Function and Inference Return Types

Inference with many Types

Functions can return an implicit type or an explicit type. By default, TypeScript returns a void type. However, if not specified and the code returns a number, the return type would implicitly be a number. If later the function returns a string, what is the expected return value? The lack of clarity can lead to confusion while having the flexibility of returning an array of potential types.

It is possible to return more than a single type by using a union and still restrict potential return type. The unpredictability of the return type is the reason why having an explicit return type is always a better practice than relying on inference. Explicitly mentioning the type defines a clear contract that cannot change without altering the return type manually.

```
function withImplicitReturnType(b: boolean) {
    if (b) {
        return 10;
    }
    return "test";
}
console.log(withImplicitReturnType(true));
```

The example above returns the implicit type of a union of `10` and the string literal “test”: `10 | "test"`. The reason is because of **line 3** and **line 5** that return at two different places, two different types.

The problem is that in the future an `else` condition could alter the function to infer a new return type: `number | string | boolean` as shown in the following example.

```
function withImplicitReturnType(b: boolean) {
    if (b) {
        return 10;
    } else {
        return true;
    }
    return "test";
}
console.log(withImplicitReturnType(true));
```

Function Inference and Evolution of Returns

Inferring the return type can be a shortcut to avoid writing explicitly what type is expected. However, it has the cost of making the code harder to understand. While the previous examples are short function, and that the good practice guide developer to write small function, the reality is different. Big functions are part of the reality and having to browse the whole function to gather all return type statements is not optimal.

I suggest writing the type explicitly to avoid any potential future misunderstanding and to document your code at the same time.