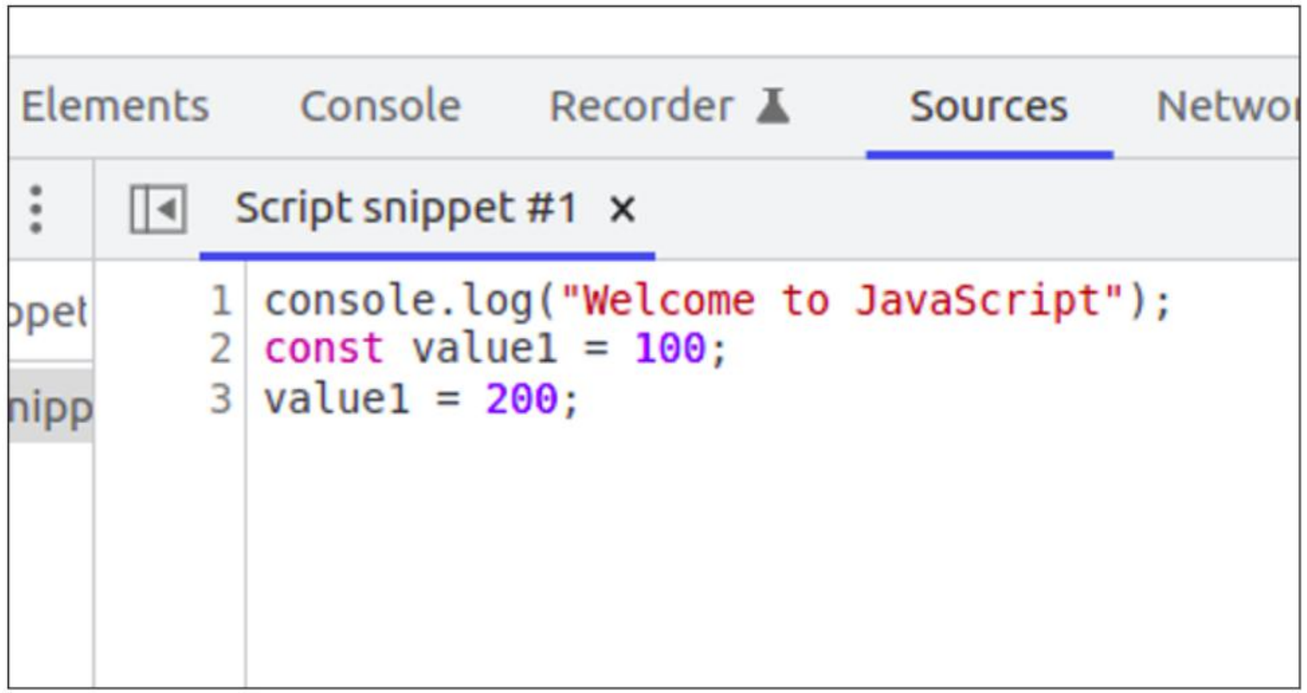
Slide Note

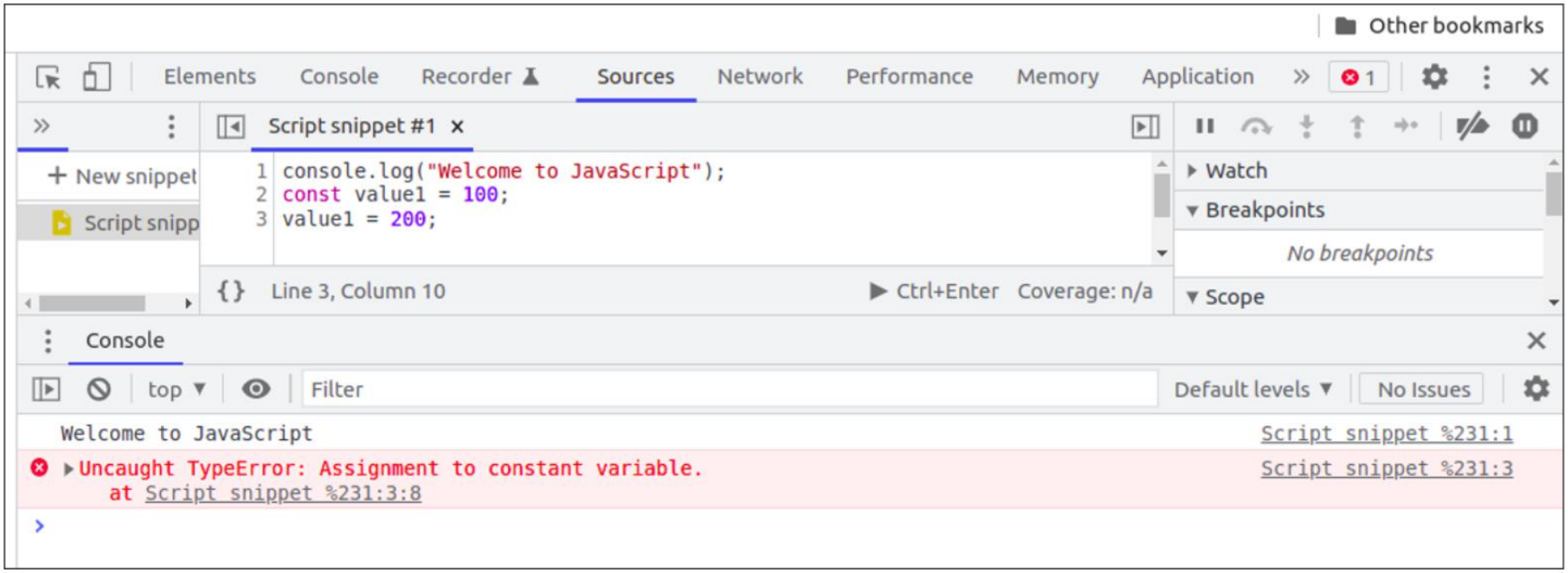
Let’s find out whether the following javascript codes are executable or not.

