```
vote("BigPig",
     function(result: string) {
         if (result === "BigPig") {
            // ...
         }
     }
);
```

In this example, the second parameter to 'vote' has the function type

```
(result: string) => any
```

which means the second parameter is a function returning type 'any' that has a single parameter of type 'string' named 'result'.

Section 3.7.2 provides additional information about function types.

1.3 Object Types

TypeScript programmers use *object types* to declare their expectations of object behavior. The following code uses an *object type literal* to specify the return type of the 'MakePoint' function.

```
var MakePoint: () => {
    x: number; y: number;
};
```

Programmers can give names to object types; we call named object types *interfaces*. For example, in the following code, an interface declares one required field (name) and one optional field (favoriteColor).

```
interface Friend {
    name: string;
    favoriteColor?: string;
}

function add(friend: Friend) {
    var name = friend.name;
}

add({ name: "Fred" });  // Ok
add({ favoriteColor: "blue" });  // Error, name required
add({ name: "Jill", favoriteColor: "green" });  // Ok
```

TypeScript object types model the diversity of behaviors that a JavaScript object can exhibit. For example, the jQuery library defines an object, '$', that has methods, such as 'get' (which sends an Ajax message), and fields, such as 'browser' (which gives browser vendor information). However, jQuery clients can also call '$' as a function. The behavior of this function depends on the type of parameters passed to the function.