

فانکشن (Functions)

@alithecodeguy

گاهی اوقات نیاز داریم تعامل مشخصی را در بخش‌های مختلفی از اسکریپت انجام دهیم.

برای مثال ، می‌خواهیم که در هر بار لاگین کاربر و لاگ اوت وی ، پیغامی به وی نمایش داده شود.

فانکشن‌ها ، بلاک‌های اصلی یک برنامه هستند و اجازه می‌دهند هر بار که نیاز داریم کد را فراخوانی کنیم بدون اینکه نیاز داشته باشیم آن را بازنویسی کنیم.

تا کنون با تعدادی از فانکشن‌های داخلی جاوا اسکریپت از جمله alert ، prompt و confirm آشنا شده ایم ولی ما می‌توانیم هر تعداد فانکشن که می‌خواهیم به سلیقه خودمان بنویسیم.

تعریف فانکشن (Function Declaration)

برای ساخت یک فانکشن ، می‌توانیم آن را تعریف کنیم. مثال :

```
function showMessage() {  
  alert('Hello everyone!');  
}
```

ابتدا کلمه function ذکر می‌شود ، سپس نامی که برای فانکشن انتخاب کردیم را می‌نویسیم ، در ادامه لیستی از پارامترها که بوسیله کما از هم جدا شده اند داخل پرانتز نوشته می‌شود و در آخر بدنه فانکشن بین آکولاد نوشته می‌شود.

```
function name(parameters) {  
  ...body...  
}
```

فانکشن جدید ما توسط نام آن می‌تواند فراخوانی شود ، برای مثال :

```
function showMessage() {  
  alert('Hello everyone!');  
}
```

```
showMessage();  
showMessage();
```

فراخوانی فانکشن باعث اجرا شدن بدنه آن می‌شود. در مثال بالا پیغام Hello everyone! را دو بار مشاهده خواهیم کرد.

مثال فوق یکی از اصلی‌ترین هدف‌های فانکشن‌ها را نمایان می‌کند : جلوگیری از دوباره نویسی کد.

اگر نیاز داشته باشیم که متن پیغام یا نحوه نمایش آن را عوض کنیم ، فقط کافی است در بدنه فانکشن این کار را انجام دهیم.

@alithecodeguy

متغیرهای محلی (local variables)

به متغیری که داخل یک فانکشن تعریف می‌شود ، فقط از داخل همان فانکشن می‌توان دسترسی داشت . برای مثال :

```
function showMessage() {  
  let message = "Hello, I'm JavaScript!"; // local variable  
  alert( message );  
}  
  
showMessage(); // Hello, I'm JavaScript!  
alert( message ); // <-- Error! The variable is local to the function
```

متغیرهای بیرونی

یک فانکشن می‌تواند به متغیرهای بیرونی خودش نیز دسترسی داشته باشد . مثال :

```
let userName = 'John';  
  
function showMessage() {  
  let message = 'Hello, ' + userName;  
  alert(message);  
}  
  
showMessage(); // Hello, John
```

یک فانکشن علاوه بر دسترسی کامل به متغیرهای بیرونی ، آنها را می‌تواند تغییر نیز بدهد . برای مثال :

```
let userName = 'John';  
  
function showMessage() {  
  userName = "Bob"; // (1) changed the outer variable  
  let message = 'Hello, ' + userName;  
  alert(message);  
}  
  
alert( userName ); // John before the function call  
showMessage();  
alert( userName ); // Bob, the value was modified by the function
```

متغیرهای بیرونی تنها در صورتی استفاده می‌شوند که متغیر محلی به همان نام نداشته باشیم .

اگر متغیر محلی هم‌نام با متغیر بیرونی داشته باشیم ، متغیر محلی روی متغیر بیرونی سایه می‌اندازد (shadows) . در مثال زیر ، فانکشن از متغیر محلی userName استفاده کرده و با متغیر بیرونی userName کاری ندارد :

```
let userName = 'John';  
  
function showMessage() {  
  let userName = "Bob"; // declare a local variable  
  let message = 'Hello, ' + userName; // Bob
```

```
alert(message);
}
// the function will create and use its own userName
showMessage();
alert( userName ); // John, unchanged, the function did not access the outer variable
```

متغیرهای Global

متغیرهایی که خارج از هر فانکشنی نوشته می‌شوند، مانند `userName` در مثال فوق، متغیرهای Global نامیده می‌شوند. متغیرهای `variable` در همه فانکشن‌ها قابل دسترسی اند مگر اینکه متغیر محلی به همان نام در آن فانکشن تعریف شده باشد. پیشنهاد می‌شود که حتی المقدور از متغیرهای Global استفاده نکنید. کدهای مدرن هیچ یا تعداد اندک متغیر Global دارند. اکثر متغیرها درون فانکشن‌های متناظر خودشان جای می‌گیرند. اگرچه گاهی اوقات ذخیره داده‌ها به صورت Global ممکن است مفید واقع شود.

