Microsoft
Learn

STUDENT AMBASSADOR

Microsoft Learn
Student Ambassadors

Typing With TypeScript

Shreya Khandelwal
Nirali Sahoo

About the Speakers:

Shreya Khandelwal

- Born and brought up in Rajasthan, India
- Beta MLSA
- **https://www.linkedin.com/in/shreyakhandelwal99/**

Nirali Sahoo

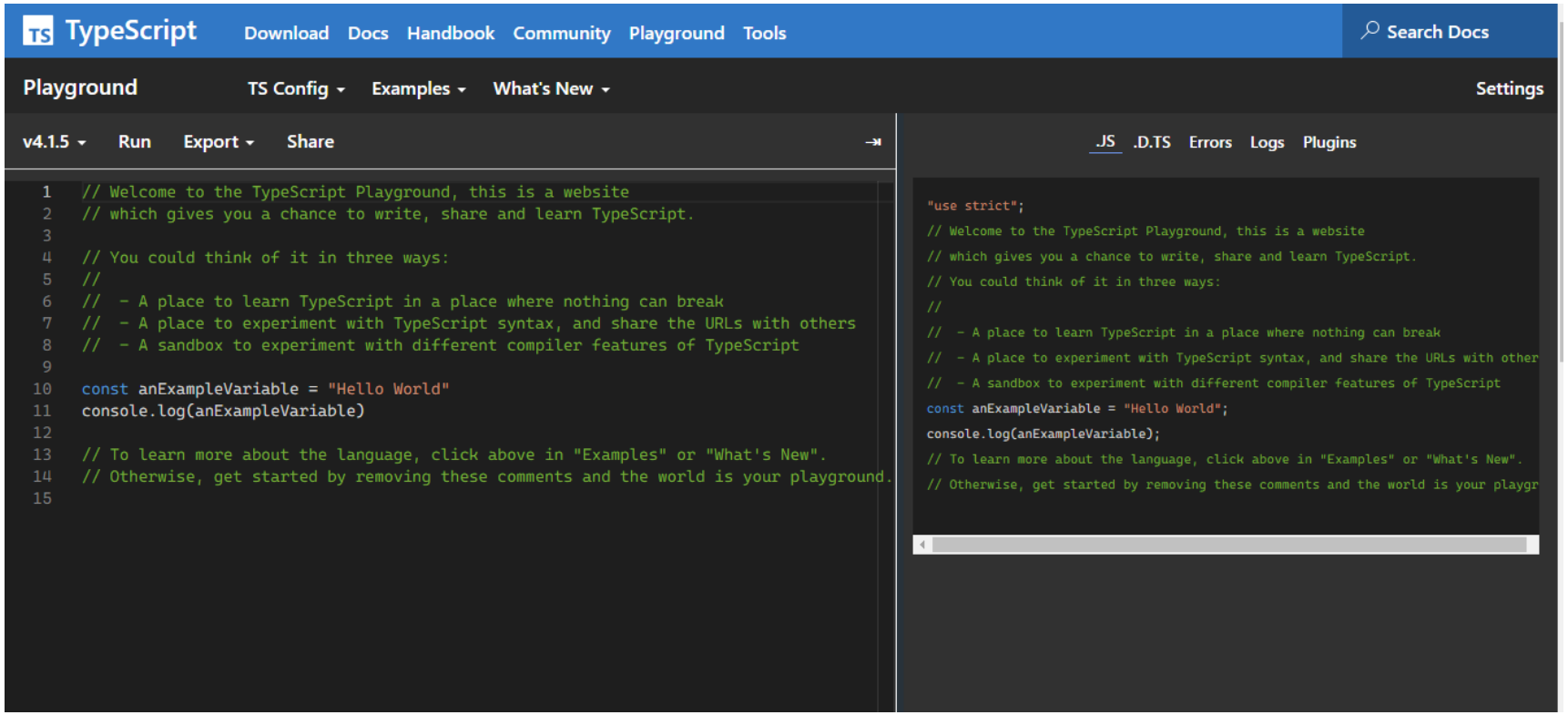
- Born and brought up in Odisha, India
- CSE Sophomore in IIIT Bhubaneswar
- Beta MLSA
- https://www.linkedin.com/in/niralisahoo/

Episode 1 Summary

- **TypeScript**:
  - open-source
  - OOPL
  - developed and maintained by **Microsoft**
  - Superset of the JavaScript language
  - ES6 version of JavaScript.
  - 3 components: Language, TSC, Language services
- **Installation:**
  - Requires npm (node package manager)
  - `npm install -g typescript`
  - Compiling : `tsc <file_path>`
  - Watch mode: `tsc –w <file_path>`
  - For extra compiler options: `tsconfig.json file`

