آموزش Typescript – آموزش توابع در تایپ اسکریپت(Typescript)

توابع بلوک‌هایی ساختاری از کدهایی است که این کدها قابل خواندن، قابل نگهداری و قابل استفاده‌ی مجدد هستند. تابع مجموعه‌ای از دستورات است که کار مشخصی را انجام می‌دهد. توابع برنامه را به بلوک‌هایی منطقی از کد سازمان‌دهی می‌کند. بعد از اینکه توابع تعریف شوند، می‌توان آن‌ها را برای دسترسی به کد فراخوانی کرد. این کار باعث می‌شود که کد قابلیت استفاده‌ی مجدد را داشته باشد. علاوه بر این، توابع کار خواندن و نگهداری کد برنامه را آسان می‌کنند.

بعد از اینکه تابعی اعلان شود، کامپایلر اسم، نوع برگشت و پارامترهای تابع را دریافت می‌کند. تعریف تابع باعث می‌شود که بدنه‌ی واقعی تابع در اختیار قرار گیرد.

ردیف	توابع و توضیحات
.1	تعریف کردن تابع تعریف کردن تابع باعث می‌شود که مشخص شود چه کاری و چگونه قرار است انجام شود.
.2	فراخوانی تابع برای اینکه بتوان تابع را اجرا کرد، باید آن را فراخوانی کرد.
.3	توابع برگشتی برخی از توابع مقدار خود را همراه با کنترل به فراخواننده ی آن برگشت می‌دهند.
.4	توابع پارامتری پارامترها مکانیزمی برای تحویل دادن مقادیر به توابع هستند.

آموزش پارامترهای اختیاری در Typescript

این پارامترها در مواقعی مورد استفاده قرار می‌گیرند که نیازی به تحویل دادن اجباری آرگومان‌ها به توابع برای اجرای آن توابع وجود نداشته باشد. برای اینکه مشخص کنید پارامتری اختیاری است می‌توانید علامت سوالی را در کنار اسم آن قرار دهید. پارامترهای اختیاری باید به عنوان آخرین آرگومان تابع تنظیم شوند. سینتکس اعلان تابع با پارامتر اختیاری را می‌توانید در زیر مشاهده کنید.

```
function function_name (param1[:type], param2[:type], param3[:type])
```

مثال: پارامترهای اختیاری

```
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");
```

- در مثال بالا یک تابع پارامتری اعلان شده است. در این مثال پارامتر سوم (mail_id) یک متغیر اختیاری است.
- در صورتی که پارامتری اختیاری طی فراخوانی تابع مقداری را عبور ندهد، مقدار پارامتر بر روی تعریف نشده تنظیم می‌شود.
- این تابع تنها در صورتی که آرگومان مقداری را از خود عبور دهد، مقدار mail_id را چاپ می‌کند.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10
function disp_details(id, name, mail_id) {
 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");
```

و خروجی زیر نمایش داده می‌شود.

```
ID:123
Name John
ID: 111
Name mary
```

Email Id mary@xyz.com

آموزش پارامترهای Rest در TypeScript

این پارامترها مشابه آرگومان‌های متغیر در جاوا هستند. پارامترهای rest در تعداد مقادیری که شما می‌توانید به تابعی عبور دهید، محدودیت ایجاد نمی‌کنند. با این حال نوع مقادیر عبوری تماماً باید یکسان باشد، به بیان دیگر پارامترهای rest به ازای آرگومان‌های متعدد با نوعی یکسان، به عنوان placeholder عمل می‌کنند.

برای اینکه بتوانید این پارامترها را اعلان کنید، اسم پارامتر باید پیش از سه period بیاید. تمامی پارامترهای غیر rest باید قبل از پارامترهای rest بیایند.

مثال: پارامترهای rest

