## \*FIRST SLIDE\*

Hi guys. I am glad to see you on my presentation. Today I'll tell you about TypeScript.

So let's start. What is the TypeScript?

Well, it is a static typed JavaScript developed by Microsoft in 2012, which serves to create complex and large applications.

JavaScript is a dynamic typed language, meaning the compiler doesn't know what type of variable you are using until the variable is initialized. This can cause problems and errors in projects. TypeScript supports static typing, which excludes errors associated with incorrect variable typing. At the same time, typing doesn't disappear at all, and it can be used.

#### \*Next slide\*

TypeScript is supported by all IDES.

TypeScript supports new ECMAScript features, so you can develop applications using the latest tools without worrying about browsers support.

#### \*Next slide\*

Code written in TypeScript is backward compatible with JavaScript and compiled into it.

With TypeScript, you can use existing JavaScript code, include popular JavaScript libraries, and call TypeScript code generated from other JavaScript.

At this slide, we can see how TS code compiles to native JS code.

#### \*Next slide\*

You can ask me, we already have JS, why do we need TS?

1. First of all, the code quality improves

2. Code is easier to support

3. At the development stage, a large number of errors are detected

4. TS allows you to create more difficult applications

5. And it’s easier to work in large teams

------------------------------------------------------------------------------------------------------------------------------------

## \*TYPES\*

Since TS is a typed programming language, let's move on to types.

#### \*Next slide\*

TypeScript provides type declarations for static validation of their matching.

The main way to assign a variable type is to write it after the variable name itself, using the colon character, as shown on this slide.

#### \*Next slide\*

If a variable is assigned a single type, you can't redefine it.

The exception to this is the type ANY.

#### \*Next slide\*

Arrays can be created in different ways:

- using brackets,

- or using generics, as in array\_2

In addition, you can create arrays with elements of several different types, as well as tuples: you can set the type of each element in them.

#### \*Next slide\*

In addition, TypeScript allows us to create own types, called aliases.

A custom type is created using the keyword TYPE.

OK, let's go to Interfaces.

------------------------------------------------------------------------------------------------------------------------------------

## \*INTERFACES\*

Interfaces contain properties and methods of custom types, but don't contain their implementation.

The class that implements the interface takes over the implementation. For clarity here is an example:

#### \*Next slide\*

On the slide, we see The readonly property of the interface Rect.

This property is used when you need to change the property only at the beginning, during implementation, and then the property becomes read-only.

If some properties should be optional, we can use the "question mark" operator after the property name, for example like the color property.

#### \*Next slide\*

TS allows interfaces to be inherited. As we can see on the slide, we can add new fields.

In this case, we add the getArea function to the RectWithArea interface, which returns the area of the shape.

OK, let's move on.

------------------------------------------------------------------------------------------------------------------------------------

## \*ENUM\*

In TypeScript, as in other object-oriented languages, there are Enums.

They allow you to define named constants.

There is also the possibility to create text and numeric constants.

Enumerations are defined by the keyword ENUM.

#### \*Next slide\*

The slide shows an example of a numeric enumeration, where each value is mapped to a number.

#### \*Next slide\*

The same enumeration (where the first value is 0, the second is 1, etc) can be achieved with this code. The result is the same.

#### \*Next slide\*

Declaring a text enumeration is identical to a numeric one, however, text values are specified instead of numeric values.

Well, next we will look the functions.

------------------------------------------------------------------------------------------------------------------------------------

## \*FUNCTIONS\*

In TypeScript, we can pass the type of function argument and specify the type of return value.

#### \*Next slide\*

It looks like this.

The example creates two functions, both of which have arguments with specific types.

They also have the type of return value defined.

Functions are called in the same way as in native JavaScript, but the compiler checks the correctness of the parameters passed, and in case of which displays an error.

#### \*Next slide\*

If the function returns nothing, we specify the type VOID.

And there is another type in TypeScript called NEVER.

It’s specified in 2 cases:

- when the function returns an error

- and when it constantly does something and never ends.

OK, let's move on.

------------------------------------------------------------------------------------------------------------------------------------

## \*CLASSES\*

----------------------

#### \*Next slide\*

TypeScript supports ECMAScript 6 classes, which allows you to support the type Declaration option.

Like others programming languages, TypeScript supports access modifiers at the class level.

TypeScript supports three access modifiers - public, private, and protected.

**Public** - By default, members (properties and methods) of TypeScript class are public - so you don’t need to prefix members with the public keyword. Public members are accessible everywhere without restrictions.

**Private** - A private member cannot be accessed outside of its containing class. Private members can be accessed only within the class.

**Protected** - A protected member cannot be accessed outside of its containing class. Protected members can be accessed only within the class and by the instance of its sub/child class.

#### \*Next slide\*

In compiled JavaScript code, there will be no such types of restriction on the members.

------------------------------------------------------------------------------------------------------------------------------------

## \*GENERICS\*

And the last thing I want to tell, it is Generics.

#### \*Next slide\*

Generics allow you to create components that are compatible with a large number of types, not just one.

#### \*Next slide\*

You might ask: why not use the type ANY to take multiple types at once?

When we using the type ANY, we will not be able to find out the original type of the variable being passed.

This code uses the generic parameter T, which type can be captured and used in the future.

------------------------------------------------------------------------------------------------------------------------------------

## \*LAST SLIDE\*

I guess that's all I wanted to tell. I hope it was clear and useful.

Thank you for watching.

See you later.