Online Compiler

https://www.typescriptlang.org/play



Episode 2 Summary

Variables

var

i. Global scope

ii. Variable can be redeclared

```
TS app.ts > ...
1  for (var i = 0; i <= 3; i++)
2  {
3      var num = i;
4  }
5
6  console.log(num);
```

Inspector | Console

Filter Output

3

let

i. Limited to block scope

ii. Variable cannot be redeclared

```
TS app.ts > num
1  for (var i = 0; i <= 3; i++)
2  {
3      let num
4  }
5
6  console.log(num);
7
```

any

Cannot find name 'num'. ts(2304)

Peek Problem (Alt+F8) No quick fixes available

Variables

const

i. Once declared, its value cannot be changed

```
TS app.ts > ...
1   const num = 5;
2
3           var num: number
4
5   for (var   Cannot redeclare block-scoped variable 'num'. ts(2451)
6   {          Peek Problem (Alt+F8)   No quick fixes available
7       var num = i;
8   }
9
10  console.log(num);
```

static

i. Associated with a class and not with the object

ii. Value can be accessed only when called on a class

```
TS app.ts > ...
1   class class_name {
2       static variable_1 = 1;
3       variable_1 = 2;
4   }
5
6   let object_1 = new class_name();
7
8   console.log(class_name.variable_1);
9   console.log(object_1.variable_1);
```

```
Inspector   Console
Filter Output
1
2
```

TypeScript Operators

- Arithmetic operators
- Relational (comparison) operators
- Logical operators
- Bitwise operators
- Assignment operators
- Ternary/Conditional operators
- Concatenation Operator
- Type operator

Type Operators

In - used to check for the existence of a property on an object.

```
let Bike = {make: 'Honda', model: 'CLIQ',
year: 2018};
console.log('make' in Bike);   //
```

Output:

```
true
```

Delete - It is used to delete the properties from the objects.

```
let Bike = { Company1: 'Honda',
             Company2: 'Hero'};
delete Bike.Company1;
console.log(Bike);   //
```

Output:

```
{ Company2: 'Hero'}
```

Typeof - It returns the data type of the operand.

```
let message = "Welcome to " + "Event";
console.log(typeof message);  //
```

Output:

```
String
```

Instanceof - It is used to check if the object is of a specified type or not.

```
let arr = [1, 2, 3];
console.log( arr instanceof Array ); //
true
console.log( arr instanceof String ); //
false
```

TypeScript Type Annotation

Type Annotations are annotations which can be placed anywhere when we use a type. It helps the compiler in checking the types of variable and avoid errors when dealing with the data types.
We can specify the type by using a **colon(: Type)** after a variable name, parameter, or property.

Syntax:

```
var variableName: TypeAnnotation = value;
```

```
var age: number = 44;          // number variable
var name: string = "Rahul";    // string variable
var isUpdated: boolean = true; // Boolean variable
```

TypeScript Arrays

```
let array_name[:datatype] = [val1,val2,valn..]
```

There are two types of an array:

- Single-Dimensional Array - let array_name[:datatype];
- Multi-Dimensional Array - let arr_name:datatype[][] = [[a1,a2,a3], [b1,b2,b3]];

array methods

- concat()
- Pop()
- Push()
- indexOf()
- reverse()

TypeScript Unions

Two or more data types are combined using the pipe symbol (|) to denote a Union Type.