Slide Note



- Can you execute this JavaScript code successfully?

Slide Note



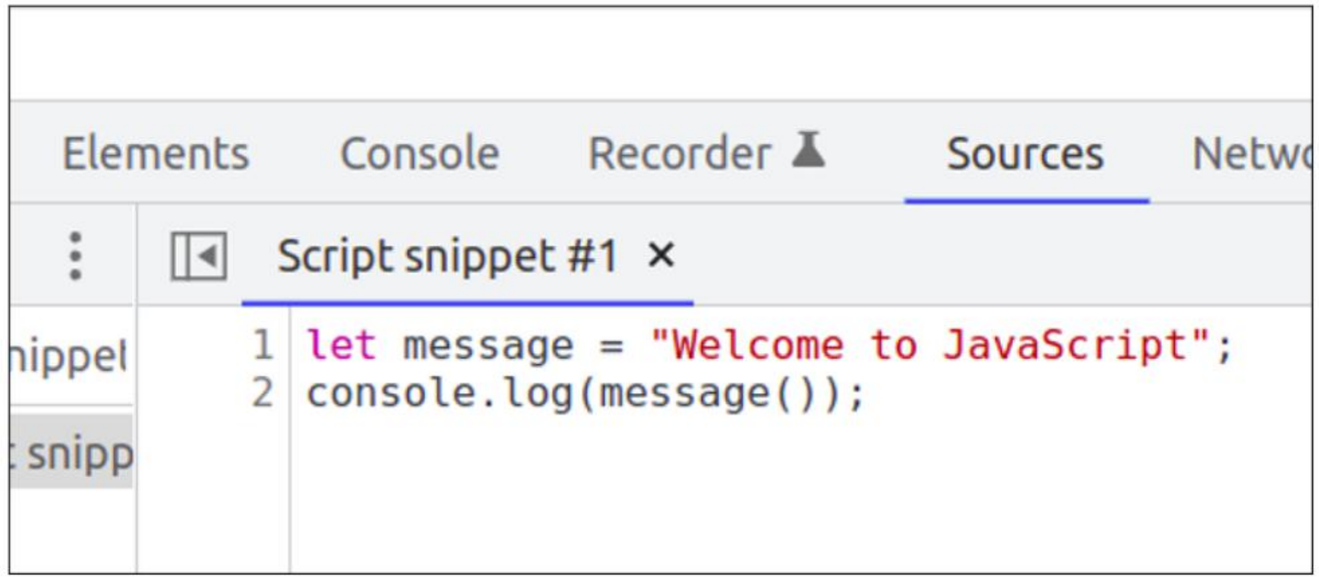
Runtime error stating “TypeError: Assignment to constant variable.”

Menu

00:03:00:00:43

03/ 43

Slide Note

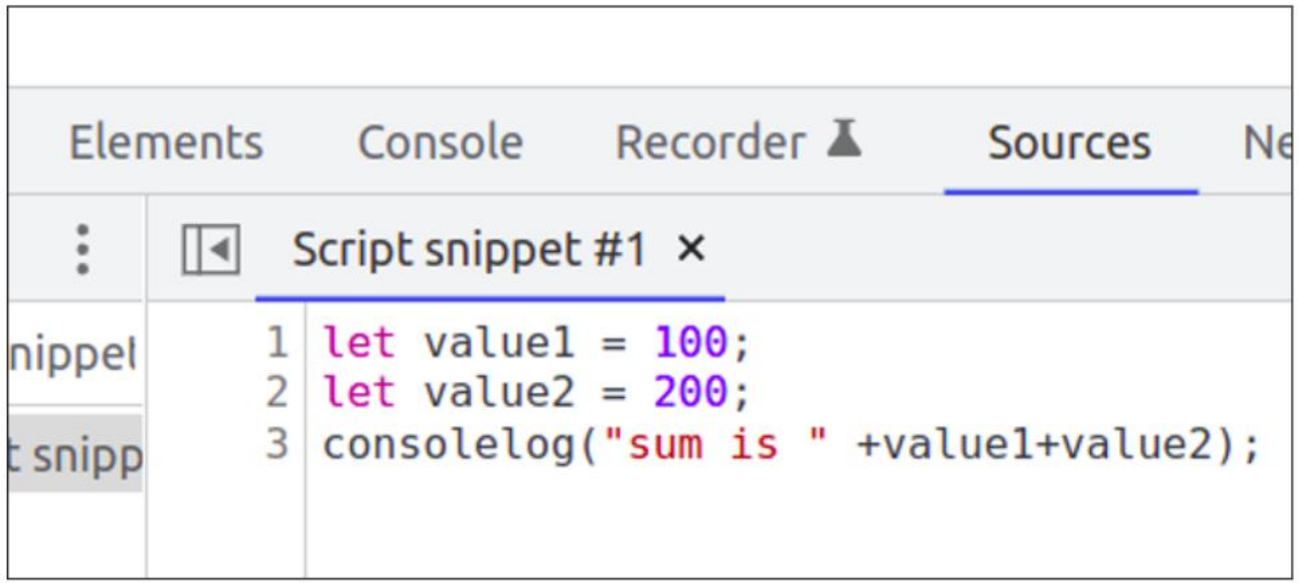


- Do you think this JavaScript code will execute successfully?

