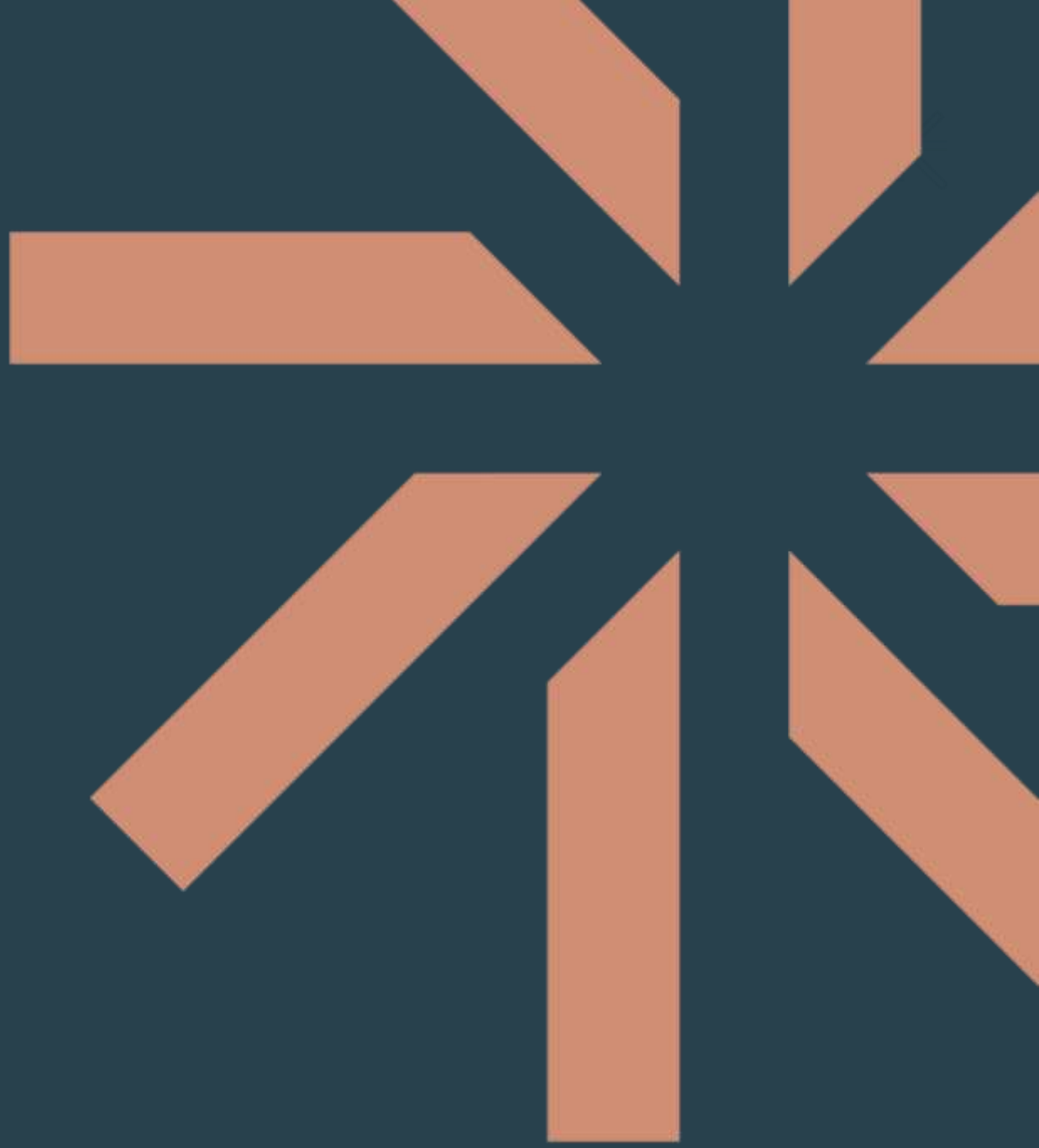
TYPESCRIPT

Qinshift
Academy

TypeScript

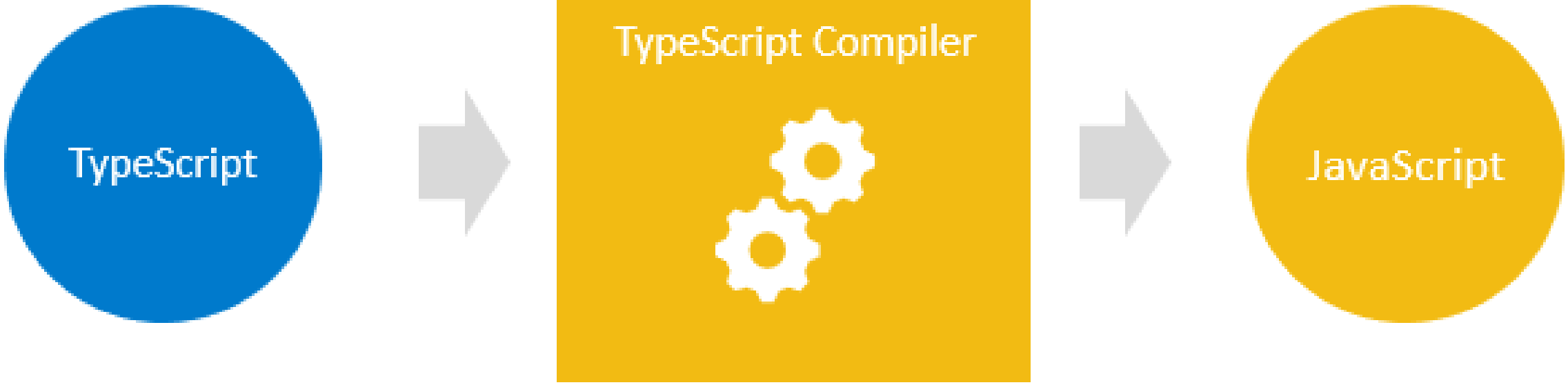
- TypeScript is a typed superset of JavaScript that