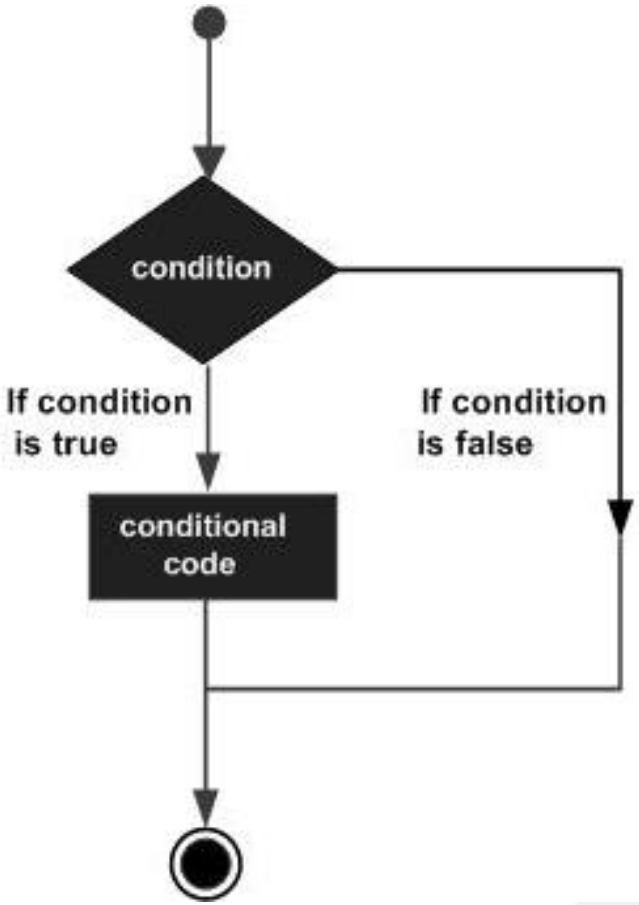
```
Type1 | Type2 | Type3
```

Episode 3 : Decision Making, Loops & Functions

Microsoft Learn
STUDENT AMBASSADOR

Decision Making

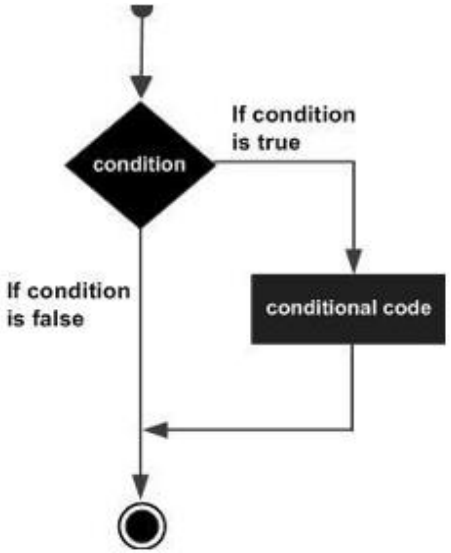
- if statement
- if...else statement
- else…if and nested if statements
- switch statement



If Statement

Syntax:

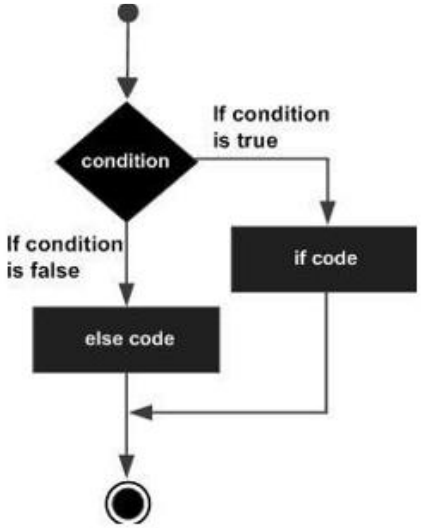
```
if(boolean_expression) {
   // statement(s) will execute
if the boolean expression is
true
}
```



if...else Statement

Syntax:

