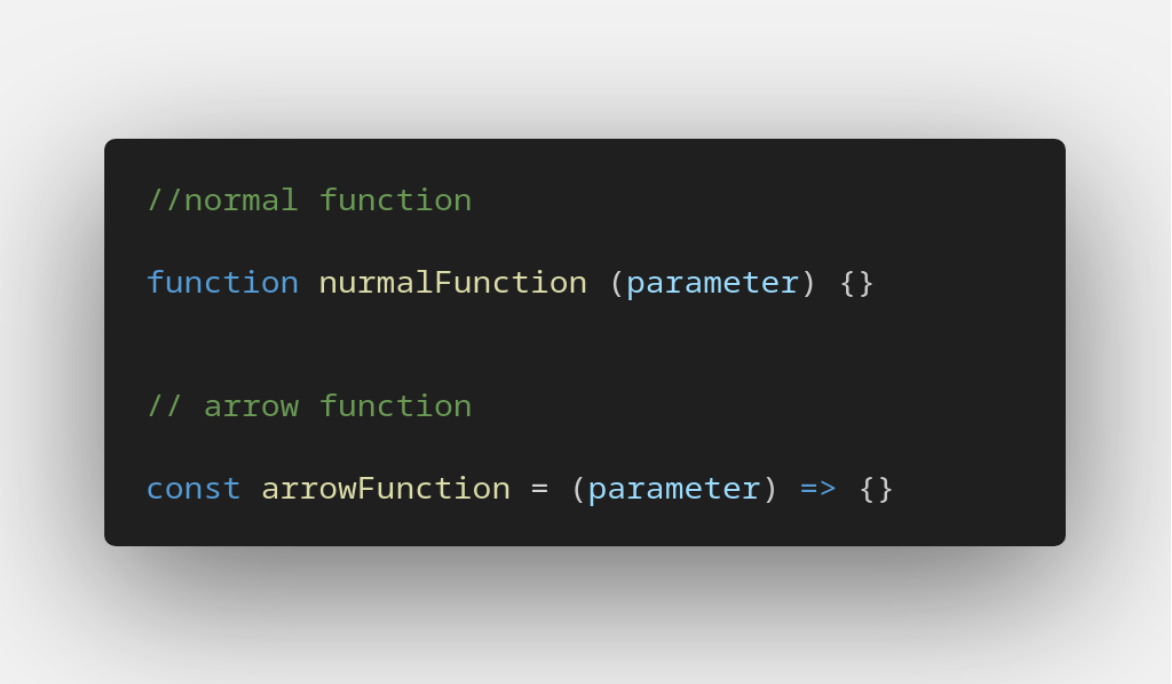
تفاوت های Arrow Function و Normal Function در javascript

تفاوت عمده ای که در نگاه اول دیده میشه تفاوت سینتکس هست ؛ میدانیم که Normal function در ES5 بوده و استفاده میشده و اما Arrow function در ES6 به بعد به جاوا اسکریپت اضافه شده است .

اما این موضوع ها را میدانستیم بپردازیم به تفاوت های ساختاری این دو :

۱- نحوه تعریف



```
//normal function

function nurmalFunction (parameter) {}


// arrow function

const arrowFunction = (parameter) => {}
```

- در arrow function شما میتوانید با نوشتن تکه خط های کوتاه تر به جواب مد نظر برسید. حتی در تعیین آرگومان های ورودی هم میتوان این خلاصه نویسی را داشت .
و در انتهای آن شما میتونید هیچ آرگومانی نداشته باشید . و از این سینتکس استفاده کنید .

```
//normal function

function nurmalFunction (arg = 0) {
    return "hello js " + arg
}


// arrow function

const arrowFunction = arg => "hello js " +
arg

// without arguments

const arrowFn = _ => console.log("hello js")
```

۲- تفاوت در اتصال (bind) کردن آرگومان ها :

اگر نمیدانید آبجکت arguments چی هست من برای شما به صورت خلاصه میگویم ؛ این یک آبجکت یک شی از تمام آرگومان های ورودی فانکشن شما هست به هر مقدار که آرگومان داشته باشید می توانید در یک آبجکت ببینید . که در arrow function به آن دسترسی ندارید و با این خطا مواجه می شوید :

```
Uncaught ReferenceError: arguments is not defined
```

```
//normal function

let examFunction = {
    nurmalFunction (arguments) {
        console.log(arguments)
// return arguments object
    }
}
examFunction.nurmalFunction("hello" , "js")
// arrow function


let examFunction = {
    arrowFunction : () => {
        console.log(...arguments)
// Uncaught ReferenceError: arguments is not
defined
    }
}

examFunction.arrowFunction("hello" , "js")
```