Slide Note

Runtime error stating "TypeError: message is not a function"

Slide Note

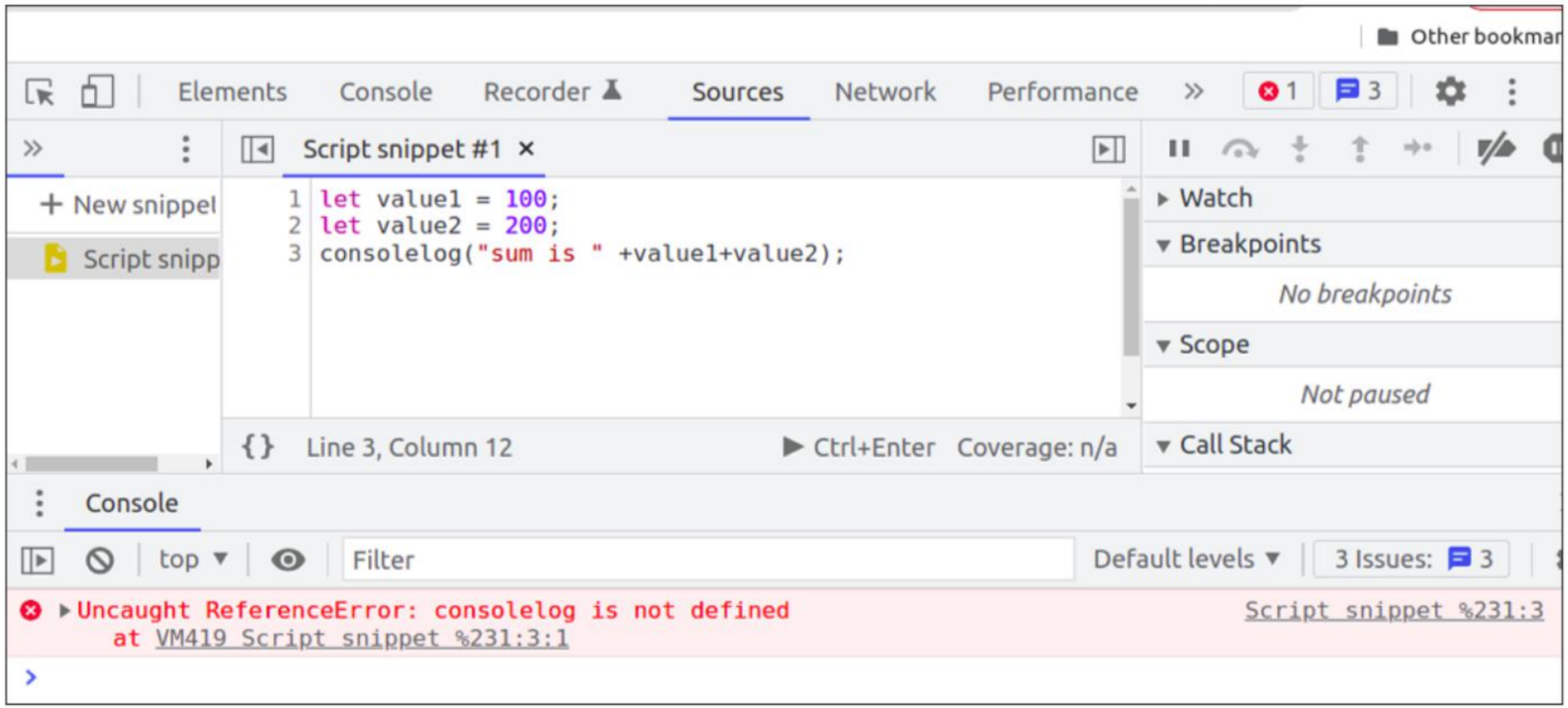


- Will this JavaScript code execute successfully?

Slide Note

For some values, such as the primitives string and number, we can identify their type at runtime using the typeof operator.

But for other things, like functions, there's no corresponding runtime mechanism to identify their types.



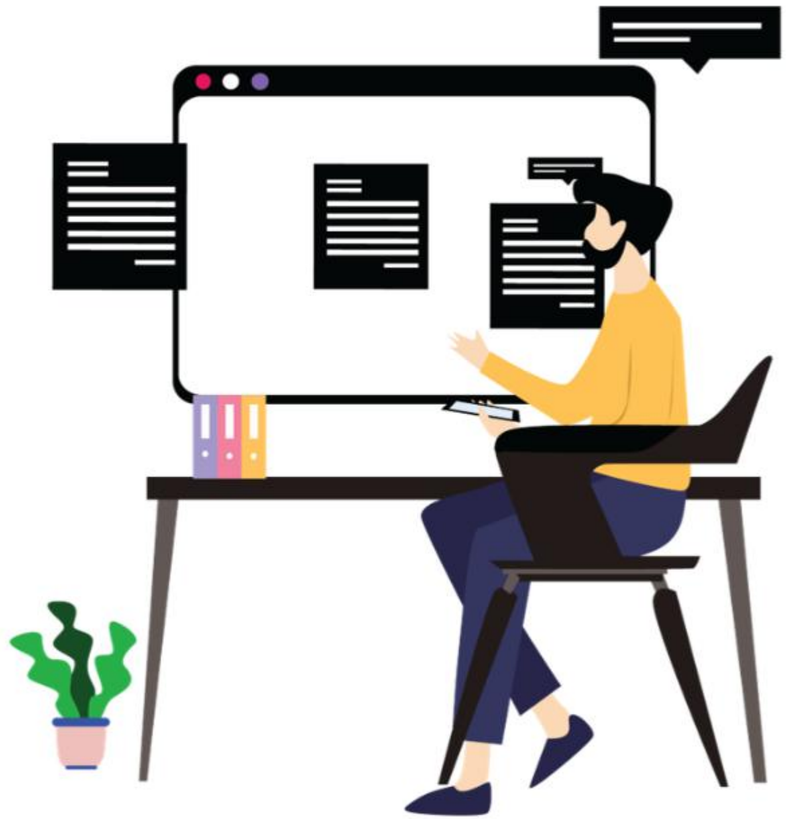
Runtime error stating "ReferenceError: consolelog is not defined"

Slide Note

Wouldn't it be great if these errors were caught early and the code was only executed after the errors were resolved?

