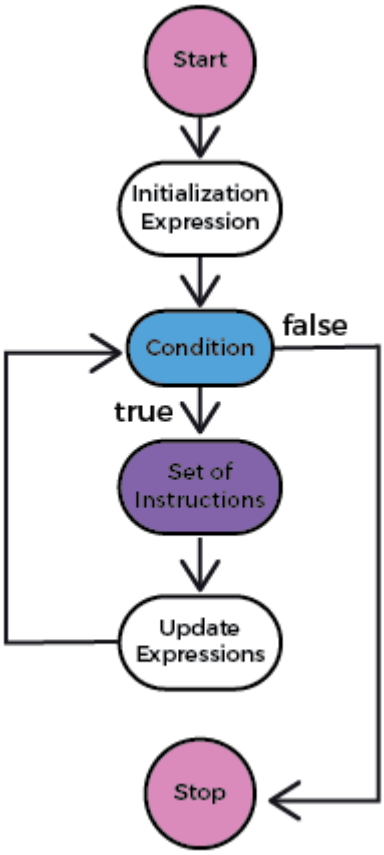
How does a For loop executes?

- Control falls into the for loop. Initialization is done
- The flow jumps to Condition
- Condition is tested.

1. If Condition yields true, the flow goes into the Body
2. If Condition yields false, the flow goes outside the loop

- The statements inside the body of the loop get executed.
- The flow goes to the Updating (increment/decrement current value)
- Updating takes place and the flow goes to Step 3 again

- The for loop has ended and the flow has gone outside.



Iterative processing in practice

Let's take an example to better understand the iterative processing.

In this type of processing, we have three kinds of iterators. We are going to cover them in these examples.

The first iterator is the for loop.

This kind of iterator is used when already knew how many times we are going to repeat the same instruction.

Let's say that we want to calculate the sum of the number from 1 to 10, in this example we already know how many time we are going to repeat the sum instruction.

The solution is shown below:

```
/* *** Calculate the Sum *** */


/* In this example we are going
to calculate the sum of number from
1 to 10 */
ALGORITHM iterative_for
VAR
```

```
i : INTEGER;
sum : INTEGER := 0;


BEGIN


FOR i FROM 1 TO 10 STEP step  DO
/*i represente the iterator variable,
the 1 represent the starting point
10 the end point
step represent the step of the iteration and it's optional
if we don't put then the default step will 1 */
   sum := sum + i
END_FOR
   Write(sum)
END
```

The second iterator is the while loop.

This iterator is used we don't have any idea about the number of iteration.

Let's take this example, suppose that the user introduces a number and we have to return how many time this number can be divided by 2

Here is the solution:

every time we divide the number n by 2, we increment the count by 1 and we repeat these instructions until the rest of the division by 2 is different than 0.

```
ALGORITHM iterative_while
VAR
count : INTEGER := 0;
// n is the number to test
n : INTEGER;


BEGIN
 WHILE (n%2 = 0 ) DO
 /* while the condition is true
```

```
 then it will repeat the instruction
 until the condition will be false
   */
   n := n/2; // every time we divide the number n by 2 and
   count := count +1; // increment the count by 1

 END_WHILE
 Write(count)
END
```

The third and last iterator is the REPEAT-UNTIL.

In this kind of iterator :

we execute the instruction

we test over the condition.

Let's have an example to make it more clear.

We suppose that we want the user to enter his age, but we have to make sure that the age is not negative or equal to 0

so we are going to ask him to enter his age, then we verify the age:

```
ALGORITHM iterative_repeat
VAR
age : INTEGER;

BEGIN
 REPEAT
     Read( age )
 UNTIL ( age>=0 )
 Write( age )
END
```

Unlike the while-do, in this type of iterator, we make sure that the instructions will be executed at least once.