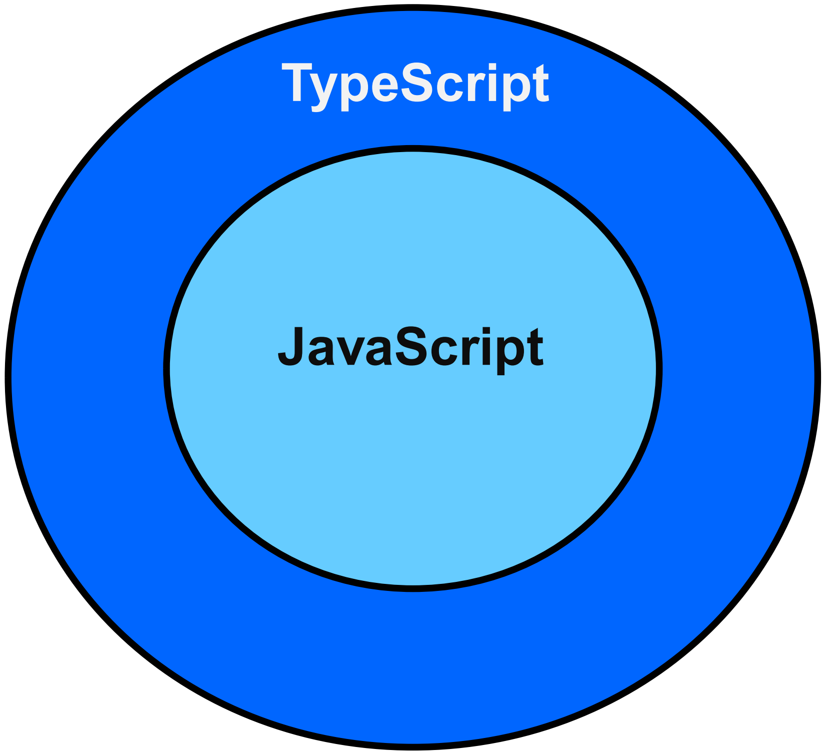
TypeScript

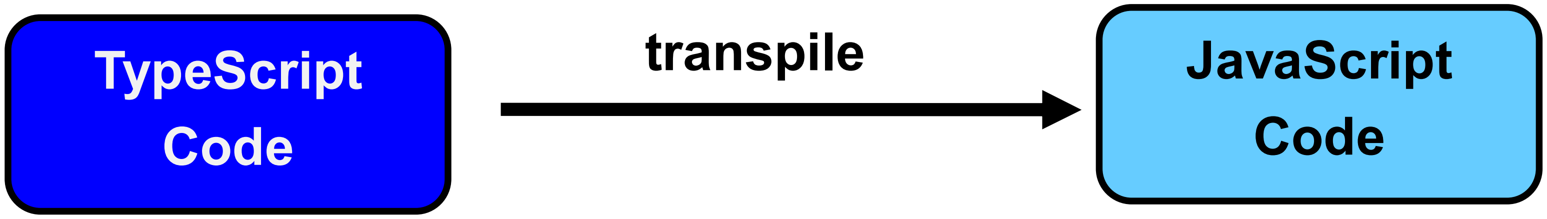
What is TypeScript

- TypeScript is superset of javascript



TypeScript features

- Strong typing
- Object oriented features (classes , interfaces , constructors etc)
- Compile-time errors
- Tool availability - code editors intelisense
- Typescript should be transpiled to javascript before sending to browsers (browsers do not support typescript)



TypeScript setup

Install typescript:

npm install –g typescript

Check version:

tsc –version

Create sample file (using microsoft visual code editor):

code main.ts

Transpile :

tsc main.ts

Run the javascript file:

node main.js

variable declaration - let

- let is same as var in global scope
- var changes the variable of global scope
- let allows to declare local variables

```
function varTest(){
    var a =30;
      if(true){
              var a = 50;
              console.log(a);
      }
      console.log(a);
}
```

```
50
50
```

output

```
function letTest(){
    let a =30;
      if(true){
              let a = 50;
              console.log(a);
      }
      console.log(a);
}
```

```
50
30
```