Slide Note

Build Interactive Web Pages With TypeScript



Slide Note

Learning Objectives

- Explain the need for TypeScript language
- Identify the different data types available in TypeScript
- Execute functions with type annotations
- Define object types using type aliases
- Create classes to build multiple objects
- Build a simple interactive web page using TypeScript

Slide Note

```
let val1 = 'Hi';
let val2 = 'Hello';
//Will throw runtime error
console.log(`Product: ${val1 * val2}`);
```

```
var message = "Hello World";
console.log(message());
//Runtime-Type error: message is not a
function
```

```
function concatString(str1, str2) {
return str1 + str2;
}
var result = concatString(3, 4);
// Prints 7
console.log(`Result is ${result}`);
```

Need For TypeScript

- The first JS code will throw a runtime error since incorrect operations are performed with the data.
- The second JS code makes a function call with string data and hence breaks at runtime.
- The third JS code has accepted incorrect arguments. Therefore, the expected output was not obtained.
- JavaScript is a dynamically-typed scripting language that executes in the browser without undergoing any prior compilation process.
- How can we overcome such runtime errors and unexpected outcomes?
- Will defining the data type while declaring the variables solve these issues so that errors are available at compile time?
- Are there any languages that accept types when declaring variables to reduce the amount of work required by developers?

TypeScript

Slide Note

What Is TypeScript?

- TypeScript is a strongly-typed programming language developed as a superset of JavaScript.
- TypeScript allows the programmer to define types to increase the reliability of a program by avoiding type mismatches.
- Datatypes are assigned by simply placing a colon after the variable name but before the equal sign:

  Let {variable name}: {variable type} = {variable value}
- The.ts extension is used to save TypeScript files.

```
//declare a variable with data type
let personName: string = "John";
console.log("Hello " + personName);

//declare 2 variables with
numeric type
let num1: number = 10;
let num2: number = 20;
console.log(`Product of 2
numbers: ${num1 * num2}`);

//declare a variable with boolean
type
let isCitizen: boolean = true;
let age: number= 23;
if(isCitizen && age > 18){
console.log("Eligible for voting");
}
```

Slide Note

How TypeScript Code Executes?

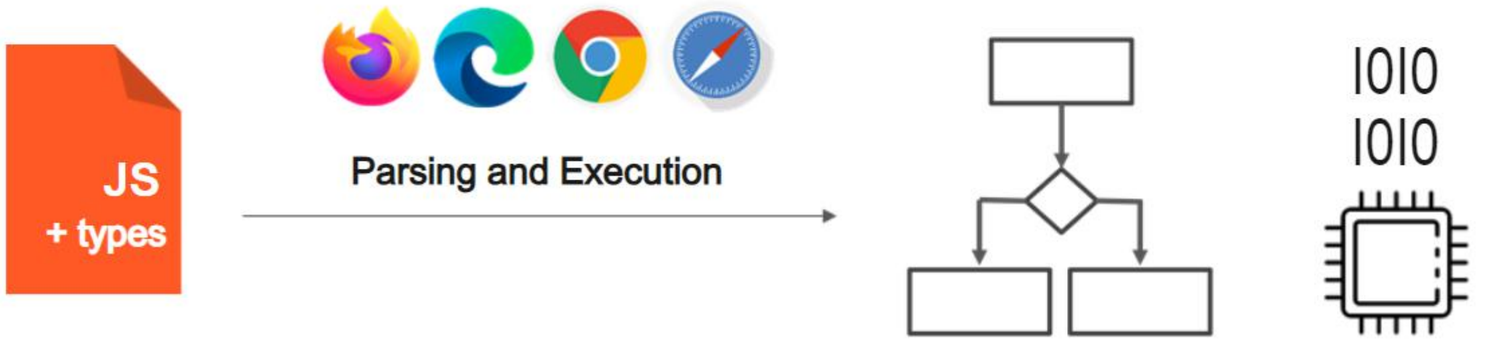
- All web browsers have an inbuilt JavaScript engine to optimize and execute the JS code.
- Then, how do you execute a web application that was created using TypeScript code?
- TypeScript code is code that is converted into its equivalent JavaScript code for execution in the browser.
- TypeScript compilation tool (tsc) is used to convert into JavaScript code.

Slide Note

TypeScript code is transpiled(compiled) to JavaScript Code.

Transpilation is the process of interpreting certain programming languages and translating it to a specific target language.

TypeScript Compilation



Slide Note

For running typescript code, there are two ways:

- After compiling typescript file to JavaScript, add .js file in index.html
- Run the JavaScript file in the terminal by using the command **node .js**

TypeScript Code Execution

Steps to Execute TypeScript Code

1. Install TypeScript as an NPM package on your local machine or in your project by using:
   npm install –g typescript

2. Check the installed version of typescript by using:
   tsc –v

3. Compile the file by using:
   tsc <filename>.ts

4. Include the <filename>.js file generated in index.html to see the output in the browser's developer console

Note: Steps 1 and 2 are done once for the TypeScript installation. Steps 3 and Step 4 should be followed for running any TypeScript file.

Click this link to know the differences between compiler and interpreter.

```
function add(num1, num2) {
return num1 + num2;
}
//declare variables with number datatype
let a: number = 10;
let b: number = 20;
//Executes and returns the sum
console.log("Sum " + add(a, b));
//declare variable with string data type
let c: string = "John";
//Gives Compilation error as argument
type is not matching
console.log("Sum "+ add(a,c));
```

Slide Note

Data Types in TypeScript

- A data type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.
- It defines how the values of that datatypes are stored in memory and what operations can be done on the data.
- JavaScript does not allow variables/constants to be declared with types. As a result, type checking is missing.
- TypeScript allows variable declarations along with data types to prevent a data mismatch.
- TypeScript provides built-in types to work with the simplest units of data and user defined datatypes to create your own types.

