

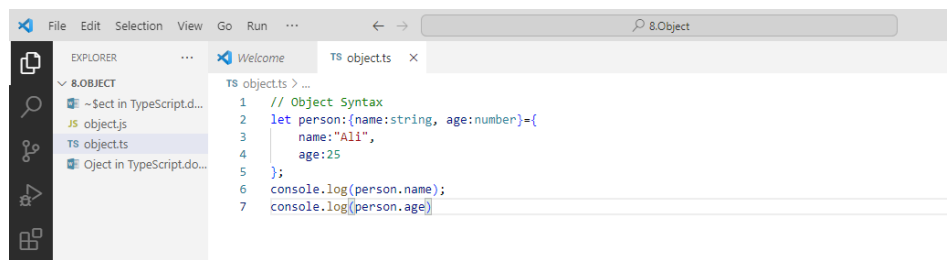
Object in TypeScript

An object is a fundamental data type that represents a collection of key-value pairs, where keys are strings or symbols, and values can be of any data type, including other objects. Objects in TypeScript are similar to objects in JavaScript, but TypeScript adds static typing to enhance code quality and maintainability.

There are different ways to define objects in TypeScript:

Object Literal Syntax:

The Object Literal Syntax in TypeScript allows you to define and create objects using a concise and straightforward syntax. It involves specifying key-value pairs within curly braces { }, where keys are typically strings or symbols, and values can be of any data type, including other objects.

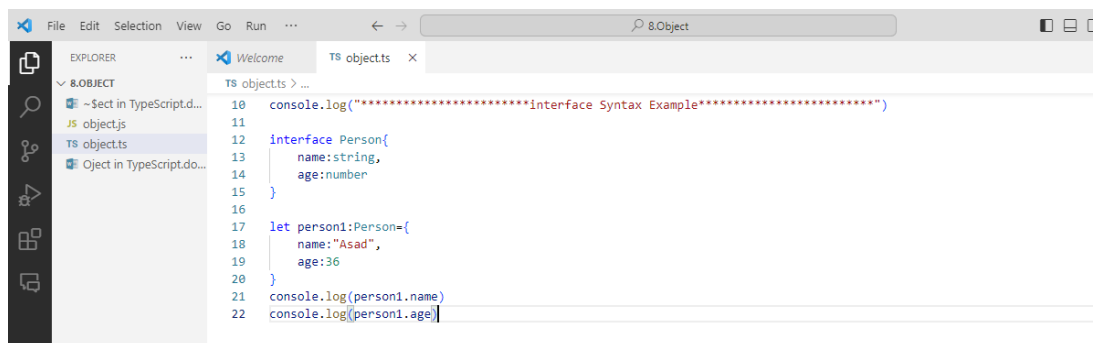


A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a project named '8.OBJECT' with files like 'TS objects'. The main editor window is open to 'TS objects.ts' and contains the following TypeScript code:

```
1 // Object Syntax
2 let person:{name:string, age:number}={
3   name:"Ali",
4   age:25
5 };
6 console.log(person.name);
7 console.log(person.age);
```

Interface:

You can use an interface to define the shape of an object and then create objects that adhere to that interface.

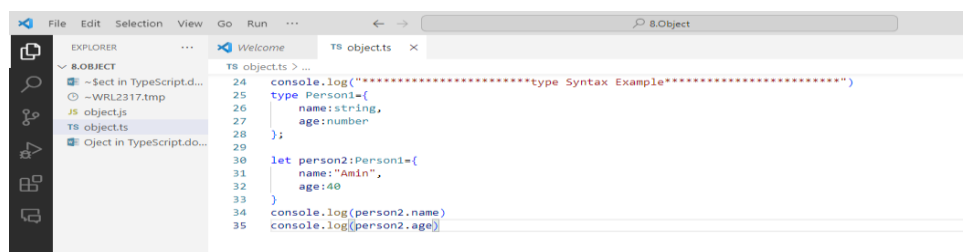


A screenshot of the Visual Studio Code editor showing the 'TS objects.ts' file. The code defines an interface and creates an object that conforms to it:

```
10 console.log("*****Interface Syntax Example*****")
11
12 interface Person{
13   name:string,
14   age:number
15 }
16
17 let person1:Person={
18   name:"Asad",
19   age:36
20 }
21 console.log(person1.name)
22 console.log(person1.age);
```

Type:

Similar to interfaces, you can use the type keyword to define the shape of an object, but assignment operator additionally used before start of curly braces.