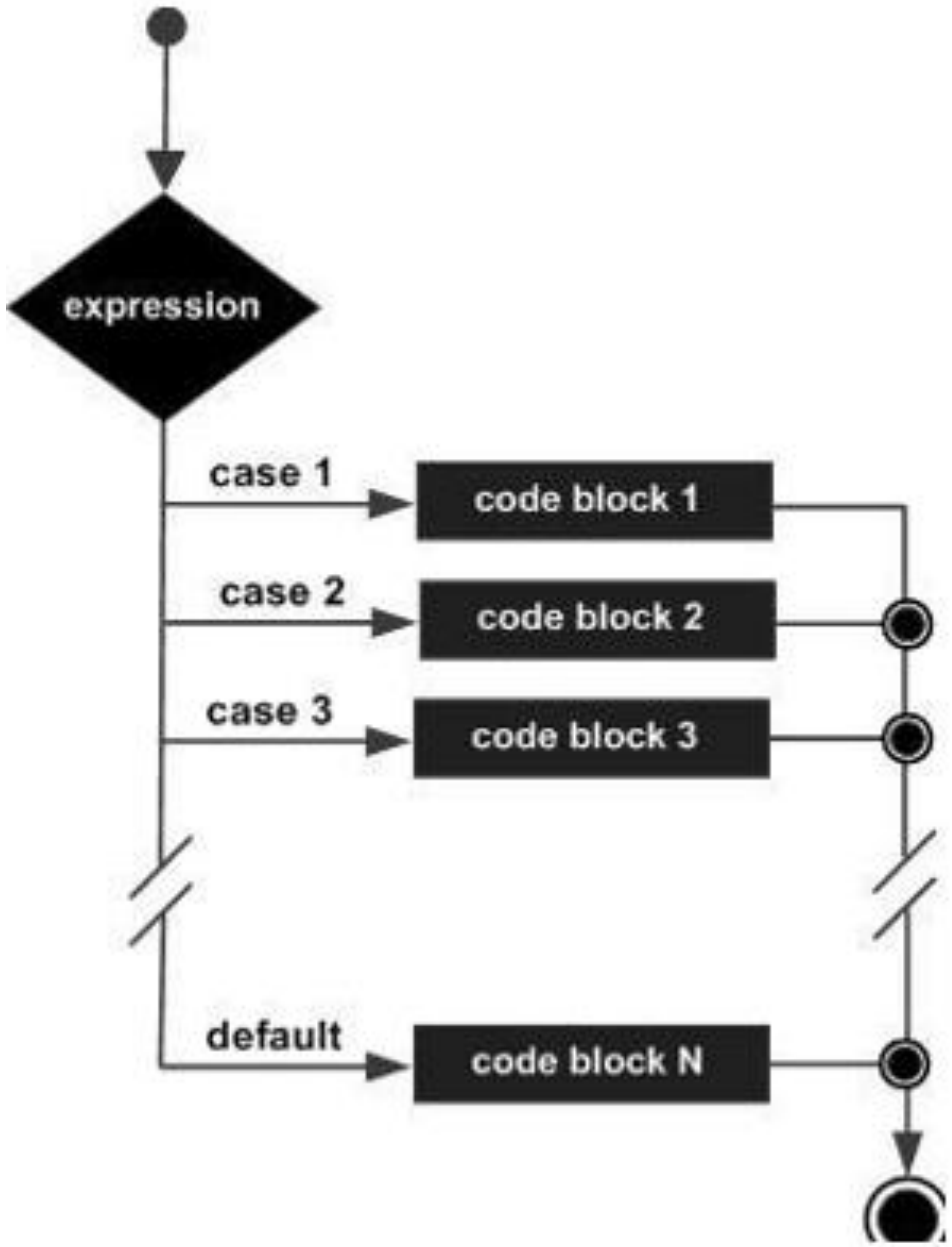
```
if(boolean_expression) {
   // statement(s) will execute if the boolean
expression is true
} else {
   // statement(s) will execute if the boolean
expression is false
}
```



Switch…case Statement

Syntax:

```
switch(variable_expression) {
   case constant_expr1: {
      //statements;
      break;
   }
   case constant_expr2: {
      //statements;
      break;
   }
   default: {
      //statements;
      break;
   }
}
```



Example

```
var grade:string = "A";
switch(grade) {
   case "A": {
      console.log("Excellent");
      break;
   }
   case "B": {
      console.log("Good");
      break;
   }
   case "C": {
      console.log("Fair");
      break;
   }
   case "D": {
      console.log("Poor");
      break;
   }
   default: {
      console.log("Invalid choice");
      break;
   }
}
```

