Loop in Java

while Loops

A **loop** statement allows to repeatedly execute a statement or group of statements.

A **while** loop statement repeatedly executes a target statement as long as a given condition is true.

Example:

```
int x = 3;

while(x > 0) {
  System.out.println(x);
  x--;
}
/*
Outputs
  3
  2
  1
*/
```

Try It Yourself

The **while** loops check for the condition x > 0. If it evaluates to true, it executes the statements within its body. Then it checks for the statement again and repeats.

Notice the statement x--. This decrements x each time the loop runs, and makes the loop stop when x reaches 0.
Without the statement, the loop would run forever.

while Loops

When the expression is tested and the result is false, the loop body is skipped and the first statement after the while loop is executed.

Example:

```
int x = 6;

while( x < 10 )
{
  System.out.println(x);
  x++;
}
System.out.println("Loop ended");

/*
6
7
8
9
Loop ended
*/
```

Try It Yourself

for Loops

Another loop structure is the **for** loop. A for loop allows you to efficiently write a loop that needs to execute a specific number of times.
Syntax:

```
for (initialization; condition; increment/decrement) {
   statement(s)
}
```

Initialization: Expression executes only once during the beginning of loop
Condition: Is evaluated each time the loop iterates. The loop executes the statement repeatedly, until this condition returns false.
Increment/Decrement: Executes after each iteration of the loop.

The following example prints the numbers 1 through 5.

```
for(int x = 1; x <=5; x++) {
  System.out.println(x);
}

/* Outputs
1
2
3
4
5
*/
```

Try It Yourself

This initializes x to the value 1, and repeatedly prints the value of x, until the condition x<=5 becomes false. On each iteration, the statement x++ is executed, incrementing x by one.

Notice the semicolon (;) after initialization and condition in the syntax.

for Loops

You can have any type of condition and any type of increment statements in the for loop.
The example below prints only the even values between 0 and 10:

```
for(int x=0; x<=10; x=x+2) {
  System.out.println(x);
}
/*
0
2
4
6
8
10
*/
```

Try It Yourself

A **for** loop is best when the starting and ending numbers are known.

do...while Loops

A **do...while** loop is similar to a **while** loop, except that a **do...while** loop is guaranteed to execute at least one time.
Example:

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 5);

/*
1
2
3
4
*/
```

Try It Yourself

Notice that the condition appears at the end of the loop, so the statements in the loop execute once before it is tested.
Even with a false condition, the code will run once. Example:

```
int x = 1;
do {
  System.out.println(x);
  x++;
} while(x < 0);

//Outputs 1
```

Try It Yourself

Loop Control Statements

The **break** and **continue** statements change the loop's execution flow.
The **break** statement terminates the loop and transfers execution to the statement immediately following the loop.
Example:

```
int x = 1;

while(x > 0) {
 System.out.println(x);
  if(x == 4) {
    break;
  }
  x++;
}

/* Outputs
1
2
3
4
*/
```

Try It Yourself

The **continue** statement causes the loop to skip the remainder of its body and then immediately retest its condition prior to reiterating. In other words, it makes the loop skip to its next iteration.
Example:

```
for(int x=10; x<=40; x=x+10) {
  if(x == 30) {
    continue;
  }
  System.out.println(x);
}
/* Outputs
  10
  20
  40
*/
```

Try It Yourself

As you can see, the above code skips the value of 30, as directed by the **continue** statement.