```
function addNumbers(...nums:number[]) {
 var i;
 var sum:number = 0;

 for(i = 0;i<nums.length;i++) {
 sum = sum + nums[i];
 }
 console.log("sum of the numbers",sum)
}
addNumbers(1,2,3)
addNumbers(10,10,10,10,10)
```

- اعلان تابع addNumbers() باعث می‌شود که پارامتر rest، nums، پذیرفته شود. نوع داده‌ی پارامتر rest باید بر روی آرایه قرار گیرد. علاوه بر این، یک تابع باید حداکثر یک پارامتر rest داشته باشد.
- این تابع به ترتیب با عبور دادن مقادیر 3 و 6 دوبار احضار می‌شود.

- حلقه‌ی for در سراسر لیست آرگومان‌هایی که به تابع تحویل داده شده‌اند تکرار شده و مجموع آن‌ها را محاسبه می‌کند.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
function addNumbers() {

 var nums = [];

 for (var _i = 0; _i < arguments.length; _i++) {

 nums[_i - 0] = arguments[_i];

 }

                var i;

 var sum = 0;



 for (i = 0; i < nums.length; i++) {

 sum = sum + nums[i];

 }

 console.log("sum of the numbers", sum);

}

addNumbers(1, 2, 3);

addNumbers(10, 10, 10, 10, 10);
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
sum of numbers 6
sum of numbers 50
```

آموزش پارامترهای پیش فرض در TypeScript

به پارامترهای توابع می‌توان به صورت پیش فرض نیز مقادیری اختصاص داد. با این حال به چنین پارامترهایی می‌توان مقادیر غیر پیش فرضی نیز اختصاص داد.

سینتکس

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

نکته: یک پارامتر نمی‌تواند به طور همزمان، به صورت اختیاری و پیش فرض اعلان شود.

مثال: پارامترهای پیش فرض

```
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)
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10
function calculate_discount(price, rate) {
 if (rate === void 0) { rate = 0.50; }
 var discount = price * rate;
 console.log("Discount Amount: ", discount);
}
calculate_discount(1000);
calculate_discount(1000, 0.30);
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
Discount amount: 500
Discount amount: 300
```

- **در مثال بالا، تابع *calculate_discount* اعلان می‌شود. این تابع دو پارامتر به نام‌های price و rate دارد.**

- مقدار پارامتر rate به صورت پیش فرض بر روی 0.5 قرار گرفته است.
- این برنامه تابع *calculate_discount* را احضار کرده و تنها مقدار پارامتر price را به آن تحویل می‌دهد. در اینجا مقدار rate همان مقدار پیش فرض 0.5 است.
- همان تابع احضار می‌شود، اما این بار با دو آرگومان. مقدار پیش فرض rate، overwrite شده و بر روی مقدار غیر پیش فرض عبوری تنظیم می‌شود.

آموزش توابع بی نام در TypeScript

توابعی که محدود به یک شناسه (نام تابع) نباشند، توابع بی نام نامیده می‌شوند. این توابع به صورت دینامیکی در runtime اجرا می‌شوند. این توابع درست مانند توابع استاندارد می‌توانند ورودی‌ها را بپذیرند و خروجی‌ها را برگشت دهند. این توابع بعد از اینکه برای بار اول ایجاد می‌شوند، معمولاً از دسترس خارج می‌شوند.

به متغیرها می‌توان یک تابع بی نام را اختصاص داد. به چنین کاری، بیان تابع گفته می‌شود.

سینتکس

```
var res = function([arguments]) { ... }
```

مثال: تابع بی نام ساده

```
var msg = function() {
 return "hello world";
}
console.log(msg())
```

بعد از کامپایل کردن کد بالا، همین کد در جاوا اسکریپت ایجاد می‌شود.

و خروجی به صورت زیر نمایش داده می‌شود.

```
hello world
```

مثال: تابع بی نام با پارامتر

```
var res = function(a:number,b:number) {
 return a*b;
};
console.log(res(12,2))
```

این تابع بی نام حاصل ضرب مقادیر داده شده به آن را برگشت می‌دهد.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

var res = function (a, b) {

 return a * b;

};

console.log(res(12, 2));
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
24
```

بیان و اعلان تابع در TypeScript – آیا این دو یک معنی را می‌دهند؟

جواب منفی است. اعلان تابع برخلاف بیان تابع، مقید به اسم تابع است.

تفاوت اساسی بین این دو، این است که اعلان تابع قبل از اجرا شدن آن تجزیه می‌شود، این در حالی است که بیان توابع تنها زمانی تجزیه می‌شود که موتور متن (script engine) طی اجرا به آن برخورد کند.

زمانی که تجزیه گر جاوا اسکریپت در جریان کد اصلی به تابعی برخورد می‌کند، فرض را بر اعلان شدن تابع می‌گذارد. اما زمانی که تابع به عنوان بخشی از دستور ظاهر شود، به این کار بیان تابع گفته می‌شود.

آموزش سازنده‌ی تابع (Function Constructor) در TypeScript

در تایپ اسکریپت می‌توان تابعی را با استفاده از constructor موجود در خود جاوا اسکریپت، تعریف کرد. به این constructor، Function () گفته می‌شود.

سینتکس