Output

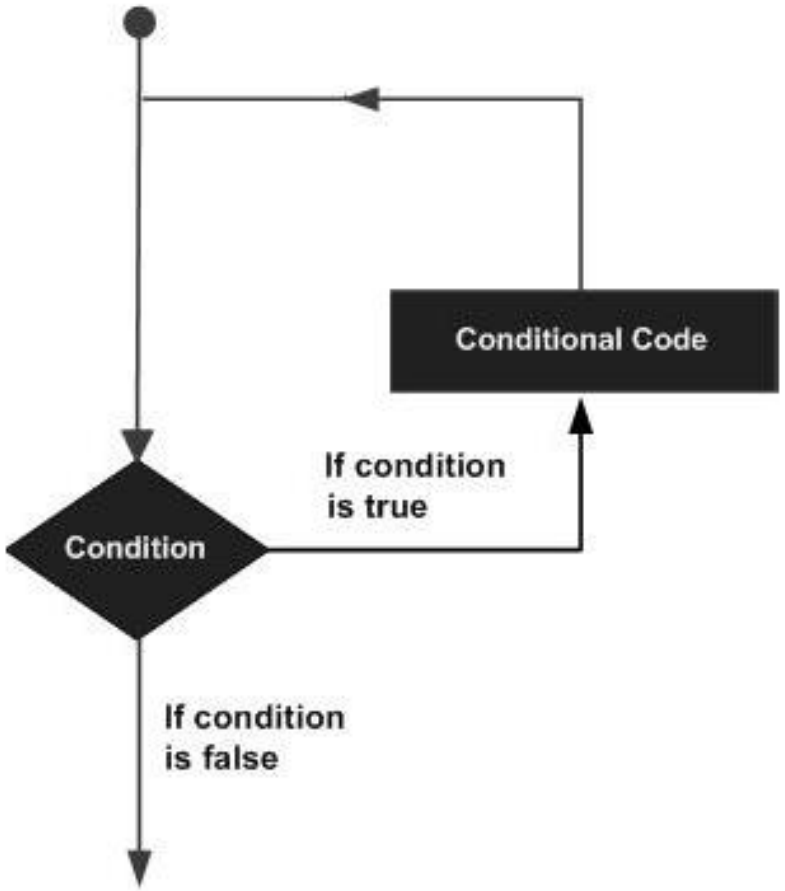
```
Excellent
```

Rules that apply to a switch statement −

- There can be any number of case statements within a switch.
- The case statements can include only constants. It cannot be a variable or an expression.
- The data type of the variable_expression and the constant expression must match.
- Unless you put a break after each block of code, execution flows into the next block.
- The case expression must be unique.
- The default block is optional.

Loops

- Definite loop:
  - for loop
- Indefinite loop:
  - while loop
  - do...while loop

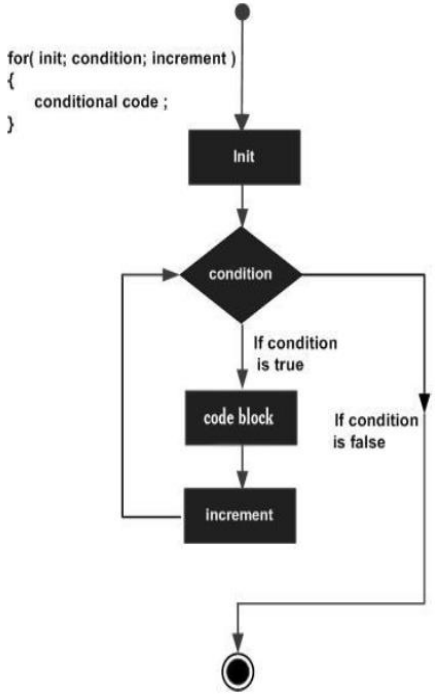


Definite Loop

for loop

Syntax:

```
for (init_count_val; trmntn-conditn; step) {
   //statements
}
```



for...in loop

Syntax:

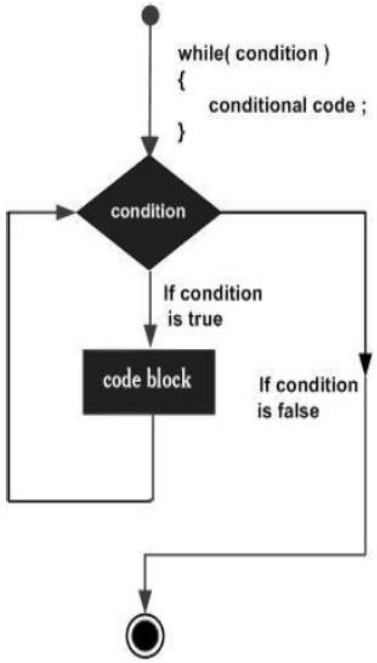
```
for (var val in list_of_values) {
   //statements
}
```

Indefinite Loop

while loop

Syntax:

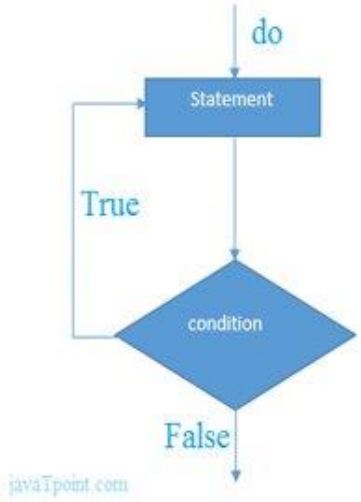
```
while(condition) {
   // statements if the
condition is true
}
```



do...while loop

Syntax:

```
do{
   //code to be executed
}while (condition);
```



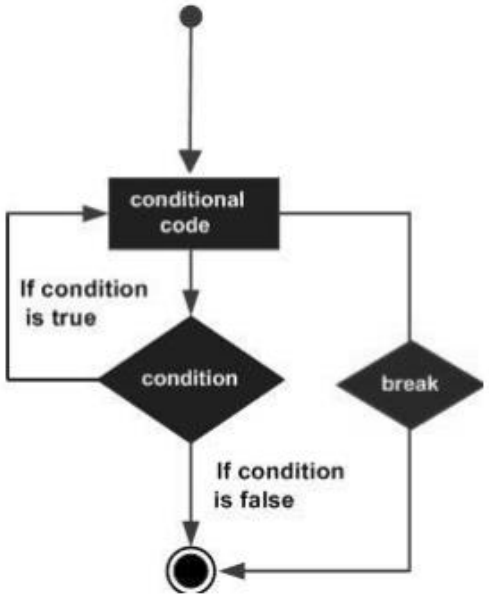
Break statement

Example:-

```
var i:number = 1
while(i<=10) {
  if (i % 5 == 0) {
    console.log (i)
    break    //exit the loop if the first multiple is found
  }
  i++
}
```