Built-in Datatypes

- Number
- Boolean
- String
- Null
- Undefined
- Any

User Defined Datatypes

- Enumerations (enums)
- Classes
- Interfaces
- Arrays
- Tuples

Slide Note

TypeScript Datatypes

Create a "Hello, World!" program using TypeScript.

Create a TypeScript program that stores employee details in different variables without associating with data types initially. Then add the appropriate data types to each variable. Observe that the TypeScript compiler notifies the error if the wrong types are used in the wrong places.

Click here for the demo solution.

DEMO

Slide Note

Basic Data Types in TypeScript

Data Type	Description	Example
string	Textual data	let str: string = "hello";
number	An integer or a floating-point number	let value: number = 100;
boolean	Any of the two values: true or false	let hasText: boolean = true;
any	Data type used when type checking errors need to be ignored	let value: any = "hello"; let value: any = 1234;
array	A sequential storage of data accessed by index	let marks: number[]= {55,89};
union	Type formed from two or more other types	let code: (string | number);

Slide Note

TypeScript Datatype: any

- TypeScript provides a special datatype called any that disables type checking.
- It effectively allows all types to be used.
- When a value is of type any, you can:
  - Access one or more properties of it
  - Call it a function.
  - Assign it to a value of any type.
- Use any type to store a value of a type you don't know at compile time or when you migrate a JavaScript project over to a TypeScript project.
- This is also useful when you are getting input from users or at a third-party library/service.
- Don't use any as a type unnecessarily since it disables type checking and hides the compilation errors.

Example without any

```
let value = false;
// Error: Type 'string' is not assignable to
type 'number'.
value = "hello";
// Error: Argument of type 'boolean' is not
assignable to parameter of type 'number'.
let roundValue = Math.round(value);
```

Example with any

```
let value: any = { x: 0 };
// Using `any` disables all further type
checking, and it is assumed  you know the
environment better than TypeScript.
value.foo();
value();
value.bar = 100;
value = "hello";
const n: number = value;
```

Slide Note

Quick Check

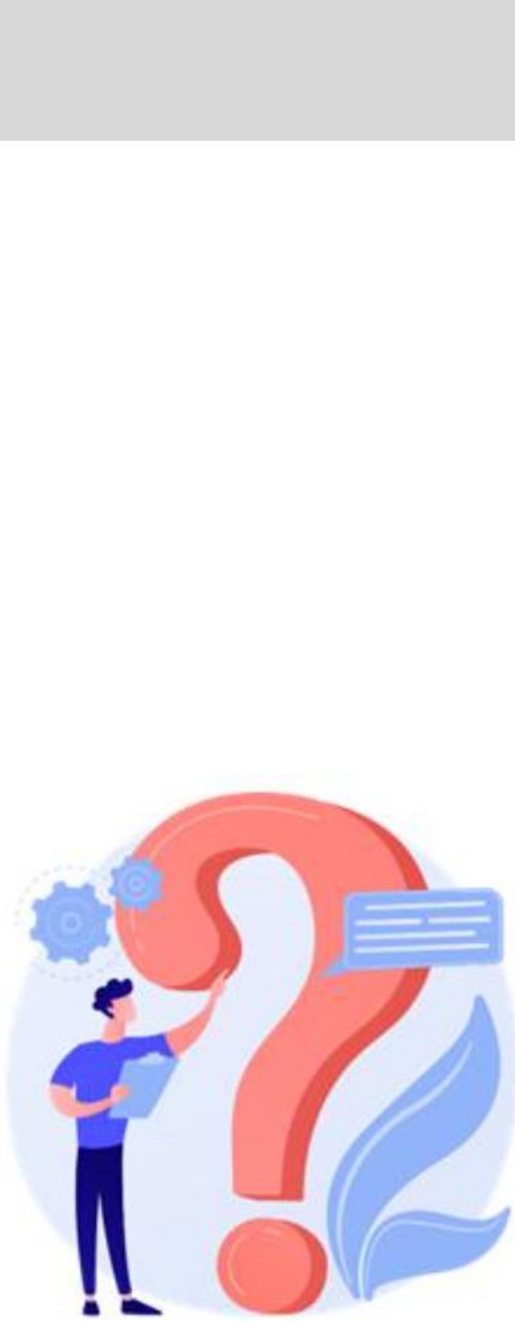
Which of the following declares a variable in TypeScript?

```
1. let value = 123;

2. let value: number = 123

3. let number: value = 123
```

