**TypeScript**

* TypeScript is a Superset of JavaScript-> A language building up on JavaScript by adding new features + Advantages to JavaScript.
* Browser can’t directly execute TypeScript
* TypeScript is a tool which compiles TypeScript code into JavaScript code.
* TypeScript will check error during the compilation process, that’s why compile time error will not generated directly on browser
* TypeScript is a powerful, open-source programming language. It is built on JavaScript by adding static type definition, making it a superset of JavaScript.
* TypeScript enhances the development experience by enabling developers to catch errors in early stage through the type checking and it facilitates the development of large-scale applications with improved code quality and maintainability.

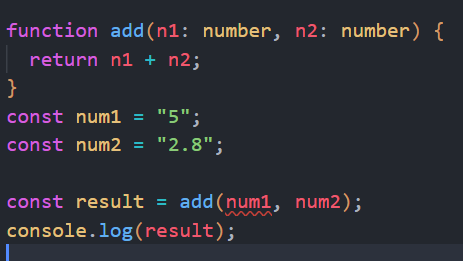
A diagram of a computer program

Description automatically generated with medium confidence

* **Key Features of TypeScript:**

1. **Static Typing:**

* TypeScript’s type system helps catch errors at compile time, reducing runtime errors and improving code reliability.



1. **Better Code Readability and Maintainability:**

* Type definition and interface make the code more understandable and easier to maintain.

1. **Adding Non-JavaScript Features like interface or Generics**
2. **It also gives Meta-programming Feature like Decorators**
3. **Rich configuration Options.**

* **Data Types:**

1. **Number:**

* All numbers, no differentiation between integers or float
* 1, 5.3, -10

1. **String:**

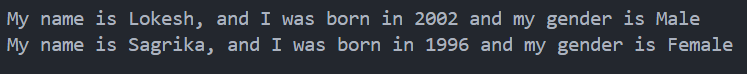
* All text values

1. **Boolean:**

* Just these two, no “truthy” or “falsy” values
* True or false

**A computer screen shot of text

Description automatically generated**

****

1. **Object:**

* Any JavaScript object, more specific types are possible
* {age:30}

1. **Array:**

* Any JavaScript array, with more specific types declaration
* [1,2,3]

1. **Tuple:**

* Tuple is fixed length array
* [number, string] = [1, “Lokesh”]

1. **Enum:**

* Automatically enumerated global constant identifiers
* enum {New, Old}
* It gives us numerated index which starting at 0

1. **any:**

* Any can used to assign any type of value in variables
* It is not type safe
* We should avoid use of any in TypeScript
* const variable: any [] = [1, true, “Lokesh”]
* **JavaScript vs TypeScript:**

|  |  |
| --- | --- |
| **TypeScript** | **JavaScript** |
| 1. TypeScript is a superset of JavaScript | 1. JavaScript is a subset of TypeScript |
| 1. Ts provide static typing checking | 1. Js provide dynamically typed |
| 1. Syntax is similar to JS with additional features | 1. Standard JavaScript syntax |
| 1. Stronger typing can help identify errors | 1. May require more debugging and testing |
| 1. Adding non-js feature like interface or generics | 1. Interface or generics are not present in js. |
| 1. In TS Error will identify during compilation phase | 1. In JS error will identify during runtime phase |

* **Type Assignment & type inference:**

**A screenshot of a computer program

Description automatically generated**

* If we assign variable with value at a time of declaration, then no need to explicitly mention **type** for that variable.
* If we only declare variables without initializing value and we want to initialize value later then we should mention **type** for that variable.
* **Type Inference:**
* Type inference allows the compiler to automatically determine data types of variables, functions return type, Object types and Array types based on the context in which they are used.
* TypeScript uses inference extensively to provide type safety while still maintaining the flexibility of JavaScript.
* **Type Assignment:**
* In Type Assignment we need to explicitly defined type of variable
* So typescript compiler can easily understand what type of data is present in that variable.
* **Objects:**
* An **object** in typescript is an instance that contains a set of key-value pairs.
* These pairs can hold various data types, including scalar values, functions, or even arrays of other objects.
* Typescript object refers to any value that isn’t primitive. This is different from the empty object type **{}**, and also different from the global type Object.

