TypeScript modules are a way to organize code into reusable and maintainable units. Modules allow you to encapsulate related functionality, variables, classes, or interfaces into separate files, making your code easier to manage and understand. TypeScript supports both ES6-style modules (using `import` and `export` statements) and CommonJS-style modules (using `require()` and `module.exports`).

Here's a basic overview of how to work with modules in TypeScript:

### Exporting from a Module

You can export functions, variables, classes, or interfaces from a module using the `export` keyword.

Example:

```typescript
// math.ts
export const sum = (a: number, b: number): number => {
    return a + b;
};

export function multiply(a: number, b: number): number {
    return a * b;
}

export class Calculator {
    add(a: number, b: number): number {
        return a + b;
    }
}
```

### Importing from a Module

To use exported items from a module, you can import them into another file using the `import` keyword.

Example:

```typescript
// app.ts
import { sum, multiply, Calculator } from './math';

const result1 = sum(5, 3);
```

```
console.log(result1); // Output: 8

const result2 = multiply(4, 2);
console.log(result2); // Output: 8

const calc = new Calculator();
const result3 = calc.add(10, 5);
console.log(result3); // Output: 15
```

### Default Exports

You can also export a single default item from a module using the `export default` syntax.

Example:

```typescript
// greetings.ts
const greet = (name: string): string => {
   return `Hello, ${name}!`;
};

export default greet;
```

### Importing Default Exports

When importing a default export, you can give it any name you like.

Example:

```typescript
// app.ts
import myGreet from './greetings';

const message = myGreet('John');
console.log(message); // Output: Hello, John!
```

### Re-exports

You can also re-export items from one module to another using the `export ... from` syntax.

Example:

```typescript
// utils.ts
export function isEven(n: number): boolean {
    return n % 2 === 0;
}

// mathUtils.ts
export { isEven } from './utils';
```

### Dynamic Imports

TypeScript also supports dynamic imports, which allow you to load modules dynamically at runtime.

Example:

```typescript
// app.ts
const mathModule = await import('./math');
const result = mathModule.sum(10, 20);
console.log(result); // Output: 30
```

### Conclusion

Modules in TypeScript help you organize your code into reusable and maintainable units. By using `export` and `import` statements, you can create a modular and scalable codebase that is easy to understand and maintain.

TypeScript has several built-in modules that provide essential functionality for working with different environments, such as the browser or Node.js. Here are some of the key built-in modules in TypeScript:

### Browser Environment Modules

1. **DOM (Document Object Model):** TypeScript provides types for working with the DOM, allowing you to manipulate HTML elements, handle events, and interact with the browser environment more effectively.

Example:
```typescript
```

```
// app.ts
const element = document.getElementById('myElement');
element.addEventListener('click', () => {
    console.log('Element clicked!');
});
```

2. **Web APIs:** TypeScript includes type definitions for various web APIs, such as `fetch`, `localStorage`, `sessionStorage`, `setTimeout`, `setInterval`, etc., making it easier to work with these APIs in a type-safe manner.

Example:
```typescript
// app.ts
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error fetching data:', error));
```

### Node.js Environment Modules

1. **CommonJS Modules (`require` and `module.exports`):** TypeScript supports CommonJS modules for Node.js, allowing you to use `require` to import modules and `module.exports` to export values from a module.

Example:
```typescript
// app.ts
const fs = require('fs');
const data = fs.readFileSync('file.txt', 'utf-8');
console.log(data);
```

2. **Built-in Node.js Modules:** TypeScript provides type definitions for built-in Node.js modules such as `fs` (File System), `path`, `http`, `https`, `os`, etc., enabling you to use these modules with type safety.

Example:
```typescript
// app.ts
import * as fs from 'fs';
const data = fs.readFileSync('file.txt', 'utf-8');
console.log(data);
```

### ES6 Modules

TypeScript also supports ECMAScript 2015 (ES6) modules, which use `import` and `export` statements to define dependencies between modules. ES6 modules are widely supported in modern JavaScript environments.

Example:
```typescript
// math.ts
export function sum(a: number, b: number): number {
    return a + b;
}

// app.ts
import { sum } from './math';
console.log(sum(5, 3)); // Output: 8
```

These are some of the essential built-in modules in TypeScript that provide functionality for working with various environments and modules. You can leverage these modules to develop applications for both the browser and Node.js with TypeScript.