4. All the above

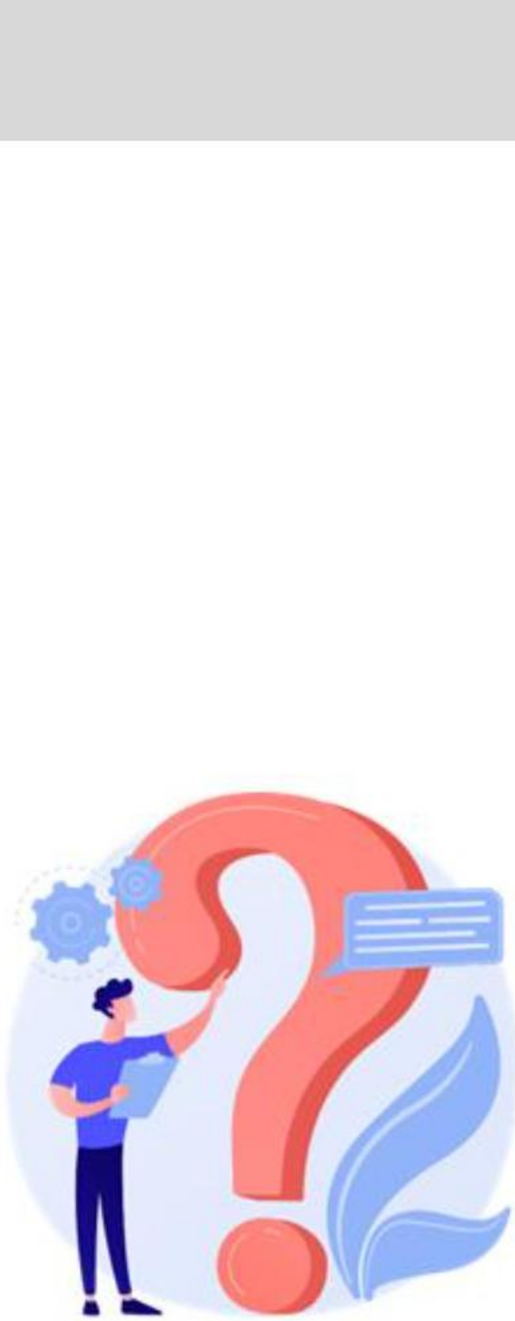


Slide Note

Quick Check: Solution

Which of the following declares a variable in TypeScript?

1. `let value = 123;`
2. `let value: number = 123`
3. `let number: value = 123`
4. All the above



Slide Note

• **Parameter type annotations**: Allows the ability to add type annotations after each parameter to declare the type of parameter that the function accepts

• **Return type annotations**: Specifies the type that the function returns and appears after the parameter list

Functions

- Functions
  - Functions are the fundamental building blocks to perform specific tasks, which make the code more readable, maintainable, and reusable. TypeScript allows us to specify the data types for parameters and return values through:
    - Parameter type annotations
    - Return type annotations
- Anonymous Functions
  - Functions without names are anonymous functions. These are inline functions, which are used only once and do not require a name. Example: Callback functions.

Slide Note

Functions With Type Annotations

Declare a function to calculate the square of a number that accepts only numeric data type.

Call the function with the numeric data type and calls it again with the string data type to find the difference.
Determine which one runs successfully and which one throws an error.

Click here for the demo solution.

DEMO

Menu

00:23:00:00:43

23/ 43

Slide Note

Functions With Type Annotations

- TypeScript functions allow us to add type annotations after each parameter to declare the type of parameter that the function accepts.
- The function getSquare() takes one parameter called "value," of type number.
- The type of the value returned is specified as a number in the function declaration.
- Calling the function with the proper data type executes it successfully; otherwise, it gives a compile time error.
- The return type can be specified as void if the function is not returning any value.

Parameter with type

Return Type

```
function getSquare(value: number): number{
return value * value;
}
//executes properly
let result: number = getSquare(10);

//would give compile time error
getSquare("Hello World");
```

Function with parameter type and return type annotations

Slide Note

Contextual Typing With Anonymous Functions

Convert each string value in an array to its uppercase using TypeScript anonymous and arrow functions.

Click here for the demo solution.

DEMO

Slide Note

In this code, the forEach function is called for the "names" array that is of "string" type. So, TypeScript used this data type(string) for the callback function of the forEach function. As a result, the type "s" will be of "string" type.

```
//No type annotation, but typescript can
understand the type
const names = ["Alice", "Bob", "Eve"];

//Contextual typing for Anonymous functions
names.forEach(function (s) {
console.log(s.toUpperCase());
});

//Contextual typing for Arrow Functions
names.forEach((s) => {
console.log(s.toUpperCase());
});
```

Anonymous Functions

- Anonymous functions behave a little differently from function declarations.
- In the example provided, no type annotation is given for the parameter "s."
- In this code, the forEach function is called for the "names" array, which is of "string" type. So, TypeScript used this data type (string) for the callback function of the forEach function.
- As a result, the data type for "s" will be of the "string" type.
- The context that the function occurred within informs what type it should have, which is then called as contextual typing."
- Contextual typing is also applicable for arrow functions.

Slide Note

Object Types

In JavaScript, the fundamental way that we group and pass around data is through objects. In TypeScript, objects are represented through object types.

- Object types are defined by listing their properties and their types for each property.
- Specifying the type part of each property is optional. If it is not specified, it will be assumed to be of any data type.
- Object types can also specify that some or all their properties are optional. For this, add a ? after the property name.
- Object Types can be:
  - Anonymous
  - Named using type aliases.

```
type Student = {
    studentId: number | string,
    firstName: string,
    LastName?: string,
    course: string,
    fees: number,
    availHostel: boolean
};
function printStudent(student) {
    return "Student ID:" + student.studentId + "\n"
+
    "First Name:" + student.firstName + "\n" +
    "Last Name:" + student.lastName + "\n" +
    "Course:" + student.course + "\n" +
    "Fees:" + student.fees + "\n" +
    "Avail Hostel Facility:" + student.availHostel;
}
let hostelStudent: Student = {
    studentId: 101,
    firstName: "Allen",
    LastName: "Keth",
    course: "Java",
    fees: 200,
    availHostel: true
};
console.log(printStudent(hostelStudent));
```

Slide Note

Object Types in TypeScript

Print the details of a university student by creating a student object type. The details to store and print are:

- Student ID
- First name
- Last name
- Fees
- Course
- Whether the student has opted for a hostel facility or not.

Click here for the demo solution.

DEMO

Slide Note

- Built-in datatypes, which stores a single data, do not suffice for complex business requirements.
- To store similar kind of data as a collection of values, **arrays** can be used.
- But how do we store composite values?
- To store some composite values, which contains a diverse set of data as well as manipulate the data, **objects** are used.
- To create multiple objects with similar attributes and behaviors, you should create different objects with different values.
- Is there a way to create a template for an object, which can then be reused to create multiple instances with different values?

Need for User Defined Datatypes

- Single Data => Primitives

Name **James** Age **18**

- Collection of data => Arrays

Marks

56	78	65	98	69	81	77

- Composite Data => Objects

Student

studentId	101
studentName	James
studentAge	18
studentMarks	98

- To create multiple objects with similar attributes and behaviors, you should create different objects with different values. This can be done using object types.

