JS

Лабораторная работа 5.

1. **TypeScript.** 
   1. Особенности TS в сравнении с JS

* TypeScript поддерживает типы.
* Наличие интерфейсов TypeScript позволяет объявлять пользовательские типы.
* TypeScript соответствует спецификациям ECMAScript6 и 7 + типы, интерфейсы, декораторы, public и private.
* Расширенная поддержка IDE.

Браузеры не понимают язык TS, прежде, чем использовать написанный код – его нужно транспилировать в JS.

* 1. Начало работы
* Установить node.js <https://nodejs.org/> LTS версию.
* Для установки компилятора Typescript необходим пакетный менеджер (npm). Он установлен вместе с системой node.js. В командную строку введите

npm install -g typescript

* Создайте папку для проекта и в ней файл app.ts
* C помощью команды (открыть терминал в VSCode можно при помощи команды ctrl+`)

tsc app.ts

можно транспилировать код в js. Можно указать редакцию ES, которой должен соответствовать js код

tsc --t ES5 main.ts

* 1. Типы данных в TS

При объявлении переменных TS позволяет явно указать тип данных, которому она будет соответствовать

let str: string;

либо устанавливает его, в зависимости от значения, которым была переменная проинициализирована

let str = 'some string';

При попытке записать в эту переменную значение, не соответствующее типу – мы получим ошибку.

let str = '';

str = 123;

В TS есть тип any – позволяющий использовать любое значение в переменной, так же можно объявить переменную указам несколько допустимых типов через |.

let anyValue: any;

let strAndNumber: string | number;

* 1. Обобщения

В TypeScript поддерживаются параметризированные типы, известные также как обобщения, которые могут использоваться в различных сценариях.

Предположим, что требуется вернуть результат функции как объект того типа, который передан в функцию. Тип any использовать не получится, т.к. это не конкретизирует возвращаемое значение. И чтобы конкретизировать возвращаемый тип, мы можем использовать обобщения:

function getId<T>(id: T): T {

    return id;

}

С помощью выражения <T> мы указываем, что функция getId типизирована определенным типом T. При выполнении функции вместо Т будет подставляться конкретный тип.

* 1. Функции в TS

Функции TS похожи на функции JS, но при этом имеется возможность явного объявления типов параметров и возвращаемых значений.

function myFunction(param: number): number {

    return param + 10;

}

Можно указать значения параметров по умолчанию

function myFunction(param: number, param2: string = 'some str'): number {

    return param + 10;

}

Параметры могут быть не обязательными, для этого, после их указания используется знак ?.

function myFunction(param: number, param2: string = 'some str', param3?: boolean): number {

    return param + 10;

}

* 1. Классы в TS

TypeScript предоставляет ключевые слова public, protected и private, для управления доступом к элементам объектов. По умолчанию все элементы класса находятся в открытом доступе (public) и видимы за пределами класса. Если элемент объявлен с модификатором protected, то он видим в классе и его подклассах. Элемент класса, объявленный с модификатором private, видим только в классе.

class Person {

    public firstName: string;

    public lastName: string;

    public age: number;

    private \_ssn: string;

    constructor(firstName: string, lastName: string, age: number, ssn: string) {

        this.firstName = firstName;

        this.lastName = lastName;

        this.age = age;

        this.\_ssn = ssn;

    }

}

let p = new Person("Иван", "Иванов", 45, "123-90-4567");

console.log("Last name: " + p.lastName + " SSN: " + p.\_ssn);

Обратите внимание, что имя закрытой переменной начинается со знака подчеркивания – это соглашение об именах.

* 1. Обобщенные классы

Обобщение может использоваться при создании классов

class User<T> {

    private \_id: T;

    constructor(id: T) {

        this.\_id = id;

    }

    getId(): T {

        return this.\_id;

    }

}

let tom = new User<number>(3);

console.log(tom.getId()); *// возвращает number*

let alice = new User<string>("str");

console.log(alice.getId()); *// возвращает string*

* 1. Интерфейсы

Интерфейс определяет свойства и методы, которые объект должен реализовать. Другими словами, интерфейс - это определение пользовательского типа данных, но без реализации.

interface User {

    id: number;

    name: string;

    someFunction(): string;

}

* 1. Метаданные класса

1. Angular
   1. Начало работы

Для начала работы необходимо установить angular CLI https://cli.angular.io/ (интерфейс командной строки).

Устанавливается он при помощи npm

npm install -g @angular/cli

Если требуется обновить уже установленную CLI

npm uninstall -g angular-cli

npm cache clean

npm install -g @angular/cli@latest