**Syntax:**let Name\_of\_object = {

object\_property : value,

}

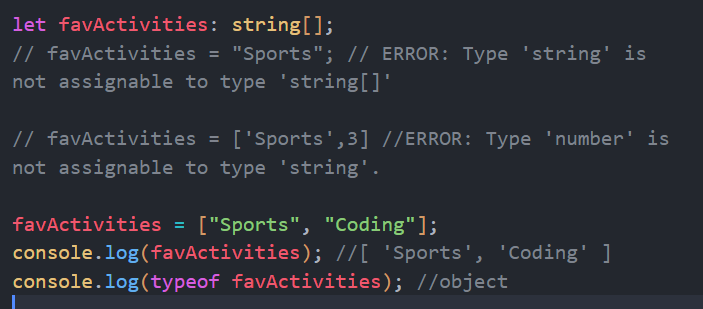
* Eg:  
  A computer screen shot of text

  Description automatically generated
* We can explicitly mention type of field while declaring object with object property or we can directly declare object than TypeScript automatically like this.

**A screen shot of a computer code

Description automatically generated**

* TypeScript detect property in compile time which is not exist in object
* **Array:**
* An array is a user defined data type.
* An array is a homogeneous collection of similar types of elements that have a contiguous memory location, and which can store multiple values of different data types.
* We can explicitly mention type of array and this array can only store that type of values.



* If we want to multiple type of array then we can use **any** of simply mention that types.

**A computer code with colorful text

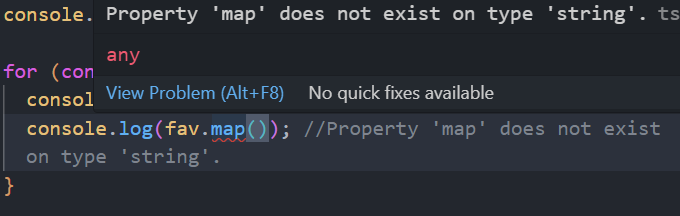
Description automatically generated with medium confidence**

* If we iterate over an array element than TypeScript help us by suggesting supported method on that element

**A screen shot of a computer code

Description automatically generated**

If we use another method which is not supported for this elements, then it will throw error on compile time.



* We can use array for code optimization and for randomly access elements.
* **Tuples:**
* As we know array consists of values of homogeneous (same) types but sometimes when we need to store a collection of different types value in a single variable, then we will go with **Tuples.**
* Tuples may be one or more than one type of data.
* Eg:

A computer screen shot of text

Description automatically generated

* Tuples have a **fixed size,** and we cannot store element more than its size.
* Also, we can store value of **fixed type,** we cannot store number in string place.
* Even we cannot assign empty array to tuple variable.
* **Enum:**
* TypeScript **Enum Type** is not natively supported by JavaScript, but it’s a notable feature of the TypeScript Programming Language.
* In general, enumeration shortly called **enum.**
* Enums allow us to create a set of named constant, Making it easier for us by being self-explanatory while declaring a set of predefined value, by default enum variables has value form **0**.

A computer screen shot of a computer code

Description automatically generated

Eg:

A screen shot of a computer

Description automatically generated

* While Enums are constant by default that’s why we cannot reassign value to enum variables after declaring it.
* Enum is user defined type that’s why we can access enum variable using **(.)** operator.
* **Any:**
* **any** type is a dynamic type that can represent values of any data type.
* It allows for flexible typing but sacrifice type safety, as it lacks compile-time checking
* We can avoid use of any in TypeScript because it affect big feature of TypeScript which is **type safety.**

A screen shot of a computer code

Description automatically generated

* **Union Type:**
* The TypeScript union has the ability to combine one or two different types of data.
* It is the most powerful way to express a variable with multiple types.
* Use **pipe (‘|’)** symbol to combine two or more data types to achieve union type.

