Chapter 69: Automatic Semicolon Insertion - ASI

Section 69.1: Avoid semicolon insertion on return statements

The JavaScript coding convention is to place the starting bracket of blocks on the same line of their declaration:

```
if (...) {

}

function (a, b, ...) {

}
```

Instead of in the next line:

```
if (...)
{

}

function (a, b, ...)
{

}
```

This has been adopted to avoid semicolon insertion in return statements that return objects:

```
function foo()
{
    return // A semicolon will be inserted here, making the function return nothing
    {
        foo: 'foo'
    };
}

foo(); // undefined

function properFoo() {
    return {
        foo: 'foo'
    };
}

properFoo(); // { foo: 'foo' }
```

In most languages the placement of the starting bracket is just a matter of personal preference, as it has no real impact on the execution of the code. In JavaScript, as you've seen, placing the initial bracket in the next line can lead to silent errors.

Section 69.2: Rules of Automatic Semicolon Insertion

> There are three basic rules of semicolon insertion:

> 1. When, as the program is parsed from left to right, a token (called the *offending token*) is encountered that is not allowed by any production of the grammar, then a semicolon is automatically inserted before the offending token if one or more of the following conditions is true:
>
> - The offending token is separated from the previous token by at least one `LineTerminator`.
> - The offending token is `}`.
>
> 2. When, as the program is parsed from left to right, the end of the input stream of tokens is encountered and the parser is unable to parse the input token stream as a single complete ECMAScript `Program`, then a semicolon is automatically inserted at the end of the input stream.
> 3. When, as the program is parsed from left to right, a token is encountered that is allowed by some production of the grammar, but the production is a *restricted production* and the token would be the first token for a terminal or nonterminal immediately following the annotation "[no `LineTerminator` here]" within the restricted production (and therefore such a token is called a restricted token), and the restricted token is separated from the previous token by at least one `LineTerminator`, then a semicolon is automatically inserted before the restricted token.
>
> However, there is an additional overriding condition on the preceding rules: a semicolon is never inserted automatically if the semicolon would then be parsed as an empty statement or if that semicolon would become one of the two semicolons in the header of a **`for`** statement (see 12.6.3).

Source: [ECMA-262, Fifth Edition ECMAScript Specification:]

Section 69.3: Statements affected by automatic semicolon insertion

- empty statement
- **`var`** statement
- expression statement
- `do-while` statement
- **`continue`** statement
- **`break`** statement
- **`return`** statement
- **`throw`** statement

Examples:

When the end of the input stream of tokens is encountered and the parser is unable to parse the input token stream as a single complete Program, then a semicolon is automatically inserted at the end of the input stream.

```
a = b
++c
// is transformed to:
a = b;
++c;
```

```
x
++
y
// is transformed to:
x;
++y;
```

Array indexing/literals

```
console.log("Hello, World")
[1,2,3].join()
// is transformed to:
console.log("Hello, World")[(1, 2, 3)].join();
```

Return statement:

```
return
  "something";
// is transformed to
return;
  "something";
```