Javascript and Typescript

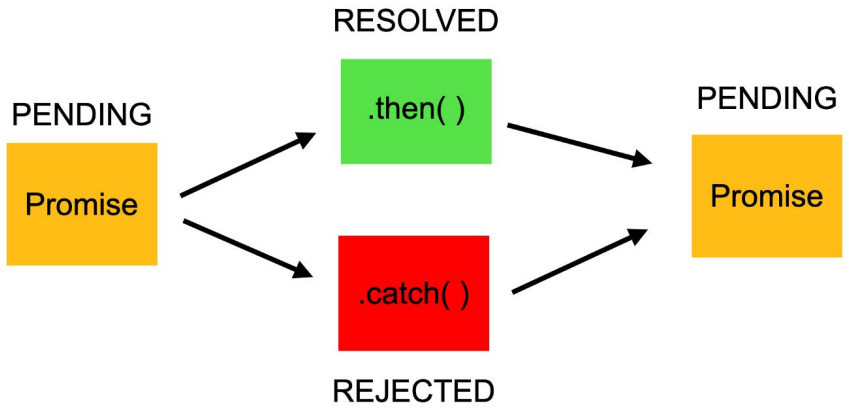
Javascript

- Developed in 1995 by Netscape
- Initially designed to interact with elements of web pages
- Now also used as server-side language
- Ex : Node JS

Promises

A promise is an **object** that may produce a **single value** sometime in the future. Either a **resolved** value or a **rejected** value.

Possible states



Promises

Use a constructor to create a Promise object

```
const myPromise = new Promise();
```

It takes two parameters, one for success (resolve) and one for fail (reject):

```
const myPromise = new Promise((resolve, reject) => {
   // condition
});
```

.then( ) for resolved Promises

.catch( ) for errors or failures

Async / Await

- Introduced in JavaScript Version ES6 - ECMAScript 2015
- When we append the keyword “async” to the function, this function returns the Promise by default on execution
  - The function contains some Asynchronous Execution
  - The returned value will be the Resolved Value for the Promise.
- Capturing Promise ?

Caution: Before getting in details of Async and Await, you should have a good understanding of **Promises** in JavaScript

```
async function asyncPromise() {
    return "I am JS";
}

asyncPromise().then(function(data){
    console.log(data)
});
```

Await

- Promises are asynchronous and waiting on another thread for completion
- JavaScript does not wait for the promise to resolve, it executes further
- Once the promise is resolved the callback function is invoked.
- Adding "**await**" before a promise makes the execution thread to wait for asynchronous task/promise to resolve before proceeding further.
- When we are adding the "**await**" keyword, we are introducing synchronous behavior to the application.
- Even the promises will be executed synchronously.

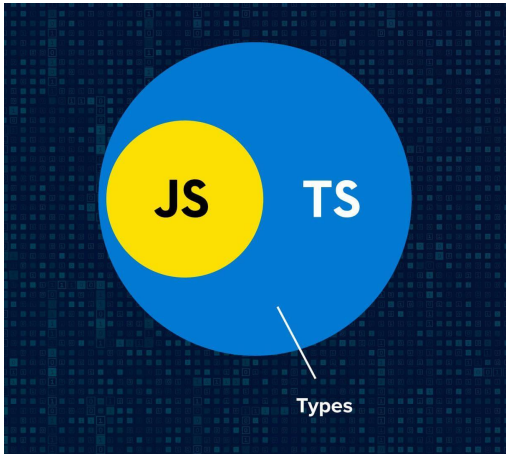
```
async function returnPromises() {
    return  new Promise((resolve) => {
      setTimeout(() => {
        console.log("Promise Executed...");
        resolve("Sample Data");
      }, 3000);
    });
  }

async function ExecuteFunction() {
    var newData = "UniCourt";
    var getPromise = await returnPromises();
    console.log(newData);
    console.log(getPromise);

  }
```

TypeScript

- TypeScript is JavaScript for application-scale development.
- **Strongly typed**, object oriented, compiled language
- JavaScript plus some additional features.



Why use Typescript ?

- Strongly typed
- Code readability
- Interfaces
- Generics

Basic Data Types

- number
  - Represents both integer and fractions
  - ex : **let num : number = 3;**
- string
  - Sequence of character
  - ex: **let str : string = “Hello”**;
- boolean
  - Logical values
  - ex: **let isLoading : boolean = true;**

Creating custom type

Using **type** keyword

Ex :

```
type Person = {
    firstName : string;
    lastName : string;
    id: Number;
}
```

Using Custom Type

```
function printPerson(person : Person){
    console.log(person);
}

const person : Person = {
    firstName:'Arun',
    lastName:'Kenjila',
    id:12
}

printPerson(person);
```

Generics <T>

- Generics is a tool which provides a way to create **reusable** components.
- It creates a component that can work with a **variety of data types** rather than a single data type

Syntax :

Using Generic Type

```
function printMyData<T>(value : T){
    console.log(value);
}

printMyData("this is string");
printMyData(123456);
printMyData(true)
```

Difference between JS & TS

- var and let keyword
- Type-safety
- Custom types
- Generics
- Classes / Interfaces

THANK YOU