A screenshot of the Visual Studio Code editor showing the 'TS objects.ts' file. The code defines a type and creates an object that conforms to it:

```
24 console.log("*****type Syntax Example*****")
25 type Person1={
26   name:string,
27   age:number
28 };
29
30 let person2:Person1={
31   name:"Amin",
32   age:40
33 }
34 console.log(person2.name)
35 console.log(person2.age);
```

What is a nested object?

Imagine you have an object, which is like a box that can hold information. Now, inside that box, you can have another smaller box. That smaller box is also an object. So, a nested object is basically an object that lives inside another object.

Advantage of nested object:

Let's say you want to organize information in a structured way. Instead of having everything in one big box, you can have smaller boxes (nested objects) inside the main box. This helps you represent information in a more organized and hierarchical manner, like building a tree of information

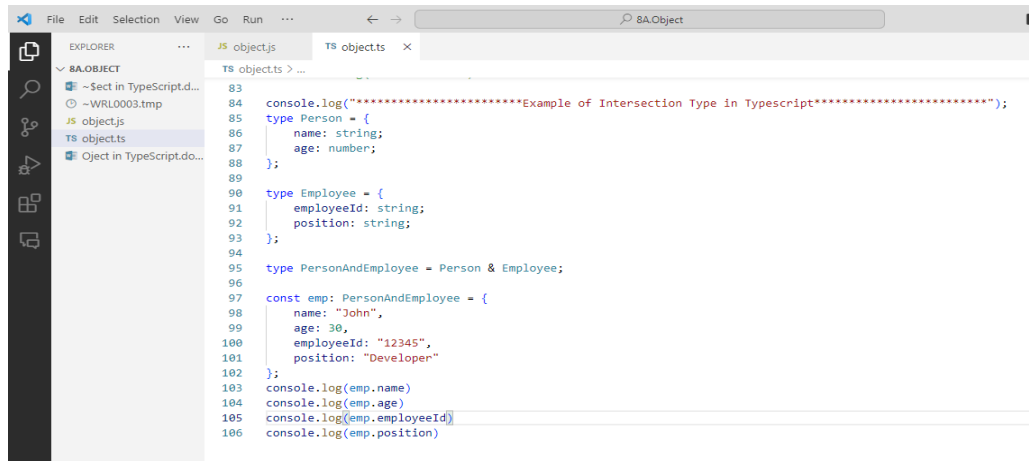
In TypeScript, a nested object refers to an object that is a property of another object. This means that the value associated with a certain key in an object is itself another object. This allows you to represent hierarchical or structured data in your TypeScript code.

```
37 console.log('*****Example of Nested object in Typescript*****');
38
39 type Author = {
40   firstName: string;
41   lastName: string;
42 };
43
44 type Book = {
45   author: Author;
46   name: string;
47 };
48 const myBook : Book = {
49   author: {
50     firstName: 'Abdul',
51     lastName: 'Nasir'
52   },
53   name: 'My Best Book'
54 };
55 console.log(myBook.name);
56 console.log(myBook.author);
```

```
58 console.log('*****2nd Example of Nested object in Typescript*****');
59 type Student = {
60   name: string;
61   age: number;
62   adress: {
63     street: string;
64     city: string;
65     postalCode: number;
66   };
67 };
68
69 let student: Student = {
70   name: 'Abdul Nasir',
71   age: 36,
72   adress: {
73     street: 'No 1 ',
74     city: 'Karachi',
75     postalCode: 23520
76   };
77 };
78
79 console.log(student.name);
80 console.log(student.age);
81 console.log(student.adress);
```

Intersection Types:

You can use intersection types (&) to combine multiple object types into a single type that has all the properties of each constituent type. This allows you to create more specific and complex types.



The screenshot shows a Visual Studio Code editor window with a TypeScript file named 'TS objects.ts'. The Explorer sidebar on the left shows the file structure. The main editor area contains the following code:

```
83
84 console.log("*****Example of Intersection Type in Typescript*****");
85 type Person = {
86   name: string;
87   age: number;
88 };
89
90 type Employee = {
91   employeeId: string;
92   position: string;
93 };
94
95 type PersonAndEmployee = Person & Employee;
96
97 const emp: PersonAndEmployee = {
98   name: "John",
99   age: 30,
100   employeeId: "12345",
101   position: "Developer"
102 };
103 console.log(emp.name)
104 console.log(emp.age)
105 console.log(emp.employeeId)
106 console.log(emp.position)
```