Output:-

5

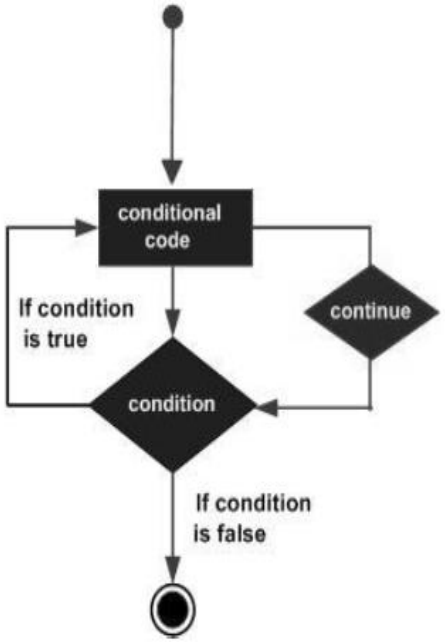


Example:-

```
var num:number = 0
var count:number= 0;

for(num=0;num<=20;num++) {
  if (num % 2==0) {
    continue
  }
  count++
}
console.log (count)
```

Output:-

10



Functions

- Specific parts of programs used to accomplish specific tasks.
- Generally, contain of three parts:
  i. Function name
  ii. Function parameters
  iii. Function body

```
TS app.ts > ...
function function_name(/*Function Parameters*/)
{
    //function body;
}
```

Function Parameters

- **Default Values:** For parameters that use a particular value frequently *(not always)*

```
TS app.ts > ...
function sum(a: number = 10, b: number = 20)
{
    return (a + b);
}

console.log(sum(20, 50));
console.log(sum(15));
console.log(sum());
```

```
Inspector   Console   Debugger
Filter Output
70
35
30
```

- **Union Types:** Assigning multiple types to a parameter using the pipe operator (|)

```
TS app.ts > ...
function sum(a: number | string, b: number | string)
{
    return (a + b);
}

console.log(sum(20, 50));
console.log(sum("Hello ", "World!"));
```

```
Inspector   Console   Debugger
Filter Output
70
Hello World!
```

Function Overloading

- Creating multiple functions with the *same name* but *different implementations*.

```
TS app.ts > type_2
1   function function_name(parameter_1: type_1, parameter_2: type_1)
2   function function_name(parameter_1: type_2, parameter_2: type_2)
3   function function_name(parameter_1: type_1 | type_2 , parameter_2: type_1 | type_2)
4   {
5       //Function body;
6   }
```

Final Episode:
OOP in TypeScript

Classes

Classes

Contents of a class:

i. Fields

ii. Methods

iii. Constructors

iv. Nested class or interface

TS app.ts > ...

```
class class_name {
    variables: <variable_type>;

    constructor () {

    };

    static variable_1 = class nested_class {

    };

    method_name () {
        //method contents
    };
}
```

Creation and Initialization of Objects

1. Creating Objects:

```
TS app.ts > ...
class class_name {
    variables: <variable_type>;
    constructor (parameter_1) {};
    method_name(parameter_2) {};
}

let parameter = "Any random parameter of any type";

let object_name = new class_name(parameter);
```

2. Initializing Objects:

```
TS app.ts > ...
class class_name {
    variable_1: <variable_type>;
    variable_2: <variable_type>;

    method_name(parameter) {};
}

let object_name = new class_name();

object_name.variable_1 = value_1;
object_name.variable_2 = value_2;
```

Reference Variable

```
TS app.ts > ...
class class_name {
    variable_1: <variable_type>;
    variable_2: <variable_type>;

    constructor (parameter_1, parameter_2) {
        this.variable_1 = parameter_1;
        this.variable_2 = parameter_2;
    }

    method_name(parameter) {};
}

let object_name = new class_name(parameter_1, parameter_2);
```

