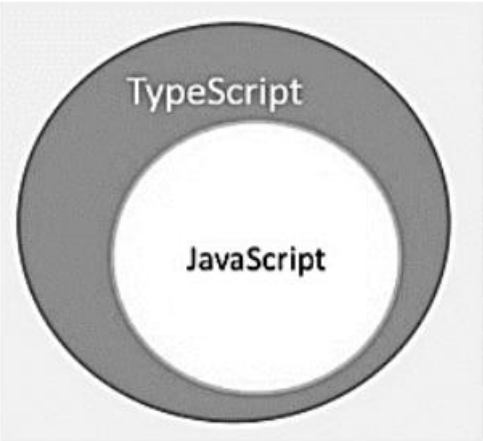
https://www.typescriptlang.org/

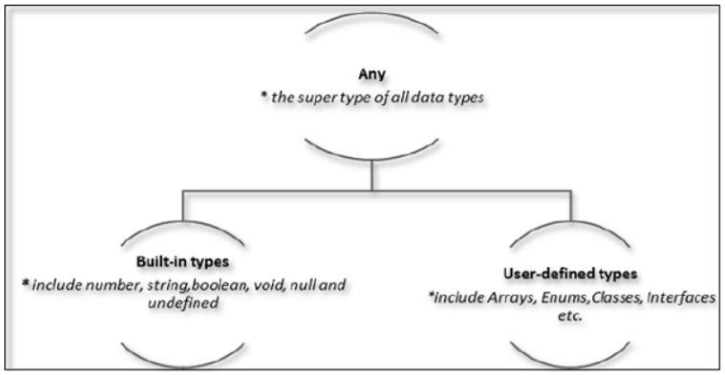
https://www.tutorialspoint.com/typescript/index.htm

JavaScript kliens oldali nyelv. A Node.js fejlődésével a JavaScript szerver oldali technológia is lett. A JavaScript kód növekedésével a kód egyre bonyolultabb és nehezebben újrahasznosítható lesz. Szükség lett az objektum orientáltságra, az erősen típusosságra. Ezt a TypeScript küszöböli ki.



- A typeScript valójában JavaScript: A TypeScript JavaScript-el kezdődik, és azzal is ér véget. A TypeScript a JavaScript nyelvi elemeit egészíti ki. Tehát igazából a TypeScript használatához végső soron elég a JavaScript-et ismerni. Az összesTypeScript kód JavaScript-re fordul le, mert a böngésző a JavaScript-et tudja értelmezni
- JavaScript az TypeScript: Ez azt jelenti, hogy bármilyen érvényes .js kiterjesztésű fájl-t át lehet írni .ts kiterjesztésűre, és le lehet fordítani más TypeScript fájllal .js-re.
- Fordítás: JavaScript interpreteres nyelv. Futtatni kell, hogy teszteljük, a szintaktikai hibákat is. Ez azt jelenti, hogy nem tudjuk, hogy nem kapunk hibaüzenetet legtöbbször, tehát hogy hol van pontosan a hiba. Ilyenkor órahosszakat kereshetjük a hibát. A TypeScript-nél viszont hibaüzenetet kapunk a fordítási idejű hibáknál. A TypeScript lefordítja a kódot, és fordítási hibát dob szintaktikai hibák esetén. Így a program futása előtt is értesülhetünk a hibák egy részéről.
- A TypeScript OOP koncepciót követ: classes, interfaces, öröklődés.

Típusok TypeScript-ben



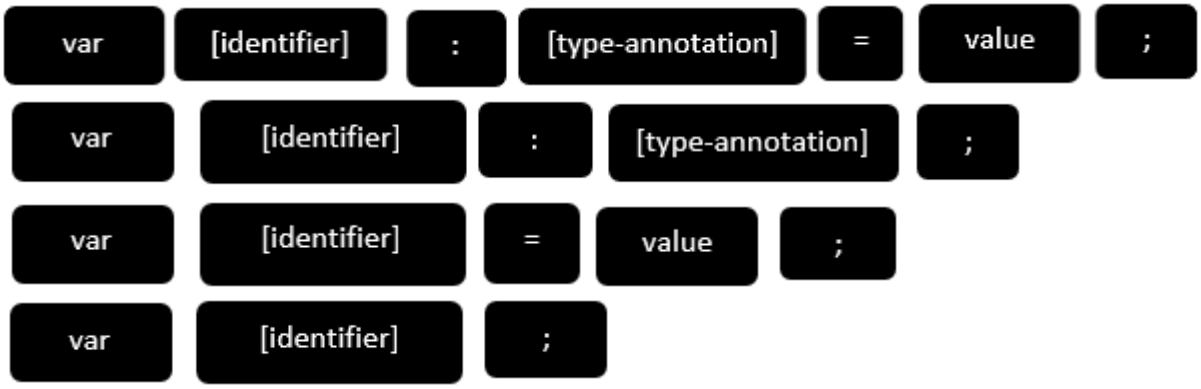
TypeScript-ben az any típus minden típus ősze. Dinamikus típust jelent. Az any típus használata a típusosság kikapcsolását jelenti egy változóra nézve.

