```
function f(x: number) {
    if (x <= 0) return x;
    return g(x);
}

function g(x: number) {
    return f(x - 1);
}
```

the inferred return type for ‘f’ and ‘g’ is Any because the functions reference themselves through a cycle with no return type annotations. Adding an explicit return type ‘number’ to either breaks the cycle and causes the return type ‘number’ to be inferred for the other.

An explicitly typed function returning a non-void type must have at least one return statement somewhere in its body. An exception to this rule is if the function implementation consists of a single ‘throw’ statement.

The type of ‘this’ in a function implementation is the Any type.

In the signature of a function implementation, a parameter can be marked optional by following it with an initializer. An optional parameter with an initializer but no type annotation has its type inferred from the initializer. Specifically, the type of such a parameter is the widened form of the type of the initializer expression.

Initializer expressions are evaluated in the scope of the function body but are not permitted to reference local variables and are only permitted to access parameters that are declared to the left of the parameter they initialize.

For each parameter with an initializer, a statement that substitutes the default value for an omitted argument is included in the generated JavaScript, as described in section 6.5. The example

```
function strange(x: number, y = x * 2, z = x + y) {
    return z;
}
```

generates JavaScript that is equivalent to

```
function strange(x, y, z) {
    if (typeof y === "undefined") { y = x * 2; }
    if (typeof z === "undefined") { z = x + y; }
    return z;
}
```

In the example