TypeScript

Table Of Content

- Partial Type
- Required Type
- Readonly
- Pick
- Omit
- Extract
- Exclude
- Record
- NonNullable
- Type guards
    - `typeof`
    - `instanceof`
    - `in`

More on TypeScript

- It is a pure Object oriented language
  - It has classes, interfaces, statically typed
  - Like C# or Java
- Supports all JS libraries and frameworks
- Javascript is basically Typescript
  - That means you can rename any valid `.js` file to `.ts`
- TypeScript is aligned with ES6
  - It has all features like modules, classes, etc

Why use TypeScript?

- It is superior to its counterparts
  - Like Coffeescript and Dart
  - TypeScript extends JavaScript - they do not
  - They are different language altogether
  - They need language-specific execution env to run
- It gives compilation errors and syntax errors
- Strong static typing
- It supports OOP
  - Classes, interfaces, inheritance, etc.
  - Like Java, c#
- Reacher code hinting due to its typed nature
- Automated documentation

    - Due to its typed nature
    - Therefore, good readability

- No need of custom validation which is clunky for large apps

    - No boilerplate code

Variable scopes

- Global scope
  - Declare outside the programming constructs
  - Accessed from anywhere within your code

```
const name = "sleepless yogi"

function printName() {
  console.log(name)
}
```

- Class scope
  - Also called fields
  - Accessed using object of class
  - Static fields are also available... accessed using class name

```
class User {
  name = "sleepless yogi"
}

const yogi = new User()

console.log(yogi.name)
```

- Local scope
  - Declared within methods, loops, etc ..
  - Accessible only withing the construct where they are declared
  - `vars` are function scoped
  - `let` are block scoped

```
for (let i = 0; i < 10; i++) {
  console.log(i)
}
```

Pick

- Use this to create new type from an existing type `T`
- Select subset of properties from type `T`
- In the below example we create `NewCustomer` type by selecting `id` and `name` from type `Customer`

```
type Customer {
  id: string
  name: string
  age: number
}

type NewCustomer = Pick<Customer, "id" | "name">


function setCustomer(customer: Customer) {
  // logic
}


setCustomer({ id: "id-1", name: "NgNinja Academy" })
// valid call


setCustomer({ id: "id-1", name: "NgNinja Academy", age: 10 })
```

```
// Error: `age` does not exist on type NewCustomer
```

- They allow you to check the type of variables
- You can check it using different operators mentioned below

typeof

- Checks if the type of the argument is of the expected type

```
if (typeof x === "number") {
  // YES! number logic
}
else {
  // No! not a number logic
}
```

instanceof

- Checks if the variable is instance of the given object/class
- This is mostly used with non-primitive objects

```
class Task {
  // class members
}

class Person {
  // class members
}
```

```
const myTasks = new Task()
const myPerson = new Person()

console.log(myTasks instanceof Task) // true
console.log(myPerson instanceof Person) // false
```

in

- Allows you to check if property is present in the given type

```
type Task {
  taskId: string
  description: string
}

console.log("taskId" in Task) // true
console.log("dueDate" in Task) // false
```