پارامترها

بوسیله پارامترها، مقادیر مختلفی را می‌توانیم به یک فانکشن ارسال کنیم. (به آنها آرگومان نیز می‌گویند)

در مثال زیر، فانکشن دو پارامتر دارد: `text` و `from`

```
function showMessage(from, text) { // arguments: from, text
  alert(from + ': ' + text);
}
```

```
showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
```

```
showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

هرگاه فانکشن بالا در خط‌های `*` و `**` فراخوانی شود، مقادیر ارسالی در متغیرهای محلی `from` و `text` کپی می‌شوند سپس فانکشن از آنها استفاده می‌کند.

یک مثال دیگر: متغیری به نام `from` داریم و آن را به یک فانکشن ارسال می‌کنیم. توجه داشته باشید که اگرچه فانکشن مقدار `from` را عوض می‌کند ولی این تغییر خارج از فانکشن صورت نمی‌پذیرد چرا که فانکشن همیشه یک کپی از مقدار ارسالی را دریافت می‌کند:

```
function showMessage(from, text) {
  from = '*' + from + '*'; // make "from" look nicer
  alert( from + ': ' + text );
}

let from = "Ann";
showMessage(from, "Hello"); // *Ann*: Hello
// the value of "from" is the same, the function modified a local copy
alert( from ); // Ann
```

مقادیر پیش فرض

اگر یک پارامتر تعریف شود ولی مقداری برای آن ارسال نگردد ، مقدار آن undefined می شود.

برای مثال ، در مثال قبلی می توانستیم فانکشن را به این صورت نیز فراخوانی کنیم :

```
showMessage("Ann");
```

فراخوانی به این شکل باعث بروز خطا نمی شود ، فقط خروجی آن به شکل "Ann : undefined" می شود. چون مقداری برای پارامتر text ارسال نشده است پس مقدار آن undefined می شود.

اگر بخواهیم در چنین حالت هایی خودمان یک مقدار پیش فرض تعریف کنیم ، می توانیم به روش زیر عمل کنیم :

```
function showMessage(from, text = "no text given") {
```

```
    alert( from + ": " + text );
```

```
}
```

```
showMessage("Ann"); // Ann: no text given
```

حال اگر مقداری برای پارامتر text ارسال نشود ، مقدار آن "not text given" می شود.

در این مثال ، مقدار پیش فرض "not text given" می شود ولی می تواند عبارات پیچیده تر نیز باشد که تنها در صورتی که مقداری ارسال نشود مقدار آن محاسبه می شود و به عنوان مقدار پارامتر ارسال می شود. پس کد زیر صحیح است :

```
function showMessage(from, text = anotherFunction()) {
```

```
    // anotherFunction() only executed if no text given
```

```
    // its result becomes the value of text
```

```
}
```

@alithecodeguy

محاسبه پارامترهای پیش فرض

در جاوا اسکریپت ، هر بار که فانکشن را فراخوانی می کنید ، مقدار پیش فرض آن محاسبه شده و در صورتی که مقداری برای پارامتر ارسال نکرده باشید ، به عنوان مقدار آن استفاده می شود.

در مثال فوق ، هر بار که showMessage() را بدون مقدار متناظر برای text فراخوانی می کنید ، فانکشن anotherFunction() فراخوانی و محاسبه می شود.

روش های دیگر انتخاب مقدار پیش فرض

^۱گاهی اوقات می خواهیم مقادیر پیش فرض را خارج از امضای فانکشن (پرانتر خط اول) بلکه در بدنه آن و در زمان اجرا انتخاب کنیم.

برای بررسی اینکه مقدار پارامتر ارسال شده است یا خیر می توانیم مقدار آن را با undefined چک کنیم:

```
function showMessage(text) {
```

```
    if (text === undefined) text = 'empty message';
```

```
    alert(text);
```

```
}
```

```
showMessage(); // empty message
```

@alithecodeguy

یا می‌توانیم از عملگر `||` استفاده کنیم :

```
// if text parameter is omitted or "" is passed, set it to 'empty'
```

```
function showMessage(text) {  
  text = text || 'empty';  
  ...  
}
```

@alithecodeguy

جاوا اسکریپت مدرن از عملگر `??` پشتیبانی می‌کند که راه مناسبتری برای چک کردن `falsy` بودن مقادیر ، فقط در صورت `null` و `undefined` بودن است . مثال :