Típusok: number, string, boolean, void, null, undefined.

Felhasználó által definiált típusok: enum, osztályok, interface.

Változók

Deklaráció, értékadás



S.No.	Variable Declaration Syntax & Description
1.	**var name:string = ”mary”**
2.	**var name:string;**
3.	**var name = ”mary”**
4.	**var name;**

```
var name:string = "John";
var score1:number = 50;
var score2:number = 42.50
var sum = score1 + score2
console.log("name"+name)
console.log("first score: "+score1)
console.log("second score: "+score2)
console.log("sum of the scores: "+sum)
```

Inferred Typing in TypeScript

Habár TypeScript erősen típusos nyelv, ez csak opcionális. Dinamikus típusosságot is megenged, ami ezt jelenti, hogy típus nélkül is deklarálhatunk egy változót. Ekkor a forító maga határozza meg a változó típusát aszerint, hogy milyen értéket adtunk neki. Az első értékadás lesz a mérvadó a típus meghatározásában.

```
var num = 2;    // data type inferred as  number
console.log("value of num "+num);
num = "12"; //Type '"12"' is not assignable to type 'number'.
console.log(num);
```

TypeScript Variable Scope

Az, hogy a változót hogy definiáltuk, meghatározza a scope-ját. A következő scope-ok lehetségesek:

- Globális (Global) scope: globális változók a programozási struktúrákon kívül deklaráljuk, és bárhonnan elérhetőek.
- Osztály (Class) scope: osztály adattagjainak is hívjuk őket. Az osztályon belül deklaráljuk őket, de metódusokon kívül. Statikus is lehet. Az osztály példányosításával vagy az osztálynévvel (static esetben) hivatkozhatunk rájuk.
- Lokális (Local) scope: blokkon belül pl. metódus, ciklus stb deklarált változók. Csak a blokkban érhetőek el.

```
var global_num = 12          //global variable
class Numbers {
   num_val = 13;             //class variable
   static sval = 10;         //static field

   storeNum():void {
      var local_num = 14;    //local variable
   }
}
console.log("Global num: "+global_num)
console.log(Numbers.sval)   //static variable
var obj = new Numbers();
console.log("Global num: "+obj.num_val)
```

Operátorok

The major operators in TypeScript can be classified as −

- Arithmetic operators
- Logical operators
- Relational operators
- Bitwise operators
- Assignment operators
- Ternary/conditional operator
- String operator
- Type Operator

Arithmetic Operators

Operator	Description	Example
+ (Addition)	returns the sum of the operands	a + b is 15
- (Subtraction)	returns the difference of the values	a - b is 5
* (Multiplication)	returns the product of the values	a * b is 50
/ (Division)	performs division operation and returns the quotient	a / b is 2
% (Modulus)	performs division operation and returns the remainder	a % b is 0
++ (Increment)	Increments the value of the variable by one	a++ is 11
-- (Decrement)	Decrements the value of the variable by one	a-- is 9

```
var num1:number = 10
var num2:number = 2
var res:number = 0

res = num1 + num2
console.log("Sum:         "+res);
```

```
res = num1 - num2;
console.log("Difference: "+res)

res = num1*num2
console.log("Product:    "+res)

res = num1/num2
console.log("Quotient:   "+res)

res = num1%num2
console.log("Remainder:   "+res)

num1++
console.log("Value of num1 after increment "+num1)

num2--
console.log("Value of num2 after decrement "+num2)
```

Relational Operators

Operator	Description	Example
>	Greater than	(A > B) is False
<	Lesser than	(A < B) is True
>=	Greater than or equal to	(A >= B) is False
<=	Lesser than or equal to	(A <= B) is True
==	Equality	(A == B) is false
!=	Not equal	(A != B) is True

```
var num1:number = 5;
var num2:number = 9;

console.log("Value of num1: "+num1);
console.log("Value of num2 :"+num2);

var res = num1>num2
console.log("num1 greater than num2: "+res)
```

```
res = num1<num2
console.log("num1 lesser than num2: "+res)

res = num1>=num2
console.log("num1 greater than or equal to  num2: "+res)

res = num1<=num2
console.log("num1 lesser than or equal to num2: "+res)

res = num1==num2
console.log("num1 is equal to num2: "+res)

res = num1!=num2
console.log("num1 is not equal to num2: "+res)
```

