```
var x = 1;
function f(a = x) {
    var x = "hello";
}
```

the local variable ‘x’ is in scope in the parameter initializer (thus hiding the outer ‘x’), but it is an error to reference it because it will always be uninitialized at the time the parameter initializer is evaluated.

6.4 Generic Functions

A function implementation may include type parameters in its signature (section 3.7.2.1) and is then called a ***generic function***. Type parameters provide a mechanism for expressing relationships between parameter and return types in call operations. Type parameters have no run-time representation—they are purely a compile-time construct.

Type parameters declared in the signature of a function implementation are in scope in the signature and body of that function implementation.

The following is an example of a generic function:

```
interface Comparable<T> {
    localeCompare(other: T): number;
}

function compare<T extends Comparable<T>>(x: T, y: T): number {
    if (x == null) return y == null ? 0 : -1;
    if (y == null) return 1;
    return x.localeCompare(y);
}
```

Note that the ‘x’ and ‘y’ parameters are known to be subtypes of the constraint ‘Comparable<T>’ and therefore have a ‘compareTo’ member. This is described further in section 3.4.1.

The type arguments of a call to a generic function may be explicitly specified in a call operation or may, when possible, be inferred (section 4.12.2) from the types of the regular arguments in the call. In the example

```
class Person {
    name: string;
    localeCompare(other: Person) {
        return compare(this.name, other.name);
    }
}
```

the type argument to ‘compare’ is automatically inferred to be the String type because the two arguments are strings.