Copilot

Certainly! Let's dive into the basics of TypeScript with a bit more detail. I'll provide concise explanations and examples for each topic:

1. **JSON Objects**:
   - JSON (JavaScript Object Notation) is a lightweight data interchange format.
   - It represents data as key-value pairs.
   - Example:

```
const person = {
    name: "Alice",
    age: 30,
    isStudent: false,
};
```

2. **Syntax Error**:
   - A mistake in the code that violates the language rules.
   - Example:

```
// Syntax error: missing semicolon
const greeting = "Hello, world"
```

3. **Type Error**:
   - Occurs when a value is used in a way that doesn't match its type.
   - Example:

```
const num: number = "42"; // Type error: string cannot be assigned to a number
```

4. **Assignability**:
   - Determines if a value can be assigned to a variable.
   - Example:

```
let x: number;
x = 10; // Valid assignment
x = "hello"; // Type error
```

5. **Strong Typing**:
   - TypeScript enforces strict type checking.
   - Example:

```
const age: number = 25;
```

6. **`const` and `let`**:
   - `const`: Immutable variable.
   - `let`: Mutable variable.
   - Example:

```
const pi = 3.14;
let count = 0;
```

7. **Modules**:
   - Organize code into reusable files.
   - Example:

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

8. **Native ECMAScript Modules**:
    - Use `import` and `export` for modular code.
    - Example:

```
// main.ts
import { add } from "./math";
console.log(add(2, 3)); // Output: 5
```

9. **Union Literals**:
    - Combine multiple types into one.
    - Example:

```
type Status = "active" | "inactive";
const userStatus: Status = "active";
```

10. **Objects**:
    - Represent structured data.
    - Example:

```
const person = {
    name: "Bob",
    age: 28,
};
```

11. **Object Aliased**:
    - Create custom type aliases.
    - Example:

```
type Point = { x: number; y: number };
const origin: Point = { x: 0, y: 0 };
```

12. **Structural Object Literals**:
    - Create objects with specific properties.
    - Example:

```
function printCoordinates(point: { x: number; y: number }) {
    console.log(`(${point.x}, ${point.y})`);
}
```

13. **Nested Objects**:
    - Objects containing other objects.
    - Example:

```
const address = {
    street: "123 Main St",
    city: "Seattle",
    zip: "98101",
};
const personWithAddress = {
    name: "Alice",
    address: address,
};
```

14. **Intersection Types**:

- Combine multiple types.
- Example:

```
type Admin = { role: "admin" };
type User = { role: "user" };
type AdminUser = Admin & User;
```

15. **`any`, `unknown`, `never` Types**:

- `any`: No type checking.
- `unknown`: Type-safe `any`.
- `never`: Represents unreachable code.
- Example:

```
let value: any = 42;
let data: unknown = "hello";
function throwError(): never {
    throw new Error("Something went wrong");
}
```

16. **Explicit Casting**:

- Convert one type to another.
- Example:

```
const length: number = (userInput as string).length;
```

17. **Enum**:

- Enumerated values.
- Example:

```
enum Color {
    Red,
    Green,
    Blue,
}
const selectedColor: Color = Color.Green;
```

18. **`const enum`**:

- Compile-time constants.
- Example:

```
const enum Direction {
    Up,
    Down,
    Left,
    Right,
}
```