Constructors

Interfaces

Interfaces

- Act like contracts that describe data and the behavior of objects.
- Work only during compile time and not during runtime.
- Some properties can be made optional.
- **Method signatures:** Define the object's behaviors
  - Simply drop the 'function' keyword while declaring them.

```
TS app.ts > ...
interface interface_name {
    variable_1: <variable_type>;
    variable_2: <variable_type>;

    method_1();
    method_2();
}
```

Implementing Interfaces

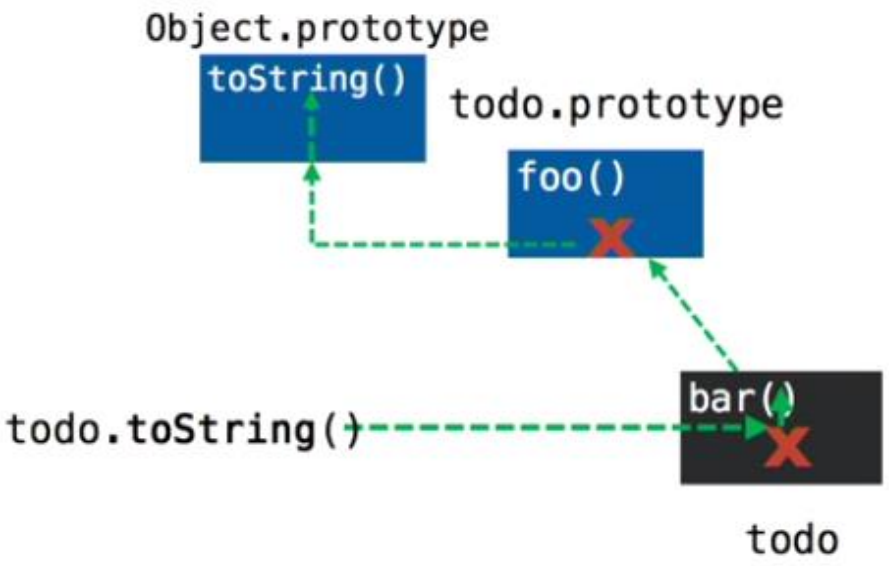
- Classes can implement interfaces to ensure that they follow the intended structure.

```
TS app.ts > ...
interface interface_name {
    variable_1: <variable_type>;
    variable_2: <variable_type>;

    method_1();
    method_2();
}

class class_name implements interface_name {
    variable_1: <variable_type>;
    variable_2: <variable_type>;

    method_1()
    {
        //contents of method
    }

    method_2()
    {
        //contents of method
    }
}
```

Inheritance

Prototypical Inheritance

- **Prototype object**: Contains that the information/properties that multiple objects will share.



Inheriting from Classes

- Classes can inherit from other classes which increases code reusability.
- Constructor, if defined, must be called on the base class.

```
TS app.ts > ...
class super_class {
    variable_1: <variable_type>;
    variable_2: <variable_type>;
    constructor (parameter_1, parameter_2) {
        this.variable_1 = parameter_1;
        this.variable_2 = parameter_2;

    }

    method_1() {
        //contents of method;
    }
}

class derived_class extends super_class {
    d_variable_1: <variable_type>;

    constructor(parameter_1, parameter_2, parameter_3) {
        /* calling the constructor of the
        super class */
        super(parameter_1, parameter_2);
        //adding to the constructor
        this.d_variable_1 = parameter_3;
    }

    d_method_1() {
        //contents of method;
    }
}
```

Abstract Classes

- Can derive new classes from it but cannot create new objects from that class

```
TS app.ts > ...
1   abstract class super_class {
2       variable_1: <variable_type>;
3       variable_2: <variable_type>;
4       constructor (parameter_1, parameter_2) {
5           this.variable_1 = parameter_1;
6           this.variable_2 = parameter_2;
7
8       }
9
10      method_1() {
11          //conte
12      }
13  }
14
15  let variable = new super_class();
16
17  class derived_class extends super_class {
18      d_variable_1: <variable_type>;
19
20      d_method_1() {
21          //contents of method;
22      }
23  }
```

```
constructor super_class(parameter_1: any, parameter_2: any): super_class
Cannot create an instance of an abstract class. ts(2511)
Peek Problem (Alt+F8)    No quick fixes available
```

`abstract` keyword

- Also used for methods to force derived classes to define their own implementation

```
TS app.ts > ...
1   abstract class super_class {
2       variable_1: <variable_type>;
3       variable_2: <variable_type>;
4       constructor(parameter_1, parameter_2) {
5           this.variable_1 = parameter_1;
6           this.variable_2 = parameter_2;
7       }
8
9       abstract method_1();
10  }
11          class d_class
12          Non-abstract class 'd_class' does not implement inherited abstract
13          member 'method_1' from class 'super_class'. ts(2515)
14          Peek Problem (Alt+F8)    Quick Fix... (Ctrl+.)
15  class d_class extends super_class {
16      d_variable_1: <variable_type>;
17
18      d_method_1() {
19          //contents of the method
20      }
21  }
```