compiles to plain JavaScript
- Maintained by Microsoft
- Widely adopted for backend and frontend development



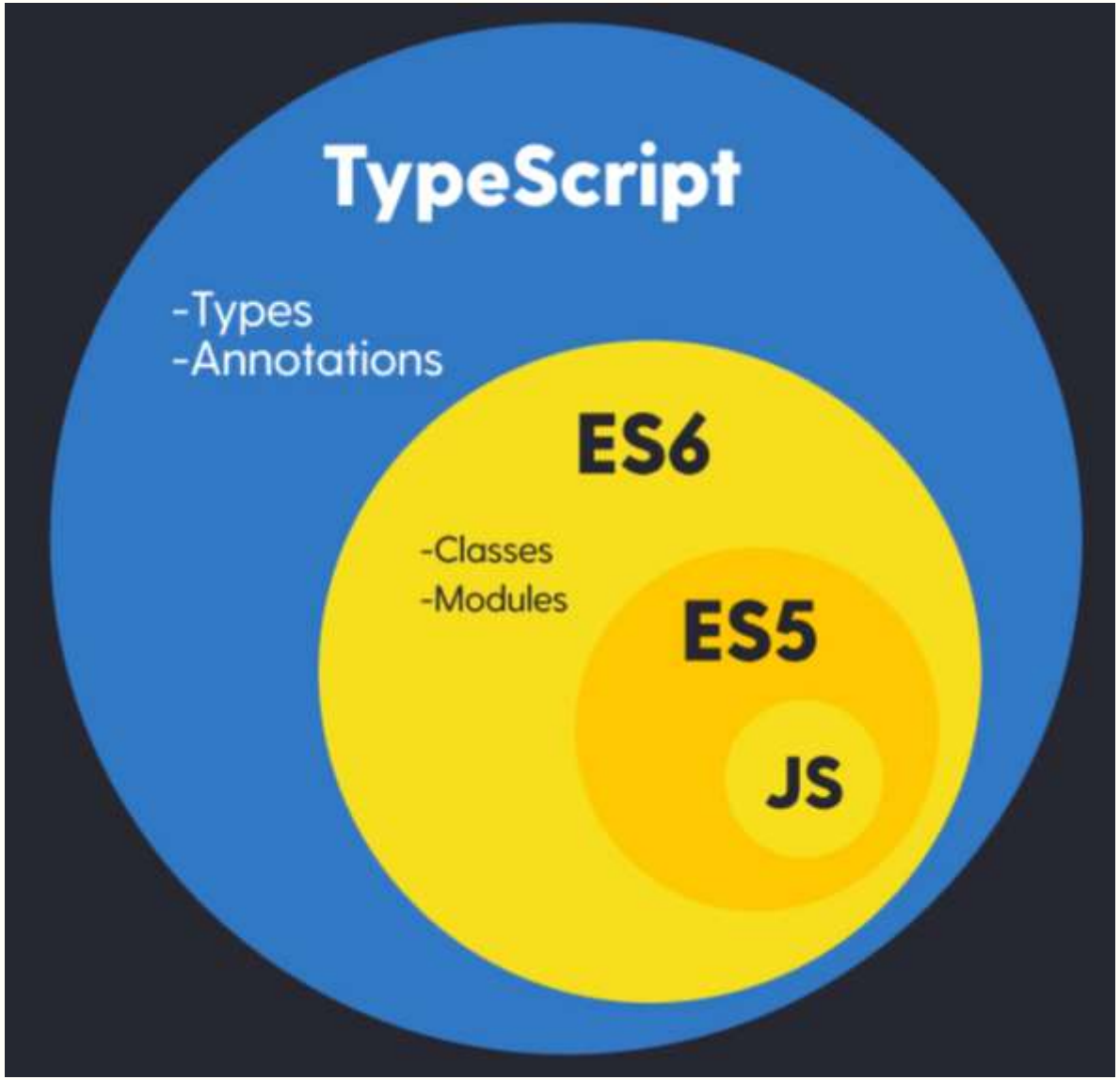
WHAT IS TYPESCRIPT?