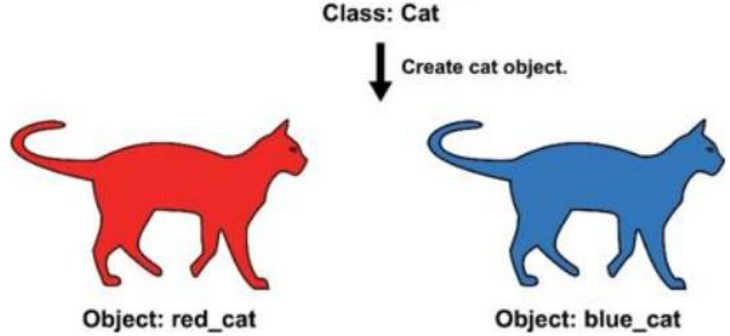
- Would it be possible to create a template for an object where multiple instances with different values could be created?

Class

Slide Note

TypeScript Class

- A class is basically a blueprint to create specific types of objects whose states and behaviours are represented by the variables and methods defined inside the class.
- A class definition can contain the following properties:
  - Field: Is a variable declared in a class.
  - Method: Represents an action for the object.
  - Constructor: Is responsible for initializing the object in the memory.



- As seen in the image, a cat is a class which has attributes like color, weight, and breed.
- They share behaviours such as eating, making sounds, walking, and sleeping.
- Each instance of a cat will have a specific color and weight and belong to a specific breed .

Slide Note

- The keyword class is used to declare a class.
- The class can be referenced using the **class** name.
- The state of objects belonging to the class type are represented using variables declared inside the class.
- The constructors are used to create objects of the class type using the keyword **constructor** with specified state passed as arguments to them.
- In the code, this.studentID, this.name and this.marks refers to class variables. The keyword **this** refers to the current object or class instance variable.
- Methods are declared and defined inside the class, which help define the behavior of objects of the class type.
- TypeScript **class** binds data and code that operates on the data, which is known as **encapsulation.**

TypeScript Class Definition

```
class Student{
    studentID: number;
    name: string;
    marks: number;
    constructor(studentID: number,
name: string, marks: number){
        this.studentID = studentID;
        this.name = name;
        this.marks = marks;
    }
    getMarks(){
        return this.marks;
    }
    changeName(name: string){
        this.name=name;
    }
}
```

Class Variables to define the state

Constructor to initialize class variables

`this` refers to the current object or current instance variable

Class Methods to define the behavior

Slide Note

Class Objects

- Objects are instances of a class.
- A class by itself does not occupy any space in a program stack memory during execution, but objects do.
- In the code, student is the name of the object of type Student.
- The keyword new is used to create a new object in the Student class.
- Constructor arguments are optional and, if provided, should match one of the constructors defined in the class body.

```
const student= new Student(11, "Rose Catherine", 87);

//Access the student's name property
student.name = "Roselin Catherine";

//call getMarks instance method
console.log(student.getMarks());
```

Slide Note

```
//Multiple instances created for the student class
const student1 = new Student(11, "Rose Catherine", 87);
const student2 = new Student(12, "John Britto", 75);
const student3 = new Student(13, "Emy Jackson", 93);

//Change the student's name property
student1.name = "Roselin Catherine";

//Prints Student Name: Roselin Catherine, Marks: 87
console.log(`Student Name: ${student1.name},
Marks: ${student1.getMarks()}`);

//Prints Student Name: John Britto, Marks: 75
console.log(`Student Name: ${student2.name}, Marks:
${student.getMarks()}`);

//Prints Student Name: Emy Jackson, Marks: 93
console.log(`Student Name: ${student3.name},
Marks: ${student3.getMarks()}`);
```

Multiple Class Objects

- Multiple objects can be constructed from a single class by setting up different initial values for each of the objects.
- The same constructor is invoked with different initial values when each object is created.
- All instances of the class share the implementation of the method declared inside the class.

Slide Note

Create a Student Class

Create a student class that has various properties like studentID, name, and marks. Define the constructor to initialize objects and getMarks() method to return the student's marks. Create multiple instances of the student class and access its properties.

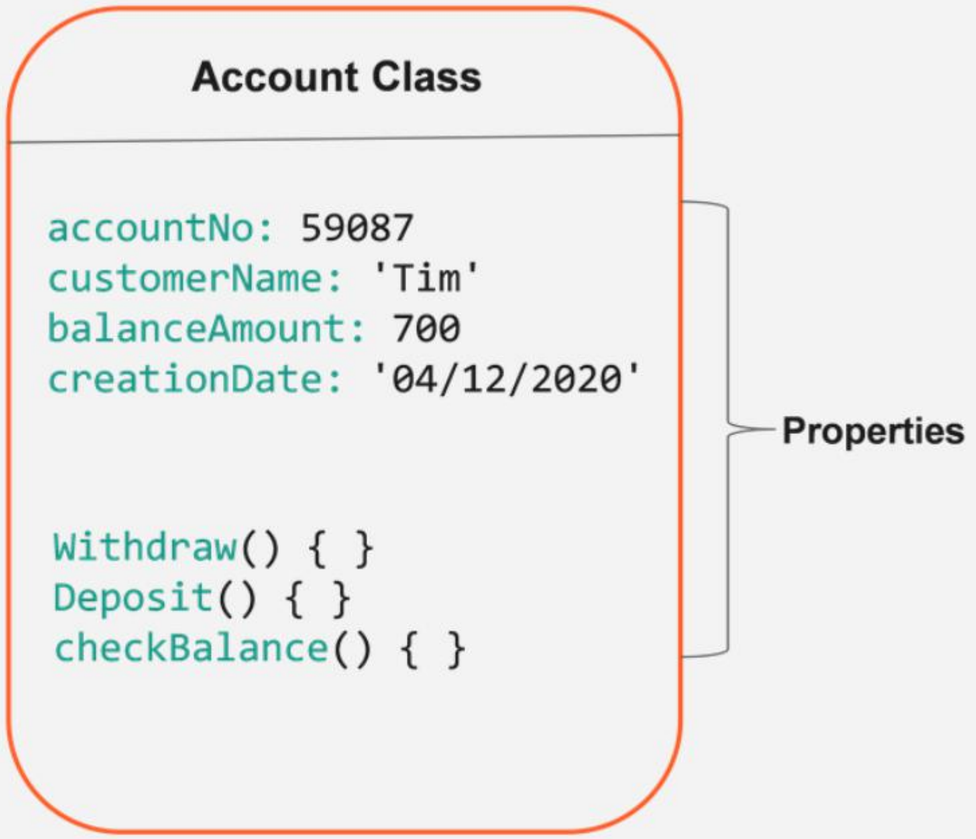
Click here for the demo solution.