Logical Operators

Operator	Description	Example
&& (And)	The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False
|| (OR)	The operator returns true if at least one of the expressions specified return true	(A > 10 || B >10) is True
! (NOT)	The operator returns the inverse of the expression's result. For E.g.: !(>5) returns false	!(A >10) is True

```
var avg:number = 20;
var percentage:number = 90;

console.log("Value of avg: "+avg+" ,value of percentage:
"+percentage);

var res:boolean = ((avg>50)&&(percentage>80));
console.log("(avg>50)&&(percentage>80): ",res);

var res:boolean = ((avg>50)||(percentage>80));
console.log("(avg>50)||(percentage>80): ",res);

var res:boolean=!((avg>50)&&(percentage>80));
console.log("!((avg>50)&&(percentage>80)): ",res);
```

Assignment Operators

Operator	Description	Example
= (Simple Assignment)	Assigns values from the right side operand to the left side operand	C = A + B will assign the value of A + B into C
+= (Add and Assignment)	It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-= (Subtract and Assignment)	It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*= (Multiply and Assignment)	It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/= (Divide and Assignment)	It divides the left operand with the right operand and assigns the result to the left operand.	

```
var x:number = 4
var y = -x;
console.log("value of x: ",x);    //outputs 4
console.log("value of y: ",y);    //outputs -4
```

```
var msg:string = "hello"+"world"
console.log(msg)
```

Conditional Operator (?)

```
Test ? expr1 : expr2
```

- **Test** − feltétel
- **expr1** – ha igaz a feltétel, akkor ezzel tér vissza
- **expr2** – ha hamis a feltétel, akkor ezzel tér vissza

```
var num:number = -2
var result = num > 0 ?"positive":"non-positive"
console.log(result)
```

Type Operators

typeof operator

```
var num = 12
```

```
console.log(typeof num);    //output: number
```

IF,SWITCH

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

```
var  num:number = 5
if (num > 0) {
   console.log("number is positive")
}
```

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

```
var num:number = 12;
if (num % 2==0) {
   console.log("Even");
} else {
   console.log("Odd");
}
```

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

```
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;
    }
}
```

LOOPS

```
for (initial_count_value; termination-condition; step) {
    //statements
}
```

```
var num:number = 5;
var i:number;
var factorial = 1;

for(i = num;i>=1;i--) {
    factorial *= i;
}
console.log(factorial)
```

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

```
var j:any;
var n:any = "a b c"

for(j in n) {
    console.log(n[j])
}
```

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

```
}
```

```
var num:number = 5;
var factorial:number = 1;

while(num >=1) {
   factorial = factorial * num;
   num--;
}
console.log("The factorial  is "+factorial);
```

```
do {
   //statements
} while(condition)
```

```
var n:number = 10;
do {
   console.log(n);
   n--;
} while(n>=0);
```

FUNCTIONS

```
function  function_name() {
   // function body
}
```

```
function () {
   //function definition
   console.log("function called")
}
```

```
function test() {    // function definition
   console.log("function called")
}
test()               // function invocation
```

```
function function_name():return_type {
   //statements
   return value;
}
```

```
//function defined
function greet():string { //the function returns a string
```

```
   return "Hello World"
}

function caller() {
   var msg = greet() //function greet() invoked
   console.log(msg)
}

//invoke function
caller()
```

```
function func_name( param1 [:datatype], ( param2 [:datatype]) {
}
```

```
function test_param(n1:number,s1:string) {
   console.log(n1)
   console.log(s1)
}
test_param(123,"this is a string")
```

Optional Parameters

```
function disp_details(id:number,name:string,mail_id?:string) {
   console.log("ID:", id);
   console.log("Name",name);

   if(mail_id!=undefined)
   console.log("Email Id",mail_id);
}
disp_details(123,"John");
disp_details(111,"mary","mary@xyz.com");
```

Default Parameters

```
function function_name(param1[:type],param2[:type] =
default_value) {
}
```

```
function calculate_discount(price:number,rate:number = 0.50) {
   var discount = price * rate;
   console.log("Discount Amount: ",discount);
}
calculate_discount(1000)
calculate_discount(1000,0.30)
```

Függvény túlterhelés

- **Paraméter adattípusa**

```
function disp(string):void;
function disp(number):void;
```

- **Paraméterek száma**

```
function disp(n1:number):void;
function disp(x:number,y:number):void;
```

- **Paraméterek sorrendje**

```
function disp(n1:number,s1:string):void;
function disp(s:string,n:number):void;
```

Tömbök

```
var array_name[:datatype];        //declaration
array_name = [val1,val2,valn..]   //initialization
```

```
var alphas:string[];
alphas = ["1","2","3","4"]
console.log(alphas[0]);
console.log(alphas[1]);
```

```
var nums:number[] = [1,2,3,3]
console.log(nums[0]);
console.log(nums[1]);
console.log(nums[2]);
console.log(nums[3]);
```