Access Modifiers

`private`

i. Can be accessed only by the class they are declared in.

ii. There is no "private" access modifier in JS.

```
TS app.ts > ...
1  class super_class {
2      private variable_1: string;
3      variable_2: number;
4
5      method_1() {
6          //contents of method;
7      }
8  }
9
10 let object_1 = new super_class();
11 object_1.variable_1 = "shows an error";
12
13
14 class derived_class extends super_class {
15     d_variable_1: <variable_type>;
16
17                 (property) super_class.variable_1: string
18
19                 Property 'variable_1' is private and only accessible within class 'super_class'. ts(2341)
20     d_method_  Peek Problem (Alt+F8)   No quick fixes available
21         this.variable_1 = "shows error again";
22     }
23 }
```

`protected`

i. Can be accessed by the class and the classes inheriting from the in which they are declared.

```
TS app.ts > ...
1  class super_class {
2      protected variable_1: string;
3      variable_2: number;
4
5      method_1() {
6          /  (property) super_class.variable_1: string
7      }
8  }          Property 'variable_1' is protected and only accessible within class 'super_class' and
9             its subclasses. ts(2445)
10 let objec  Peek Problem (Alt+F8)   No quick fixes available
11 object_1.variable_1 = "shows an error";
12
13
14 class derived_class extends super_class {
15     d_variable_1: <variable_type>;
16
17     d_method_1() {
18         this.variable_1 = "shows no errors";
19     }
20 }
```

`public`

i. Can be accessed in any part of the program.

```
TS app.ts > ...
1  class super_class {
2      public variable_1: string;
3      variable_2: number;
4
5      method_1() {
6          //contents of method;
7      }
8  }
9
10 let object_1 = new super_class();
11 object_1.variable_1 = "shows no error";
12
13
14 class derived_class extends super_class {
15     d_variable_1: <variable_type>;
16
17     d_method_1() {
18         this.variable_1 = "shows no error";
19     }
20 }
```

Generics

Generics

- Functions or classes applicable to all types while retaining their information and functionalities
  - **Example:** Array type

```
TS app.ts > ...
1   function func_name<T> (parameter: T) {
2       //contents of the function;
3   }
```

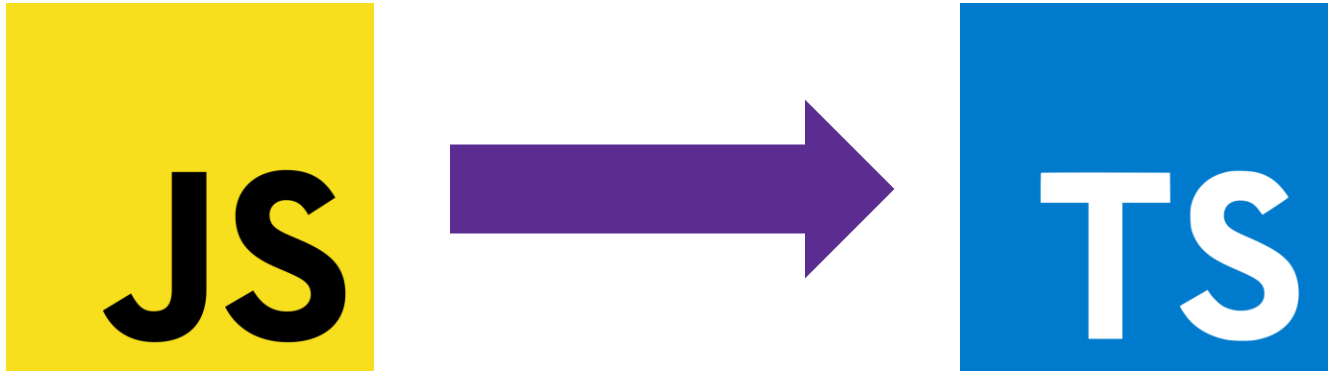
Generic functions

```
TS app.ts > ...
1   class class_name<T, U> {
2       variable_1: T;
3       variable_2: U;
4   }
```

Generic classes

Migrating from JS to TS

- `Tsconfig.json` file
- Copying JS file
- Correcting JS file



QnA

https://www.linkedin.com/in/shreyakhandelwal99/

https://www.linkedin.com/in/niralisahoo/

Microsoft Learn
STUDENT AMBASSADOR