```
// if there's no "count" parameter, show "unknown"
```

```
function showCount(count) {  
  alert(count ?? "unknown");  
}  
  
showCount(0); // 0  
showCount(null); // unknown  
showCount(); // unknown
```

مقدار بازگشتی

یک فانکشن می‌تواند مقداری را به عنوان نتیجه ، به کدی که آن را فراخوانی کرده ، بازگرداند . ساده‌ترین مثال برای آن ، فانکشنی است که جمع دو عدد را برمی‌گرداند :

```
function sum(a, b) {  
  return a + b;  
}  
  
let result = sum(1, 2);  
alert(result); // 3
```

کلمه `return` در هر جای فانکشن می‌تواند نوشته شود . وقتی که اجرای کد به آن برسد ، فانکشن متوقف شده و مقدار مورد نظر به عنوان نتیجه به تابع فراخواننده ، بازگردانده می‌شود .

در یک فانکشن ، ممکن است بیشتر از یک `return` داشته باشیم . برای مثال :

```
function checkAge(age) {  
  if (age >= 18) {  
    return true;  
  } else {  
    return confirm("Do you have permission from your parents?");  
  }  
}  
  
let age = prompt("How old are you?", 18);
```

@alithecodeguy

```
if ( checkAge(age) ) {  
    alert( 'Access granted' );  
} else {  
    alert( 'Access denied' );  
}
```

می‌توانیم `return` را بدون مقدار به کار ببریم. این عمل باعث خروج فوری از فانکشن می‌شود. مثال :

```
function showMovie(age) {  
    if ( !checkAge(age) ) {  
        return;  
    }  
    alert( "Showing you the movie" ); /*  
    // ...  
}
```

@alithecodeguy

در مثال فوق ، اگر `checkAge(age)` مقدار `false` برگرداند ، `showMovie` به `alert` نخواهد رسید .

یک فانکشن با `return` خالی یا بدون مقدار `return` مقدار `undefined` را برمی‌گرداند .

اگر فانکشن مقدار بازگشتی نداشته باشد ، مانند این است که مقدار `undefined` را برمی‌گرداند .

```
function doNothing() { /* empty */ }  
alert( doNothing() === undefined ); // true
```

یک `return` خالی نیز مانند `return undefined` رفتار می‌کند .

```
function doNothing() {  
    return;  
}  
alert( doNothing() === undefined ); // true
```

هرگز بین `return` و مقدار بازگشتی ، فاصله `newLine` وارد نکنید .

اگر مقدار بازگشتی طولانی باشد ، آدم وسوسه می‌شود که مقدار بازگشتی را در یک خط جدید بنویسد . مثال :

```
return  
(some + long + expression + or + whatever * f(a) + f(b))
```

مثال فوق کار نخواهد کرد چرا که جاوا اسکریپت در انتهای `return` علامت ; را تصور خواهد کرد . یعنی در واقع به شکل زیر با آن برخورد خواهد کرد :

```
return;  
(some + long + expression + or + whatever * f(a) + f(b))
```

پس عملاً `return` خالی خواهد بود .

اگر می خواهیم مقداری که بازگردانیم را در چند خط بنویسیم ، باید شروع آن در خطی باشد که **return** در آن قرار دارد یا حداقل از پرانتز استفاده کنیم و پرانتز باز را در جلوی **return** بنویسیم. مثال :

```
return (  
    some + long + expression  
    + or +  
    whatever * f(a) + f(b)  
)
```

در این حالت کد به درستی اجرا خواهد شد .

نام گذاری فانکشن

فانکشن ها ، عملکردهای برنامه هستند بنابراین نام آن معمولا " فعل " هستند . نام آنها باید خلاصه و دقیق بوده و کاری که انجام می دهند را تا حد ممکن توضیح دهد . این باعث می شود کسی که کد شما را می خواند ، سریع تر متوجه آن شود .
به طور معمول ، نام گذاری فانکشن ها را با انتخاب پیشوند به صورت فعل شروع می کنند که کاری که آن فانکشن انجام می دهد را توضیح می دهد . نام گذاری ها باید طبق قوانین به توافق رسیده بین اعضای تیم ، صورت پذیرد .
برای مثال تابعی که نام آن با **show** شروع می شود ، معمولا چیزی را نمایش می دهد .
مثال های دیگر از نام گذاری فانکشن ها :

- **get** : مقداری را برمی گرداند .
- **calc** : چیزی را محاسبه می کند .
- **create** : چیزی را می سازد .
- **check** : چیزی را چک می کند .

مثال :