```
var res = new Function([arguments]) { ... }.
```

مثال

```
var myFunction = new Function("a", "b", "return a * b");

var x = myFunction(4, 3);

console.log(x);
```

new Function() فراخوانی constructor محسوب می‌شود، که مرجع تابعی را ایجاد کرده و آن را برگشت می‌دهد.

بعد از کامپایل کردن کد بالا، همین کد به زبان جاوا اسکریپت ایجاد می‌شود.

و خروجی به صورت زیر نمایش داده می‌شود.

```
12
```

آموزش بازگشت در توابع تایپ اسکریپت(TypeScript)

بازگشت روشی برای تکرار عملیاتی خاص با ارجاع دادن تابع به خود، به صورت مکرر است، تا زمانی که این تابع به نتیجه‌ی مطلوبی برسد. بهترین جایی که می‌توانید از بازگشت استفاده کنید، زمانی است که نیاز دارید تابع یکسانی را که دارای پارامترهای مختلفی است از داخل یک حلقه به صورت مکرر فراخوانی کنید.

مثال: بازگشت

```
function factorial(number) {

 if (number <= 0) {// termination case

 return 1;

 } else {

 return (number * factorial(number - 1));// function invokes itself

 }

};

console.log(factorial(6));// outputs 720
```

بعد از کامپایل کردن کد بالا، همین کد به زبان جاوا اسکریپت ایجاد می‌شود.

و خروجی به صورت زیر نمایش داده می‌شود.

```
720
```

مثال: تابعی بازگشتی بی نام

```
(function () {

 var x = "Hello!!";
```

```
console.log(x)

})()// the function invokes itself using a pair of parentheses ()
```

بعد از کامپایل کردن کد بالا، همین کد به زبان جاوا اسکریپت ایجاد می‌شود.

و خروجی به صورت زیر نمایش داده می‌شود.

```
Hello!!
```

آموزش توابع لامبدا در TypeScript

لامبدا به توابع بی نام در برنامه نویسی اشاره دارد. توابع لامبدا، مکانیزمی مختصر برای بیان توابع بی نام می‌باشند. به این توابع، توابع برداری (Arrow function) می گویند.

ساختار توابع لامبدا

این توابع به سه بخش تقسیم می‌شوند:

- پارامترها: توابع می‌توانند به صورت اختیاری دارای پارامتر باشند.
- علامت بردار بزرگ یا علامت لامبدا (=>): به این علامت عملگر " Go to" گفته می‌شود.
- دستورات: دستورالعمل‌های تابع را نشان می‌دهد.

نکته: به صورت قراردادی بهتر است که برای جمع و جور کردن و دقیق کردن اعلان تابع، از پارامترهای تک حرفی استفاده شود.

آموزش بیان توابع لامبدا در TypeScript

این کار برای بیان توابع بی نامی استفاده می‌شود که این به یک خط از کد اشاره دارند. سینتکس آن به صورت زیر است:

```
([param1, parma2,...param n])=>statement;
```

مثال: بیان تابع لامبدا

```
var foo = (x:number)=>10 + x

console.log(foo(100))//outputs 110
```

در این برنامه تابع بیان لامبدا اعلان می‌شود. این تابع مجموع ده آرگومان و آرگومان عبوری را برگشت می‌دهد.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

var foo = function (x) { return 10 + x; };

console.log(foo(100));//outputs 110
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
110
```

دستور لامبدا

دستور لامبدا اعلان تابع بی نامی است که به بلوکی از کد اشاره دارد. این سینتکس زمانی کاربرد دارد که بدنه‌ی تابع بیش از یک خط را اشغال کند. سینتکس آن به صورت زیر است:

```
([param1, parma2,...param n])=> {

//code block
}
```

مثال: دستور لامبدا

```
var foo = (x:number)=> {

 x = 10 + x

 console.log(x)

}

foo(100)
```

مرجع تابع برگشت داده شده و در متغیر foo ذخیره می‌شود.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

var foo = function (x) {

 x = 10 + x;

 console.log(x);
```

```
};

foo(100);
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
110
```

آموزش اختلافات مربوط به سینتکس در TypeScript

استنتاج نوع پارامتر

اجباری در مشخص کردن نوع داده‌ی یک پارامتر وجود ندارد. در چنین مواردی نوع داده‌ی پارامتر any می‌باشد. بیایید نگاهی به کد زیر بیاندازیم.

