Loops

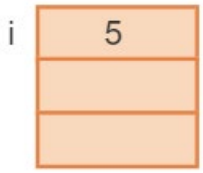
While loop

Three general-purpose looping constructs exist: while, do-while, and for loops. The ***while loop*** is a looping structure that checks if the loop's condition is true before executing the loop body, repeating until the condition is false. The ***loop body*** is the statements that a loop repeatedly executes.

Syntax : while loop.

```
while (condition) {
   body
}
```

```
i = 1;
while (i <= 4) {
   console.log(i);
   i++;
}
console.log("Done!");
```




```
1
2
3
4
Done!
```

Explanation :

1. Assign i with 1.
2. 1 <= 4 is true, so the loop's body executes.
3. Output i to the console.
4. Increment i.
5. End of loop so go back to the top and re-evaluate the condition.
6. Keep executing loop until i <= 4 is false.

Developers must be careful to avoid writing infinite loops. An ***infinite loop*** is a loop that never stops executing. Ex: `while (true);` is an infinite loop because the loop's condition is never false.

JavaScript infinite loop with Chrome's "Page Unresponsive" message

If the web browser is running JavaScript that contains an infinite loop, the web browser tab will cease to respond to user input.

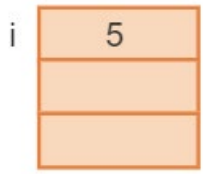
Do-while loop

The ***do-while loop*** executes the loop body before checking the loop's condition to determine if the loop should execute again, repeating until the condition is false.

Syntax do-while loop.

```
do {
   body
} while (condition);
```

```
i = 1;
do {
   console.log(i);
   i++;
} while (i <= 4);
console.log("Done!");
```




```
1
2
3
4
Done!
```

Explanation :

1. Assign i with 1.
2. do..while loop executes the loop body, evaluating the loop condition after the first execution.
3. The loop repeatedly executes until i <= 4 is false.

For loop

A for loop collects three parts — the loop variable initialization, loop condition, and loop variable update — all at the top of the loop. A ***for loop*** executes the initialization expression, evaluates the loop's condition, and executes the loop body if the condition is true. After the loop body executes, the final expression is evaluated, and the loop condition is checked to determine if the loop should execute again.

Syntax : for loop.

```
for (initialization; condition; finalExpression) {
   body
}
```

```
for (i = 1; i <= 4; i++) {
    console.log(i);
}
console.log("Done!");
```


```
1
2
3
4
Done!
```

Explanation :

1. for loop's initial expression assigns i with 1.
2. for loop's condition is then evaluated. 1 <= 4 is true, so the loop's statements are executed.
3. Output i to the console.
4. After the loop executes, the final expression is evaluated, which increments i.
5. Loop repeatedly executes until i <= 4 is false.

break and continue statements

Two *jump* statements alter the normal execution of a loop.
The ***break*** statement breaks out of a loop prematurely.
The ***continue*** statement causes a loop to iterate again without executing the remaining statements in the loop.

```
for (c = 1; c <= 5; c++) {
   if (c == 4) {
      break;    // Leave the loop
   }
   console.log(c);    // 1 2 3 (missing 4 and 5)
}

for (c = 1; c <= 5; c++) {
   if (c == 4) {
      continue;    // Iterate again immediately
   }
   console.log(c);    // 1 2 3 (missing 4) 5
```

```
}
```

Some developers use `break` and `continue` to write short and concise code, but jump statements may introduce subtle logic errors that are difficult to find. This material does not use jump statements.