A black and white text

Description automatically generated

Eg:

A screen shot of a computer program

Description automatically generated

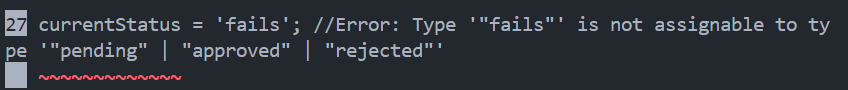
* **Literal Types:**
* The string literal type was added in TypeScript version 1.8.
* String literal types work well with union types and type aliases in practice. These properties can be combined to give strings enum-like functionality.
* The string literal type allows you to specify a set of possible string values for a variable, only those string values can be assigned to a variable.
* TypeScript throws a compile-time error if we tries to assign a value to the variable that isn’t defined by the string literal type
* Syntax:

**Let** variableName : **‘stringLiteral’;**

* Eg:

A screen shot of a computer code

Description automatically generated



* **Alias type:**
* Type alias give a type a new name.
* They are similar to interfaces in that they can be used to name primitives and any other kinds that you’d have to define by hand otherwise.
* Aliasing doesn’t create a new type, instead , it gives that type a new name.

A screen shot of a computer program

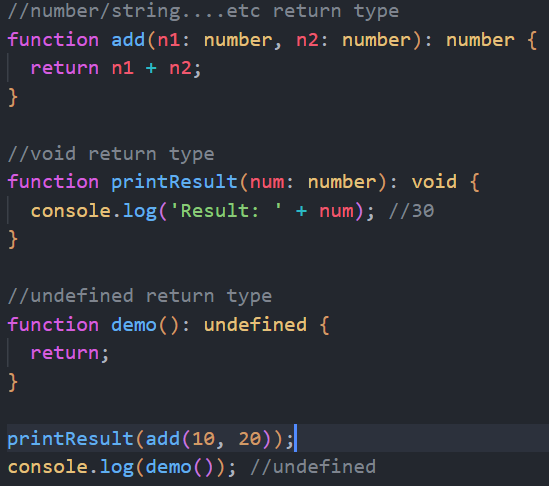
Description automatically generated

* String literals as type alias.

A screen shot of a computer program

Description automatically generated

* **Function Return Type:**
* TypeScript support function with return types where JavaScript does not support functions with return type

Eg:  


* There are 3 major return type
  + Number, string, Boolean …. etc.
  + Void return type: if we want to print something than we use void
  + Undefined return type: if we want to return nothing or undefined value then we use this
* If we want to return nothing or undefined value, we can also use void
* **Function as a Type:**

A computer screen shot of a code

Description automatically generated

* **Callback Function:**
* Callback function mean passing a function as a arguments to another function.

A computer screen shot of text

Description automatically generated

* **Unknown Type:**
* In TypeScript any value can assign to unknown, but without a type assertion, unknown can’t be assigned to anything.
* **Unknown** type is used to make our code type-safe, when we deal with unknown type.
* it looks like **any** type but it is different, **any** is fully flexible and not type safe while **unknown** if partially flexible and type safe.
* If we declare two variables one with **unknown** and other with **string/number** type and store string in **unknown type** variable and then try to assign **unknown type** variable in **string** type variable then it will throw error

A screen shot of a computer program

Description automatically generated

* Unknown type variable can only be assigned to another unknown type variable or a variable of type **any,** unknown type is displayed as **undefined.**
* **Never type:**
* When we are certain that a particular situation will never happen, we use the never type.
* Eg: suppose we construct a function that never returns or always throws an exception then we can use the **never** type on than function.
* Neve is a new type in TypeScript that denotes values that will never be encountered.

A computer code with colorful text

Description automatically generated

This function doesn’t return anything.

* **null** value cannot be assigned to **never type.**
* We are not able to store any value in **never type** variables.