Data types

- Variables do have type

  let count = 5; // type is number

  count = ‘a’; // error

- Explicit declaration:
- other types include boolean, string , any, array

  examples :

  let a:number;

  let b:boolean;

  let e:number[] = [3,4,5];

  let f: any[] = [3,’abc’, true, 45];

enums

- enums used to declare predefined values

```
enum Color { Red = 0, Blue = 3, Green = 4};
enum Color {Red, Blue, Green }   // values 0, 1, 2 etc
let bgColor = Color.Red;
```

arrow functions

- Function can be defined using fat arrow
- Example :

```
let doLog = (message) => console.log(message);
let doLog = message=> console.log(message);
let doLog =  () => console.log(“hello”);
let adder = (x:number,y:number) => { return x+y;  };
let adder =(x:number, y:number)  => x+y;
```

interfaces

- Used to group data members
- Can be used as single entity for function class etc
- Example:

```
 interface Point {
        x:number,
        y:number,
 }

let draw = (p : Point)=>{   //….   };

 let  pt:Point = {  x:30, y:40 };
 draw(pt);
```

Classes

- Combines data, methods and constructors
- Similar to Java classes
- Example:

```
class Point {
   x:number;
   y:number;
   draw() {
     console.log(“x : “+this.x+” , y : “+this.y);
   }
   getDistance((other: Point) {
      // …..
   }
}
```

Using Classes

- Variables of class type can be declared
- Initialized with constructors

- Example:

```
 let pt : Point = new point();
  pt.draw();    //  x & y undefined

let pt = new point();      //  type is implicitly Point
 pt.x=50;     // initializing data
 pt.y=25;
 pt.draw();
```

Constructors

- keyword constructor can be used to include constructors in class

- Example:

```
class Point {
     x:number;
     y:number;
    constructor(x:number, y:number){
           this.x = x;
           this.y = y;
    }
}

let pt = new point(30, 20);
```

Constructors

- Multiple constructors not allowed
- To have different options, optional parameters can be used
- Once a parameter is made optional all those on right of that should be optional (use ? for optional parameter)

- Example:

```
class Point {
     x:number;
     y:number;
    constructor(x?:number, y?:number){
            this.x = x;
            this.y = y;
    }
}

let pt = new Point( );    pt.x=10;  pt.y =5;
let pt = new Point(20);
let pt = new Point(20,10);
```

Access Modifiers

- private used to hide members
- keyword public can also be used. But default is public

- Example:

```
class Point {
    private  x:number;
    private y:number;
    constructor(x:number, y:number){
        this.x = x;
        this.y = y;
    }
}

let pt = new Point(20,10);
pt.x =20;   // not allowed
```

Fields as constructor arguments

- Fields can be specified in constructor argument with keyword private or public
- No separate declaration required
- Constructor code also not required
- Example:

```
class Point {
        constructor( private x:number,  private y:number){
        }
}

let pt = new Point(20,10);
pt.x =20;   // not allowed
```

Properties

- Properties can be defined with getters and setters
- Example:

```
class Point {
        constructor( private x:number,  private y:number){
        }
   get X()  {  return this.x;  }
   set X(val)   {  this.x = val; }
}

let pt = new Point(20,10);
pt.X  = 10 ;  // directly refer the setter method
let var =  pt.X;   // calls getter
```

Abstract Classes

- Define an abstract class in Typescript using the abstract keyword
- Abstract classes are mainly for inheritance where other classes may derive from them
- We cannot create an instance of an abstract class.
- An abstract class typically includes one or more abstract methods or property declarations
- The class which extends the abstract class must define all the abstract methods.

Abstract Classes

```
abstract class Person {
    name: string;
    constructor(name: string) {
        this.name = name;
    }

    display(): void{
        console.log(this.name);
    }

    abstract find(string): Person;
}
```

```
class Employee extends Person {
    empCode: number;

    constructor(name: string, code: number) {
        super(name); // must call super()
        this.empCode = code;
    }

    find(name:string): Person {
        // get employee data from a db
        return new Employee(name, 1);
    }
}
```

```
let emp: Person = new Employee("James", 100);
emp.display(); //James

let emp2: Person = emp.find('Steve');
```

Modules

- separate files can be created as modules
- export statement is used to make the components in other modules
- Other modules can use import statement
- Example:

point.ts

```
export class Point {
        constructor(   …){
        }
    get X()  {  return this.x;  }
    set X(val)   {  this.x = val; }
 }
```

main.ts

```
import {Point}  from  ‘./point’;
 let pt = new Point(20,10);
 pt.X  = 10 ;
```

Default Export

- Only one default export per file is allowed

```
export default class Person {   }
```

- import looks like this (without braces)

```
import Person from “./modules";
```

- We can give any name while importing

```
import User  from "./modules";
```

- Named Export: should be imported using braces

```
export class Person {   }
export class Employee {  }
```

- Multiple imports are allowed

```
import {Person, Employee} from “./modules”;
```

const

- Used to declare constants

```
const colors=[ ];
colors.push(‘red’);
colors.push(‘blue’):

console.log(colors);

colors =  345;        // error
```