- TypeScript is a super set of JavaScript
- TypeScript builds on top of JavaScript. First, you write the TypeScript code. Then, you compile the TypeScript code into plain JavaScript code using a TypeScript compiler.
- Once you have the plain JavaScript code, you can deploy it to any environments that JavaScript runs.
- TypeScript files use the .ts extension rather than the .js extension of JavaScript files.

Compiling TypeScript



TypeScript adds types and annotations to JavaScript



Why TypeScript?

- TypeScript improves your productivity while helping avoid bugs

Types increase productivity by helping you avoid many mistakes. By using types, you can catch bugs at the compile-time instead of having them occurring at runtime.

- TypeScript brings the future JavaScript to today

TypeScript supports the upcoming features planned in the ES Next for the current JavaScript engines. It means that you can use the new JavaScript features before web browsers (or other environments) fully support them.

TypeScript vs JavaScript

JavaScript	TypeScript
Dynamic typing	Static typing
Loose error detection	Early error detection
Less tooling support	Rich tooling support (IntelliSense, refactoring)
Scalable, but harder to maintain	Scalable, easier to maintain

TYPES IN TYPESCRIPT

TypeScript inherits the built-in types from JavaScript. TypeScript types is categorized into:

- Primitive types
- Object types

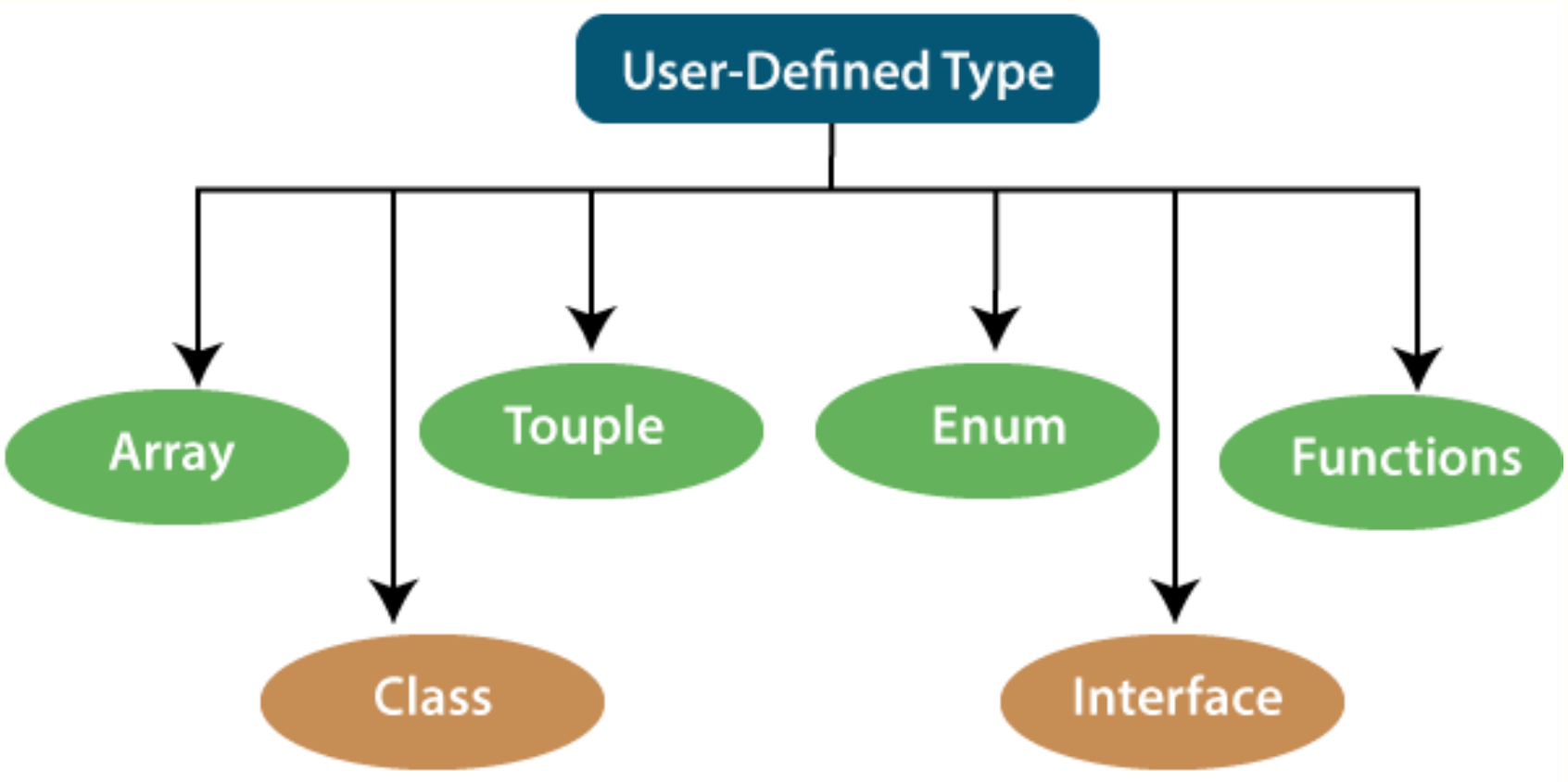
Primitive types

The following illustrates the primitive types in TypeScript

Name	Description
`string`	represents text data
`number`	represents numeric values
`boolean`	has true and false values
`null`	has one value: null
`undefined`	has one value: `undefined`. It is a default value of an uninitialized variable
`symbol`	represents a unique constant value

Object types

The following illustrates the object types in TypeScript



FUNCTIONS

TypeScript functions are the building blocks of readable, maintainable, and reusable code.

Like JavaScript, you use the function keyword to declare a function in TypeScript:

```
function name(parameter: type, parameter:type,...): returnType {
    // do something
}
```

Unlike JavaScript, TypeScript allows you to use type annotations in parameters and return value of a function.

CLASSES IN TYPESCRIPT?

TypeScript class adds type annotations to the properties and methods of the class. The following shows the Person class in TypeScript

```
class Person {
    ssn: string;
    firstName: string;
    lastName: string;

    constructor(ssn: string, firstName: string, lastName: string) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}
```

CLASSES IN TYPESCRIPT?

When you annotate types to properties, constructor, and method, TypeScript compiler will carry the corresponding type checks.

For example, you cannot initialize the ssn with a number. The following code will result in an error:

```
let person = new Person(171280926, 'John', 'Doe');
```

ACCESS MODIFIERS

Access modifiers change the visibility of the properties and methods of a class. TypeScript provides three access modifiers:

- The **private** modifier allows access within the same class.
- The **protected** modifier allows access within the same class and subclasses.
- The **public** modifier allows access from any location.

Note that TypeScript controls the access logically during compilation time, not at runtime.

GETTERS & SETTERS

The getters and setters allow you to control the access to the properties of a class.

For each property:

- A **getter** method returns the value of the property's value. A getter is also called an accessor.
- A **setter** method updates the property's value. A setter is also known as a mutator.

GETTERS & SETTERS

A getter method starts with the keyword get and a setter method starts with the keyword set.

```
class Person {
    private _age: number;
    private _firstName: string;
    private _lastName: string;


    public get age() {
        return this._age;
    }

    public set age(theAge: number) {
        if (theAge <= 0 || theAge >= 200) {
            throw new Error('The age is invalid');
        }
        this._age = theAge;
    }
```

TYPESCRIPT INHERITANCE

To inherit a class, you use the **extends** keyword. For example the following Employee class inherits the Person class:

```
class Employee extends Person {
    //..
}
```

In this example, the Employee is a child class and the Person is the parent class.

TYPESCRIPT ABSTRACT CLASSES

An abstract class is typically used to define common behaviors for derived classes to extend. Unlike a regular class, an abstract class **cannot be instantiated directly**.

To declare an abstract class, you use the abstract keyword:

```
abstract class Employee {

    //...

}
```

TYPESCRIPT ABSTRACT CLASSES

Typically, an abstract class contains one or more abstract methods.

An abstract method does not contain implementation. It only defines the signature of the method without including the method body. An abstract method must be implemented in the derived class.

The following shows the Employee abstract class that has the getSalary() abstract method:

```
abstract class Employee {
    constructor(private firstName: string, private lastName: string) {
    }
    abstract getSalary(): number
```

GENERICS

TypeScript generics allow you to write the reusable and generalized form of functions, classes, and interfaces.

The following shows a generic function that returns the random element from an array of type T:

```
function getRandomElement<T>(items: T[]): T {
    let randomIndex = Math.floor(Math.random() * items.length);
    return items[randomIndex];
}
```

GENERICS

This function uses type variable T. The T allows you to capture the type that is provided at the time of calling the function. Also, the function uses the T type variable as its return type.

This getRandomElement() function is generic because it can work with any data type including string, number, objects,…

By convention, we use the letter T as the type variable. However, you can freely use other letters such as A, B C, …

Questions?

Trainer Name

Trainer

trainer@mail.com

Assistant Name

Assistant

asistant@mail.com

Qinshift Academy