

## Set 32: *for* Loops

**Skill 32.01:** Explain how a *for* loop works

**Skill 32.02:** Explore special features of the *for* loop

**Skill 32.03:** Write nested *for* loops

**Skill 32.04:** Write a while loop as a *for* loop and vice versa

**Skill 32.01:** Explain how a *for* loop works

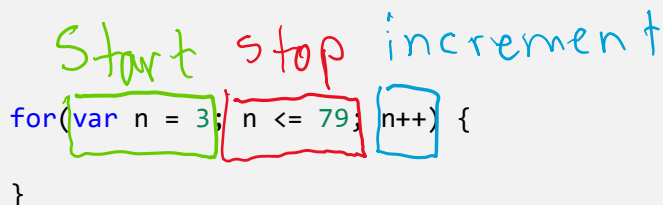
### Skill 32.01 Concepts

One of the most important control structures in JavaScript is the *for*-loop. A *for*-loop is a block of code that is repeated with certain rules about how to start, increment, and end the process.

Suppose for example we want to sum all the integers from 3 to 79. The rules for starting, incrementing, and stopping are as follows,

- **start** = 3
- **incrementing** = +1
- **stop** = 79

This information can be translated to the *for*-loop shown below,



The diagram shows a handwritten *for* loop: `for(var n = 3; n <= 79; n++) {`. Above the code, the words "Start", "stop", and "increment" are written in green, red, and blue respectively. Colored boxes highlight the corresponding parts of the code: a green box around `var n = 3`, a red box around `n <= 79`, and a blue box around `n++`. The closing brace `}` is also present.

Now, to sum all the numbers from 3 to 79, we can let the *for*-loop do the work,

```
var sum = 0;

for(var n = 3; n <= 79; n++) {
    sum += n;
}

console.log(sum); //prints 3157
```

The above example illustrates the following components of a *for*-loop

1. **Initializing expression.** The *for* loop above was initialized by setting `j = 3`. If we had wanted to start summing at 19, this part of the *for* loop would have read `j = 19`.
2. **Control expression.** The control expression, `j <= 79` indicates how long to continue looping. This is a boolean expression. As long as the expression is true, the loop will continue.

**Warning:** There is something really bad that can happen here. You must write your code so as to ensure that this control statement will eventually become false, thus causing the loop to terminate. Otherwise, you will have an endless loop which will crash your program.

3. **Step expression.** The step expression, `j++`, tells us how our variable should change each time through the loop. In this case we are incrementing `j` each time. However, other possibilities could include,
- `j--`
  - `j = j + 4`
  - `j = j * 3`
  - etc.

**Note:** the step expression occurs at the bottom of the loop. Consider the pseudocode below,

```
var sum = 0;

for(var n = 3; n <= 79; ...) {
    sum += n;
    n++; //Just think of the n++ as the last line of code inside the braces
}

console.log(sum); //prints 3157
```

#### [Skill 32.01 Exercises 1 & 2](#)

#### Skill 32.02: Explore special features of the *for* loop

##### Skill 32.02 Concepts

The *break* command: If the keyword *break* is executed inside a *for-loop*, the loop is immediately exited (regardless of the control statement). Execution continues with the statement immediately following the closing brace of the *for-loop*.

Declaring the start loop variable: The start loop variable can be declared in the parenthesis of the *for-loop* or outside the loop. Both situations are illustrated below,

```
var sum = 0;

//The start variable is declared in the parenthesis

for(var n = 3; n <= 79; n++) {
    sum += n;
}

console.log(sum); //prints 3157
```

```
var sum = 0;

//The start variable is declared outside the loop
```

```
var n = 0;

for(n = 3; n <= 79; n++) {
    sum += n;
}

console.log(sum); //prints 3157
```

If *n* is declared in the parenthesis of the loop, its scope is limited to the interior of the loop and is not recognized outside the loop as illustrated below.

```
var sum = 0;

//The start variable is declared in the parenthesis

for(var n = 3; n <= 79; n++) {
    sum += n;
}

console.log(sum); //prints 3157
console.log(n); //Will not compile because n is declare in the loop
```

#### [Skill 32.02 Exercises 1 & 2](#)

#### **Skill 32.03: Write nested for loops**

##### **Skill 32.03 Concepts**

*Nested loops* is the term used when one loop is placed inside another as in the following example,

```
for(var outer = 0; outer < 5; outer++) {

    console.log(outer); //outer is printed 5 times

    for(var inner = 0; inner < 8; inner++){

        console.log(inner); //inner is printed 40 times

    } //end inner loop

} //end outerloop
```

In the example above, the inner loop executes 8 times for each of the five iterations of the outer loop. Therefore, the code inside the inner loop will execute 40 times.

### [Skill 32.03 Exercises 1](#)

#### Skill 32.04: Write a while loop as a for loop and vice versa

##### Skill 32.04 Concepts

The *while* loop you previously learned about is basically the same as the *for-loop*. The only difference is that the initializing and step expressions are not part of the while-loop basic structure. The following code illustrates the basic structure of the while loop.

```
while(n <= 79){  
  //some code we want to repeat  
}
```

In the above code, notice that the only part similar to the for loop is the control expression, `n <= 79`. The initializing and step expression are absent. As with the for-loop, the while-loop keeps repeating as long as the control statement is true.

In a previous example we used a *for-loop* to sum all the numbers from 3 to 79,

```
var sum = 0;  
for(var n = 3; n <= 79; n++) {  
  sum += n;  
}  
console.log(sum); //prints 3157
```

The *for-loop* above requires the following elements to function,

- `start = 3`
- `incrementing = +1`
- `stop = 79`

*while* loops also require these elements. In fact, the *for-loop* above can also be written as a *while* loop as follows,

```
var sum = 0;  
var n = 3;  
while(n <= 79){  
  sum += n;  
  n = +1;  
}  
console.log(sum); //prints 3157
```

#### [Skill 32.04 Exercise 1](#)