Certainly! TypeScript itself doesn't have "inbuilt" modules in the sense that it provides its own runtime environment or standard library like some other programming languages do. However, TypeScript does come with type definitions (`.d.ts` files) for various JavaScript environments, libraries, and APIs. These type definitions allow TypeScript to provide static type checking and IntelliSense for the corresponding JavaScript code.

Here are some of the commonly used type definitions that TypeScript provides:

1. **DOM (Document Object Model):** Types for working with the browser DOM, including elements, events, and APIs.
2. **Web APIs:** Types for web APIs such as `fetch`, `localStorage`, `sessionStorage`, `setTimeout`, `setInterval`, etc.
3. **Node.js Built-in Modules:** Types for Node.js built-in modules such as `fs` (File System), `path`, `http`, `https`, `os`, etc.
4. **Node.js Global Objects:** Types for global objects available in Node.js, such as `Buffer`, `console`, `process`, `module`, etc.
5. **CommonJS Modules:** Types for CommonJS modules, allowing you to use `require` and `module.exports` in TypeScript.
6. **ES6 Modules:** Types for ECMAScript 2015 (ES6) modules, including `import` and `export` syntax.

These are just a few examples of the type definitions that TypeScript provides. There are many more type definitions available for popular libraries, frameworks, and environments, which you can install using npm or yarn, or they may come bundled with the library itself.

Sure, here are five code examples demonstrating the usage of TypeScript with various built-in modules and environments:

1. DOM (Document Object Model):

```typescript
// Example 1: Accessing DOM elements
const element = document.getElementById('myElement');
element.innerText = 'Hello, TypeScript!';

// Example 2: Handling events
element.addEventListener('click', () => {
    console.log('Element clicked!');
});

// Example 3: Manipulating styles
element.style.color = 'blue';

// Example 4: Creating new elements
```

```
const newElement = document.createElement('div');
newElement.innerText = 'New element';
document.body.appendChild(newElement);

// Example 5: Working with HTML collections
const buttons = document.getElementsByTagName('button');
for (const button of buttons) {
    console.log(button.innerText);
}
```

2. Web APIs:

```typescript
// Example 1: Fetching data from an API
fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error fetching data:', error));

// Example 2: Working with local storage
localStorage.setItem('username', 'John');
const username = localStorage.getItem('username');
console.log('Username:', username);

// Example 3: Using setTimeout
setTimeout(() => {
    console.log('Delayed message');
}, 2000);

// Example 4: Working with query parameters
const urlParams = new URLSearchParams(window.location.search);
const paramValue = urlParams.get('paramName');
console.log('Parameter value:', paramValue);

// Example 5: Managing cookies
document.cookie = 'username=John Doe; expires=Fri, 31 Dec 2021 23:59:59 GMT';
const cookies = document.cookie.split('; ');
console.log('Cookies:', cookies);
```

3. Node.js Built-in Modules:

```typescript
```

```
// Example 1: Reading from a file using fs module
import * as fs from 'fs';
const data = fs.readFileSync('file.txt', 'utf-8');
console.log('File content:', data);

// Example 2: Using path module to manipulate file paths
import * as path from 'path';
const filename = path.basename('/path/to/file.txt');
console.log('Filename:', filename);

// Example 3: Creating an HTTP server
import * as http from 'http';
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World!');
});
server.listen(3000);

// Example 4: Using os module to get system information
import * as os from 'os';
console.log('OS platform:', os.platform());

// Example 5: Working with streams using fs module
const stream = fs.createReadStream('file.txt');
stream.on('data', chunk => {
    console.log('Chunk received:', chunk.toString());
});
```

4. Node.js Global Objects:

```typescript
// Example 1: Accessing process object
console.log('Process ID:', process.pid);
console.log('Node.js version:', process.version);

// Example 2: Accessing buffer module
const buf = Buffer.from('Hello, World!', 'utf-8');
console.log('Buffer:', buf.toString());

// Example 3: Using console object
console.log('Hello, Console!');

// Example 4: Working with module object
```

```
console.log('Main module:', module.filename);
console.log('Module exports:', module.exports);

// Example 5: Working with global object
console.log('Global object:', global);
```

5. CommonJS Modules:

```typescript
// Example 1: Importing from another module
const math = require('./math');
console.log('Sum:', math.sum(5, 3));

// Example 2: Exporting a function from a module
function greet(name: string): string {
    return `Hello, ${name}!`;
}
module.exports = greet;

// Example 3: Importing and using a module
const greet = require('./greet');
console.log(greet('John'));

// Example 4: Exporting an object from a module
const constants = {
    PI: 3.14159,
    E: 2.71828
};
module.exports = constants;

// Example 5: Importing and using a module
const constants = require('./constants');
console.log('Value of PI:', constants.PI);
```

These examples illustrate how to work with TypeScript and various built-in modules and environments, including the DOM, web APIs, Node.js built-in modules, global objects, and CommonJS modules.