Lecture Template

Do & While Loops

Topics

- **While Loops**
- **Do While Loops**
- **Continue and Break Statements**

While Loops

Parts of While Loops

There are many different kinds of loops, but they all essentially do the same thing: they repeat an action some number of times.

Three main pieces of information that any loop should have are:

1. **When to start:** The code that sets up the loop — defining the starting value of a variable for instance.
2. **When to stop:** The logical condition to test whether the loop should continue.
3. **How to get to the next item:** The incrementing or decrementing step — for example, `x = x * 3` or `x = x - 1`

While Loop

The JavaScript while statement creates a loop that executes a block of code as long as the test condition evaluates to true.

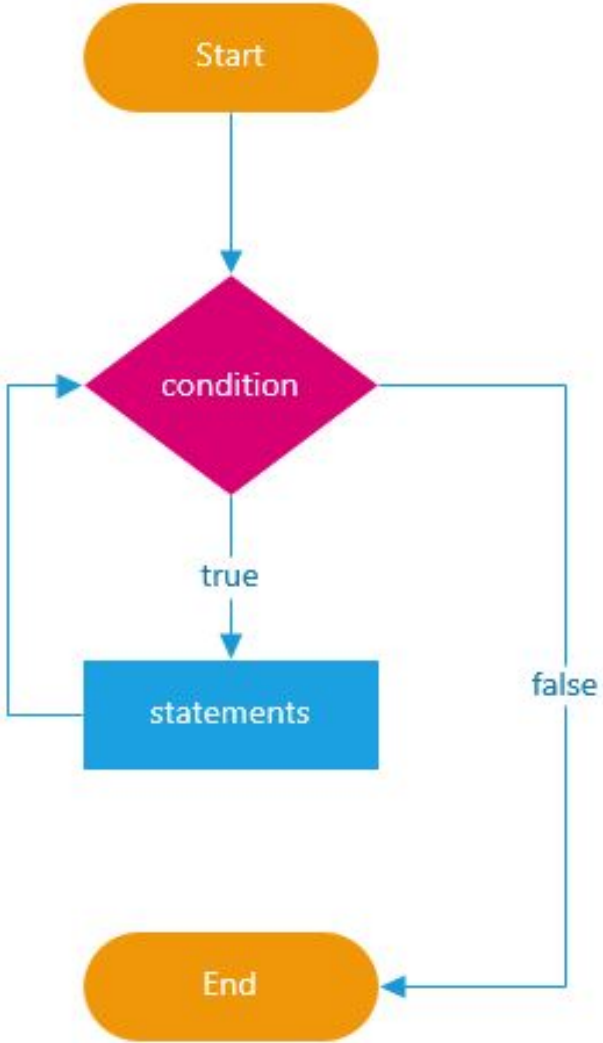
The while statement evaluates the expression before each iteration of the loop.

If the expression evaluates to true, the while statement executes the statement. If the expression evaluates to false, execution continues with the statement after the while loop.

While Loop

The while loop evaluates the expression before each iteration, therefore, the while loop is known as a pretest loop.

For this reason, it is possible that the statement inside the while loop is never executed.



```
var start = 0; // when to start
while (start < 10) { // when to stop
  console.log(start);
  start = start + 2; // how to get to the next item
}
```

> *Prints:*
>
> *0*
> *2*
> *4*
> *6*
> *8*

Here's a basic while loop example that includes all three parts.

While Infinite Loops

If a loop is missing any of these three things, then you might find yourself in trouble. For instance, a missing stop condition can result in a loop that never ends!

Don't run this code!

```
while (true) {
  console.log("true is never false, so I will never stop!");
}
```

If you did try to run that code in the console, you probably crashed your browser tab.

While Infinite Loops cont..

Here's an example where a loop is missing how to get to the next item; the variable `x` is never incremented. `x` will remain 0 throughout the program, so the loop will never end.

Don't run this code!

```
var x = 0;

while (x < 1) {
  console.log('Oops! x is never incremented from 0, so it will ALWAYS be less
}
```

This code will also crash your browser tab, so we don't recommend running it.

```
let x = 0;
while (x < 10) {
  console.log(x);
  x--; // same as i = i - 1
}

Output:
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
...etc.
```

Video - While Loop

Do While Loops

Do While Loop

The do-while loop statement creates a loop that executes a block of code until a test condition evaluates to false.