۳- اشاره به آبجکت جاری یا همان this :

شما در normal function دسترسی به آبجکت جاری دارید و میتوانید از آن استفاده کنید ولی در arrow function دسترسی شما محدود می باشد .

```
let myInfo = {

 name: "mohsen salehi",

 arrowFunction:() => {

   return ("My name is " + this.name); // no 'this' binding here
 },

 normalFunction(){
   return("My name is " + this.name); // 'this' binding works here
 }
};


myInfo.arrowFunction();
myInfo.normalFunction();
```

۴- استفاده از کلمه کلید New :

توابع معمولی که با عبارت function ایجاد می‌شوند، قابل ساخت(construct) و فراخوانی هستند. پس می‌توان آن‌ها را با استفاده از کلمه کلیدی new فراخوانی کرد. و می توان نمونه ای از آن را کپی کرد اینستنس آبجکت از آن داشت .

اما توابع arrow فقط قابل فراخوانی هستند و قابلیت ساخته شدن ندارند، یعنی این توابع هرگز نمی‌توانند به عنوان توابع سازنده(constructor) مورد استفاده قرار بگیرند. پس نمی‌توان آن‌ها را با کلمه کلیدی new فراخوانی کرد.

```
let add = (x, y) => console.log(x + y);
new add(2,3);  // TypeError: add is not a constructor
```

۵- استفاده از پارامتر ها با نام های تکراری :

در حالت معمول این موضوع مشکلی پیش نمیاره چون ورودی ها شماره دارن و میتوان با اینستنس به آنها دسترسی پیدا کرد در هر دو سینتکس . اما در حالت 'use strict' یا همان حالت سخت گیرانه شما در normal function نباید نام های آرگومان های ورودی یکسان باشند ؛ اما در arrow function شما در حالت سختگیرانه و غیر آن هر دو غیر قابل استفاده و نا معتبر می باشند .

```
'use strict';
function add(x, x){}
// SyntaxError: duplicate formal argument  x



(x, x) => {}
// SyntaxError: duplicate argument names not allowed in this context
```

۶- مرتبه اجرا یا function hoisting :

در normal function شما از قابلیت اجرا قبل از ساختن یا همان hoisting میتوانید استفاده کنید ؛ اما در arrow function شما از این امکان محدود می شوید . به این دلیل که در جاوا اسکریپت رفرنس یک فانکشن و متغیر متفاوت می باشد .

```
// Hoisting

normalFunction()

function normalFunc() {
    return "Normal Function"
}
// "Normal Function"


arrowFunction ()
const arrowFunction  = () => {
    return "Arrow Function"
}
// ReferenceError: Cannot access 'arrowFunction ' before initialization
```

۷- استفاده از متد در کلاس کامپوننت :

به صورت دیفالت شما در normal function به متد ها جاوا اسکریپت دسترسی دارید که در بالا به ان اشاره کرده بودم . اما این ها کجا استفاده دارند ؟

```
class FullName {
    constructor(name) {
        this.name = name;
    }
    result() {
        console.log(this.name)
    }
}
let name = new FullName("Mohsen")
console.log(name)
// FullName {name: "Mohsen"}



setTimeout(name.result, 2000)
// after 1 second logs ""


// But if you bind this

setTimeout(name.result.bind(name), 2000)
// Mohsen


//In arrow function, you don't have to bind with conte
xt.


class FullName {
    constructor(name) {
        this.name = name;
    }
    result = () => {
        console.log(this.name)
    }
}
let name = new FullName("Mohsen")
setTimeout(name.result, 2000) // Mohsen
```

سخن پایانی :
اما سوالی که پیش میاد این است که چه زمانی از arrow function ها استفاده کنیم ؟ به صورت خلاصه و عامیانه هر زمان که به موارد بالا نیازی نداشتید . (در مدل های ابجکتی مثلا کلاس ها) برای مثال شما اگر تابعی دارید که نیازی به مباحث پیشرفته و کلی نداره و کار های کوچکی انجام میدهند بهتر است از این توابع استفاده کرد .

منبع : [GeeksForGeeks normal vs arrow function]