4.14.2 The +, –, and ~ operators

These operators permit their operand to be of any type and produce a result of the Number primitive type.

The unary + operator can conveniently be used to convert a value of any type to the Number primitive type:

```
function getValue() { ... }

var n = +getValue();
```

The example above converts the result of 'getValue()' to a number if it isn't a number already. The type inferred for 'n' is the Number primitive type regardless of the return type of 'getValue'.

4.14.3 The ! operator

The ! operator permits its operand to be of any type and produces a result of the Boolean primitive type.

Two unary ! operators in sequence can conveniently be used to convert a value of any type to the Boolean primitive type:

```
function getValue() { ... }

var b = !!getValue();
```

The example above converts the result of 'getValue()' to a Boolean if it isn't a Boolean already. The type inferred for 'b' is the Boolean primitive type regardless of the return type of 'getValue'.

4.14.4 The delete Operator

The 'delete' operator takes an operand of any type and produces a result of the Boolean primitive type.

4.14.5 The void Operator

The 'void' operator takes an operand of any type and produces the value 'undefined'. The type of the result is the Undefined type (3.2.6).

4.14.6 The typeof Operator

The 'typeof' operator takes an operand of any type and produces a value of the String primitive type. In positions where a type is expected, 'typeof' can also be used in a type query (section 3.6.3) to produce the type of an expression.

```
var x = 5;
var y = typeof x;  // Use in an expression
var z: typeof x;   // Use in a type query
```

In the example above, 'x' is of type 'number', 'y' is of type 'string' because when used in an expression, 'typeof' produces a value of type string (in this case the string "number"), and 'z' is of type 'number' because when used in a type query, 'typeof' obtains the type of an expression.