DEMO

Slide Note

Model Savings Account

- How to model a bank savings account as an object?
- The image lists the properties of a savings account.
- The `balanceAmount` can only be accessed using the `checkBalance()` method since accessing it directly is not the secured option.
- When a customer withdraws or deposits money, the `balanceAmount` is updated using the `withdraw()` and `deposit()` methods.
- If the withdrawal amount is greater than the `balanceAmount`, then the transaction fails.
- The `balanceAmount` is not updated directly but only through the class property methods.



Slide Note

Data Access

- Imagine that the bank has an overdraft amount of $50 for all customers who have had their account for 3 years.
- In such a case, another new Boolean property called `isOverDraftAllowed` should be added to the class.
- When the customer withdraws money, the customer is checked for eligibility using the `isOverDraftAllowed` value.
- Then, the withdraw() method updates the `balanceAmount` if it does not exceed the overdraft amount. Otherwise, the transaction fails without updating the `balanceAmount`.
- Authenticated users can be validated first inside the withdraw() method before accessing the `balanceAmount`.. As a result, accessing it within the withdraw() method is more secure than accessing it directly outside the class.
- How do you protect these values from being accessed from outside the class?
- You need to restrict their accessibility by controlling the visibility of the balance of data outside the class.

Slide Note

TypeScript Data Modifiers

In object-oriented programming, class members can be made public or private, i.e., a class can control the visibility of its data members. This is done using access modifiers.

There are three types of access modifiers in TypeScript:

- Public: By default, all members of a class in TypeScript are public, and these members can be accessed anywhere without any restrictions.
- Private: The private access modifier ensures that class members are visible only to that class and are not accessible outside the containing class.
- Protected: The protected access modifier is similar to the private access modifier, except that the protected members can be accessed using their deriving classes.

Slide Note

Class Properties With Access Specifiers

- The visibility of data members should be restricted. It should be available to modify outside the class body only if needed.
- In this code, declaring studentID and studentName as private restricts their access outside the class.
- The studentID and studentName are initialized with values inside the constructor.
- The studentID is accessed outside the class only through its getter method.

```
class Student {
   private _studentID: number;
   private _studentName: string;
   constructor (studentID: number,
studentName: string) {
      this._studentID = studentID;
      this._studentName = studentName;
   }
   //getter method
   get studentID(): number {
      return this._studentID;
   }
}
let ob = new Student(111,"John");

//compilation error-property studentID is
private and accessible only within the class
ob._studentID = 123;

//properties accessed through getter methods
console.log("Student id:" + ob.studentID);
```

Slide Note

In this demo, when the user submits the contact form after filling the valid data, the form field values are captured using JavaScript FormData object. The **FormData** interface provides a way to easily construct a set of key/value pairs representing form fields and their values.

Later, The **Object.fromEntries()** method is used to transform the list of key-value pairs into an object. Using this object properties, the Contact class object is constructed and the new contact object is added to the database using fetch API.

A confirmation message is displayed to the user with his name and contact mail address. The Contact class properties are not directly accessed outside the class. Instead, they are accessed only through their respective getter methods.

Fetch API is used to store the contact information into the database. Similar to Axios API, the Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses

Contact Form

Create a contact form page by modelling the contact class with username, email, and message properties. The properties are declared private to restrict their accessibility.

The contact details are stored in a database using the fetch API. A confirmation message is displayed to the user after a successful submission.

Click here for the demo solution.

Learn more about Fetch API.

DEMO

Slide Note

tsconfig.json

```
{
"compilerOptions":
  {
   "target": "es2019"/* Specify
ECMAScript target version */,
   "outDir": "public/js" /*Redirect
output structure to the directory*/,
   "rootDir": "src" /* Specify the root
directory of input files. */,
  "strict": true /* Enable all strict
type-checking options. */,
  },
  "include": ["src"]
}
```

Typescript Configuration File

- For a Typescript project with multiple files, a configuration file called tsconfig.json is added, guiding the compiler to generate JavaScript files.
- tsconfig.json specifies the root files and the compiler options required to compile the project.
- The presence of a tsconfig.json file in a directory indicates that the directory is the root of a TypeScript project.
- Compiler Options: These options specify the TypeScript's configuration and cover how the language should work.
- Include: Specifies an array of file names or patterns to include in the program. These file names are resolved relative to the directory containing the tsconfig.json file.

Slide Note

Quick Check

Which of the below is a valid Union type variable?

```
1. let emp: [number, string] = [1,"James"]

2. let emp: (number | string) = 555

3. let emp: string[ ] = ["Sam"];

4. let emp: (number | string) = (1,"James")
```

Slide Note

Quick Check: Solution

Which of the below is a valid Union type variable?

1. `let emp: [number, string] = [1,"James"]`
2. **`let emp: (number | string) = 555`**
3. `let emp: string[ ] = ["Sam"];`
4. `let emp: (number | string) = (1,"James")`