Tuples

Lehetséges, hogy változó típusú adatokat szeretnénk egy kollekcióban tárolni. A tömb erre nem alkalmas. A TypeScript erre a tuple-t alkalmazza. Ez heterogén típusú értékek kollekcióját tartalmazza. Más szavakkal, a tuple lehetővé teszi különböző típusú értékek tárolását. A tuple akár függvény paramétere is lehet.

```
var tuple_name = [value1,value2,value3,…value n]
```

```
var mytuple = [10,"Hello"]; //create a  tuple
console.log(mytuple[0])
console.log(mytuple[1])
```

```
var mytuple = [10,"Hello","World","typeScript"];
console.log("Items before push "+mytuple.length)     // returns
the tuple size

mytuple.push(12)                                     // append
value to the tuple
console.log("Items after push "+mytuple.length)
console.log("Items before pop "+mytuple.length)
console.log(mytuple.pop()+" popped from the tuple") // removes
and returns the last item

console.log("Items after pop "+mytuple.length)
```

Union

A TypeScript lehetővé teszi, hogy egy vagy több típust kombináljuk, a | operátor használatával. Ilyenkor valójában felsoroljuk, hogy egy változó milyen típusú lehet, egy függvény milyen típust vár paraméterként.

```
Type1|Type2|Type3
```

```
var val:string|number
val = 12
console.log("numeric value of val "+val)
val = "This is a string"
console.log("string value of val "+val)
```

```
function disp(name:string|string[]) {
   if(typeof name == "string") {
      console.log(name)
   } else {
      var i;

      for(i = 0;i<name.length;i++) {
         console.log(name[i])
      }
   }
}
disp("mark")
console.log("Printing names array....")
disp(["Mark","Tom","Mary","John"])
```

```
var arr:number[]|string[];
var i:number;
arr = [1,2,4]
console.log("**numeric array**")

for(i = 0;i<arr.length;i++) {
   console.log(arr[i])
```

```
}

arr = ["Mumbai","Pune","Delhi"]
console.log("**string array**")

for(i = 0;i<arr.length;i++) {
   console.log(arr[i])
}
```

Interfaces

```
interface interface_name {
}
```

```
interface IPerson {
   firstName:string,
   lastName:string,
   sayHi: ()=>string
}

var customer:IPerson = {
   firstName:"Tom",
   lastName:"Hanks",
   sayHi: ():string =>{return "Hi there"}
}

console.log("Customer Object ")
console.log(customer.firstName)
console.log(customer.lastName)
console.log(customer.sayHi())

var employee:IPerson = {
   firstName:"Jim",
   lastName:"Blakes",
   sayHi: ():string =>{return "Hello!!!"}
}

console.log("Employee  Object ")
console.log(employee.firstName);
console.log(employee.lastName);
```

Classes

```
class class_name {
   //class scope
}
```

Az osztály definíció az alábbiakat tartalmazhatja:

- **Adattag**
- **Konstruktor**
- **Metódus**

```
class Car {
   //field
   engine:string;

   //constructor
   constructor(engine:string) {
      this.engine = engine
   }

   //function
   disp():void {
      console.log("Engine is  :   "+this.engine)
   }
}
```

Objektum példány létrehozása

```
var object_name = new class_name([ arguments ])
```

```
var obj = new Car("Engine 1")
```

Adattagok, metódusok hozzáférése

```
//accessing an attribute
obj.field_name

//accessing a function
obj.function_name()
```

Öröklődés

Syntax

```
class child_class_name extends parent_class_name
```

TypeScript nem támogat többszörös öröklődést.

```
class Shape {
   Area:number

   constructor(a:number) {
      this.Area = a
```

```
    }
}

class Circle extends Shape {
    disp():void {
        console.log("Area of the circle:  "+this.Area)
    }
}

var obj = new Circle(223);
obj.disp()
```

Metódus felüldefiniálás

```
class PrinterClass {
    doPrint():void {
        console.log("doPrint() from Parent called…")
    }
}

class StringPrinter extends PrinterClass {
    doPrint():void {
        super.doPrint()
        console.log("doPrint() is printing a string…")
    }
}

var obj = new StringPrinter()
obj.doPrint()
```

Statikus változó

```
class StaticMem {
    static num:number;

    static disp():void {
        console.log("The value of num is"+ StaticMem.num)
    }
}

StaticMem.num = 12     // initialize the static variable
StaticMem.disp()      // invoke the static method
```

instanceof operator

```
class Person{ }
var obj = new Person()
var isPerson = obj instanceof Person;
console.log(" obj is an instance of Person " + isPerson);
```

Adatrejtés

Itt is lehet az adattag public, private, protected

```
class Encapsulate {
   str:string = "hello"
   private str2:string = "world"
}

var obj = new Encapsulate()
console.log(obj.str)      //accessible
console.log(obj.str2)    //compilation Error as str2 is private
```