A screen shot of a computer code

Description automatically generated

* **NOTE:**

1. To run TypeScript file in watch mode use following syntax

**tsc filename.ts -w**

**or**

**tsc filename.ts –watch**

1. To compile entire project at one time
   1. **tsc --init 🡪**it will create **tsconfig.json** file for configuration.
   2. **tsc** or **tsc –w**
2. To exclude any file from compilation when we compiling entire project for that we make changes in **tsconfig.json** file
   1. Open **tsconfig.json**  file

**A screen shot of a computer

Description automatically generated**

1. Also there is **include** property in that file which indicate all file which mention in **include** property only compile that files remaining file is not compiled
2. If we want to set folder path for compiled file then enable following property in **tsconfig.json** file

****

* **Intersection Type:**
* An intersection type is a type that merges several kinds into one.
* This allows you to combine many types to create a single type with all of the properties that you require.
* An object of this type will have members from all of the types given.
* The ‘**&**’ operator is used to create the intersection type.

**A screen shot of a computer program

Description automatically generated**

* By intersection multiple type in single type, we can access all properties of all types by assigning only single type to that variable.
* If we do not mention any filed from all types then it will throw error.
* Intersection and union type almost work same but with some difference which is when we declare intersection type with **&** then we need to compulsory used all properties in that object of all types. But when we declare union type then we don’t need to compulsory mention all the field/Properties of all types.

|  |  |
| --- | --- |
| **Intersection type** | **Union type** |
|  |  |
| This will throw the error **& , and** | This is not throwing error **‘|’ , or** |

* **Type Guards:**
* Type Guards are TypeScript features that allow developers to narrow down the type of a variable within a conditional block.
* They are particularly useful when dealing with union types or when the types of variables cannot be determined by TypeScript’s static analysis alone.
* Type Guard provides a way to ensure type safety at runtime, thereby preventing potential errors.
* There are several type of Type Guard:  
  **1. typeOf():**

**A computer screen with white text

Description automatically generated**

* + it can be used to check the type of a variable using the **typeOf** operator.

**2. instanceof:**

- This **instanceof** operator checks if an object is an instance of a particular class or constructor function.

**3. in:**

**-** it is used to check given property present in the object or not

**A screen shot of a computer program

Description automatically generated**

**4. Discriminated Union type:**

- It is also called ‘tagged union type’ or ‘algebraic data types’.

- it is a type that can represents a value that can be one of several different type, with a tag indicating the type of the value.

* **Type Casting:**
* In TypeScript type casting happen by two ways

**A white background with black text

Description automatically generated**

1. **as:**
   * A straightforward way to cast a variable is using the **as** keyword, which will directly change the type of the given variables.
   * **as** is used to set return type of variable
2. **<>:**
   * **It** works the same as casting with **as.**

* **Index type Properties:**
* It is a feature that allow us to create object more flexible regarding the property they might hold.
* We can also set types on object properties/keys
* Then we cannot create property of other type like if we create property with number type then we cannot create property of string type in that variable

**A screen shot of a computer code

Description automatically generated**

* **Function Overloading:**
* Function overloading is the process in which one method could be used multiple times with same name and different parameter types or return types.
* Function name and number of parameters in overloaded functions should be same.

**A screen shot of a computer code

Description automatically generated**

* **Optional Chaining(?.):**
  + The optional chaining is an error-proof way to access nested object properties, even if an intermediate property doesn’t exist.
  + It is similar to **chaining ‘.’** Except that it does not report the error, instead it returns a value which is **undefined**.
  + It also works with a function call when we try to make a call to a method which may not exist.
  + When we want to check a value of the property which is deep inside a tree-like structure, we often must perform check whether intermediate nodes exist.

A screen shot of a computer code

Description automatically generated

* + It is also work on method, it checks method exists or not if exist then it execute that method or return undefined.

A screen shot of a computer code

Description automatically generated

* **Nullish Value:**
  + Nullish value are **null** and **undefined** only it is not contain other falsy value like **0,’ ’, NaN.**
  + It is fix that error

A computer screen shot of text

Description automatically generated