```
var func = (x)=> {
 if(typeof x=="number") {
 console.log(x+" is numeric")
 } else if(typeof x=="string") {
 console.log(x+" is a string")
 }
}
func(12)
func("Tom")
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10
var func = function (x) {
 if (typeof x == "number") {
 console.log(x + " is numeric");
 } else if (typeof x == "string") {
```

```
 console.log(x + " is a string");

 }

};

func(12);

func("Tom");
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
12 is numeric
Tom is a string
```

پرانتزهای اختیاری برای یک پارامتر تنها

```
var display = x=> {

 console.log("The function got "+x)

}

display(12)
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

var display = function (x) {

 console.log("The function got " + x);

};

display(12);
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
The function got 12
```

آکولادهای اختیاری برای یک دستور تنها، پرانتزهای خالی برای مواقع عدم وجود پارامتر

مثال زیر دو اختلاف مربوط به سینتکس را نشان می‌دهد.

```
var disp =()=> {

 console.log("Function invoked");

}

disp();
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

var disp = function () {

 console.log("Function invoked");

};

disp();
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
Function invoked
```

آموزش Overload توابع در TypeScript

توابع این قابلیت را دارند تا بر اساس ورودی داده شده به آن‌ها متفاوت عمل کنند. به بیان دیگر، برنامه‌ها می‌توانند نام یکسان، متدهای متعدد و پیاده سازی های مختلف، داشته باشند. به این مکانیزم Function Overloading گفته می‌شود. تایپ اسکریپت از این مکانیزم پشتیبانی می‌کند.

برای اینکه بتوان در تایپ اسکریپت تابعی را overload کرد، می‌توان از مراحل زیر پیروی کرد.

مرحله‌ی اول: توابع متعددی را با نام یکسان اما با امضای تابع مختلف اعلان کنید. امضای تابع شامل موارد زیر است:

- **نوع داده‌ی پارامتر**

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

- **تعداد پارامترها**

```
function disp(n1:number):void;
```

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

- ترتیب پارامترها

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

نکته: امضای تابع شامل نوع برگشت تابع نمی‌شود.

مرحله دوم: اعلان تابع باید همراه با تعریف تابع انجام شود. اگر نوع پارامترها طی مکانیزم overloading متفاوت باشند، نوع پارامترها باید بر روی any تنظیم شود. علاوه بر این، برای حالت b که در بالا توضیح داده شده است، می‌توانید طی تعریف تابع یک یا چند پارامتر را به عنوان پارامتر اختیاری لحاظ کنید.

مرحله سوم: در نهایت، برای اینکه این تابع کار خود را آغاز کند، باید تابع را احضار کنید.

مثال

بیایید به مثال زیر نگاهی بیاندازیم.

```
function disp(s1:string):void;

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


function disp(x:any,y?:any):void {

 console.log(x);

 console.log(y);

}

disp("abc")

disp(1,"xyz");
```

- **دو خط اول اعلان overload تابع را نشان می‌دهد. این تابع دارای دو overload است:**
  - **تابعی که تنها یک پارامتر رشته‌ای را قبول می‌کند.**
  - **تابعی که دو مقدار را قبول می‌کند، که به ترتیب نوع یک از آن‌ها عدد و دیگری رشته‌ای است.**
- **خط سوم تابع را تعریف می‌کند. نوع داده‌ی پارامترها بر روی any تنظیم شده‌اند. علاوه بر این، پارامتر دوم در اینجا اختیاری است.**

- تابع overload شده توسط دو دستور آخر احضار می‌شوند.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می‌شود.

```
//Generated by typescript 1.8.10

function disp(x, y) {

 console.log(x);

 console.log(y);

}

disp("abc");

disp(1, "xyz");
```

و خروجی به صورت زیر نمایش داده می‌شود.

```
abc
1
xyz
```