The following statement illustrates the syntax of the do-while loop:

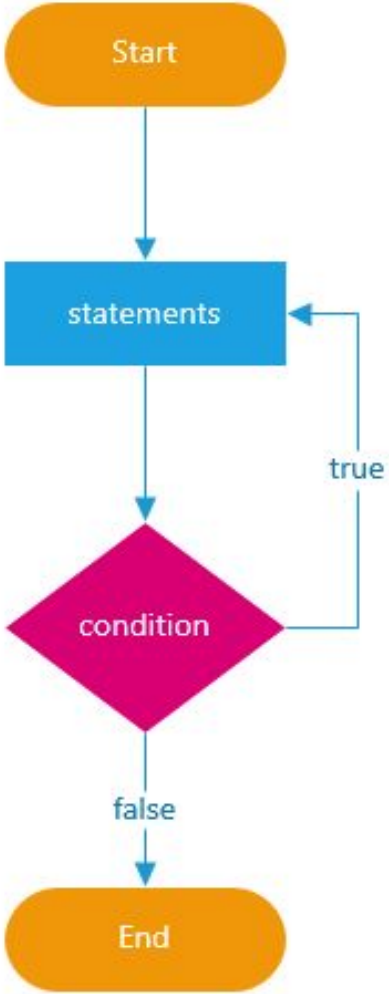
```
do {
  statement(s);
} while(expression);
```

Unlike the while loop, the do-while loop always executes the body at least once before it evaluates the expression.

Do While Loop

Because the expression is evaluated only after the body of the loop has been executed, the do-while loop is called a post-test loop.

Inside the body of the loop, you need to make changes to some variable to ensure that the expression evaluates to false after iterations. Otherwise, you will have an indefinite loop.



In this example, the count variable is set to 0 and is incremented by one in each loop iteration. The loop continues as long as the count is less than 10.

```
let count = 0;
do {
    count++;
    console.log('count is:' + count);
} while (count < 10);
```

You often use the do-while statement in the situation that the body of the loop needs to execute at least one. This is an important feature of the do-while loop.

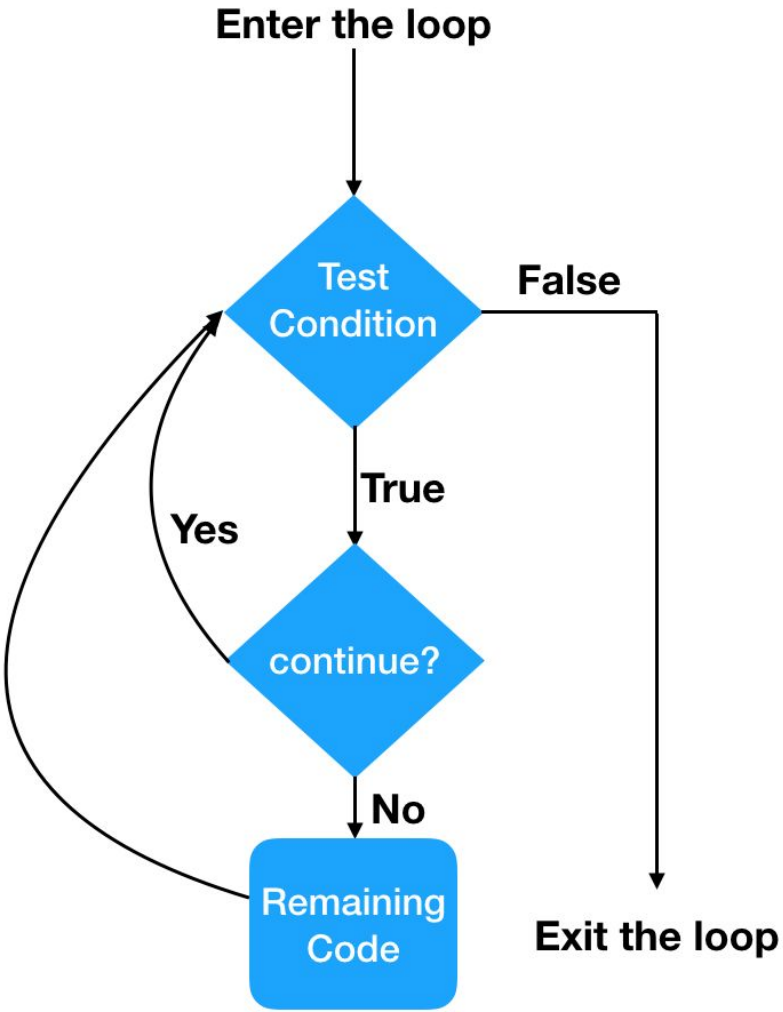
The most typical example of using the do-while loop is getting input from the user until the value provided is expected.

Continue Statement

Continue statement

The continue statement skips the current iteration of a loop and goes to the next one.

Because of this, the continue statement must appear in the body of a loop or you will get an error.



for loop - continue

The continue statement skips the current iteration of a for, do-while, or while loop. The continue statement skips the rest of the code to the end of the innermost body of a loop and evaluates the expression that controls the loop.

In a for loop, the continue skips all the statements underneath it and pass the execution of the code to the update expression, in this case, it is i++;

```
for (var i = 0; i < count; i++) {
    if (condition)
        continue; // Jumps to expression: i++
    // more statement here
}
```

while loop - continue

In a while loop, it jumps back to the expression that controls the loop.

```
while (expression){ // continue jumps here
    if (condition) {
        continue; // Jumps to expression
    }
    // more statements here
    // ...
}
```

do while loop - continue

In a do while loop, it jumps back to the expression that controls the loop.

```
do{
    if (condition) {
        continue; // Jumps to expression
    }
    // more statements here
    // ...
} while(expression); // continue jumps here
```

Video - Break & Continue