TypeScript Namespaces and Modules:

Namespaces are just named JavaScript objects that exist globally, and can span multiple files. This makes them a good way to structure code in a web application that has all its dependencies included in an HTML page. Modules are the same, though the main difference is that the latter declares its dependencies rather than putting them in an HTML page. Module dependencies are added on a loader, which brings maintainability and long-term modularity. The three ways to locate the type information for a module are finding its .ts file, finding its .d.ts file, or reading its ambient module declaration. The last one associates a quoted name when declaring a module. A key feature of modules is that different modules will never contribute to the same scope. It is important to not namespace module declarations, because while namespaces provide logical groupings and prevents name collisions, modules already do so. Therefore, it is unnecessary. To conclude, namespaces and modules are beneficial to TypeScript as they wrap up functions and prevent collisions, so two functions in two namespaces sharing a name will not be confused with each other. This emulates the idea of encapsulation in object-oriented design, except for web languages.

Functions:

As TypeScript is a superscript of JavaScript, there are lots of similarities between the two, however, when writing the former, there are differences that will compile into the latter. Functions in general build layers of abstraction, mimicking classes, information hiding, and module functionality. TypeScript functions share this with JavaScript, but they have a key role in how it accomplishes what functions set out to do, adding new capability to JavaScript functions. Function types in TypeScript have two parts, the type of the arguments and the return type. The parameters are identified by type, so any name can be used, while return types are made clear with a '=>' symbol. TypeScript has type inference in the form of contextual typing, which helps cut down work needed to keep the program typed. Contextual typing happens when assigning a function to a variable with a type, but it only has types on one side. Another feature of TypeScript is that every parameter is assumed to be required in a function, which will be checked by the compiler. Therefore, the number of parameters passed in, need to match the number assigned to the function. However, TypeScript allows optional parameters, and the ability to set a default if a parameter is not passed in. It also allows a user to pass in a list of parameters, so that the function can use them as a group rather than individually. These are called rest parameters and can be any size, even empty. TypeScript also solves the problem of using this in JavaScript. TypeScript allows functions to bind to the right ‘this’ to use, simplifying the problem of having to know about what context ‘this’ is when using it in a function. To conclude, while TypeScript builds off of JavaScript, it simplifies writing functions, which will compile into the more complex JavaScript at runtime. It allows for clearer typing than JavaScript, which makes it easier to maintain typed programs.