```
showMessage(..) // shows a message  
getAge(..)      // returns the age (gets it somehow)  
calcSum(..)     // calculates a sum and returns the result  
createForm(..)  // creates a form (and usually returns it)  
checkPermission(..) // checks a permission, returns true/false
```

استفاده پیشوند ها به این شکل ، باعث می شود که در یک نگاه متوجه شویم که فانکشن چه کاری انجام می دهد .

یک فانکشن = یک عملکرد

هر فانکشن باید دقیقا کاری که اسم آن می گوید انجام نه بیشتر.

دو عملکرد متفاوت ، دو فانکشن متفاوت نیاز دارند ، حتی اگر هر دو با هم فراخوانی شوند . در این حالت می توانیم فانکشن سومی بسازیم که این دو فانکشن را با یکدیگر فراخوانی کند .

مثال هایی که غلط هستند :

- **getAge** : اگر سن را نمایش دهد غلط است چرا که فقط باید مقدار سن را برگرداند .

• **createForm** : اگر فرمی را اصلاح کرده و یا مقداری به آن اضافه کند غلط است چرا که فقط باید فرم جدید ساخته و آن را برگرداند .

• **checkPermission** : اگر پیغام نمایش دهد غلط است چرا که فقط باید مجوز دسترسی را چک کرده و مقداری را برگرداند .

در مثال های فوق ، معنای پیشوند ها ، همان معنای لغوی آنها در نظر گرفته شده است . شما و هم تیمی هایتان می توانید هر لغت و هر معنی متنناظری برای آنها را انتخاب کنید ولی معولا نباید تفاوت زیادی داشته باشند . در هر صورت شما باید فهم درستی از پیشوند و فانکشنی که آن را استفاده می کند ، داشته باشید .

نام های خیلی خیلی کوتاه

فانکشن ها به ندرت از اسامی خیلی خیلی کوتاه استفاده می کنند .

برای مثال ، **jQuery** فانکشن را با علامت **\$** مشخص می کند . کتابخانه **Lodash** نیز نام فانکشن اصلی خود را **_** گذاشته است .

این موارد استثنا هستند . به طور کلی اسامی فانکشن ها باید دقیق و گویا باشد .

فانکشن == کامنت

فانکشن باید کوتاه بوده و دقیقا یک کار انجام دهد . اگر فانکشن نوشته شده طولانی است ، شاید بهتر باشد به چند فانکشن کوچکتر تبدیل شود . گاهی اوقات پیروی از این قانون آسان نیست ولی به طور حتم ، بسیار خوب است .

یک فانکشن خوب ، خودش یک کامنت خوب است !

برای مثال دو فانکشن زیر را مقایسه کنید . هر دو اعداد اول تا **n** را نمایش می دهند .

مثال اول از **label** استفاده می کند :

```
function showPrimes(n) {
  nextPrime: for (let i = 2; i < n; i++) {
    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }
    alert(i); // a prime
  }
}
```

مثال دوم از فانکشن ثانویه به نام **isPrime(n)** برای تست اینکه عدد مورد نظر اول است یا خیر ، استفاده می کند :

```
function showPrimes(n) {
  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;
    alert(i); // a prime
  }
}
```

```
function isPrime(n) {
  for (let i = 2; i < n; i++) {
```



```
if ( n % i == 0) return false;
}
return true;
}
```

فهم مثال دوم آسان تر است. اینطور نیست؟ به جای یک تکه کد، نام فانکشن `isPrime` را می بینیم. بعضی افراد به این گونه فانکشن ها، `self-describing` می گویند. بنابراین حتی اگر نخواهیم از تکه کدی دوباره استفاده کنیم، می توانیم آن را به شکل فانکشن بنویسیم. فانکشن ها، ساختار کد را تشکیل می دهند و خوانایی آن را ساده تر می کنند.

@alithecodeguy

خلاصه

صورت کلی یک فانکشن به شکل زیر است :

```
function name(parameters, delimited, by, comma) {
  /* code */
}
```

- مقادیری که به شکل پارامتر به فانکشن ارسال می شود، به شکل یک کپی در متغیرهای محلی آن ذخیره می شود.
- یک فانکشن می تواند به متغیرهای بیرونی دسترسی داشته باشد ولی کدهای خارج از فانکشن نمی توانند به متغیرهای محلی دسترسی داشته باشند.
- یک فانکشن می تواند مقداری را برگرداند. اگر مقداری برگرداند، مقداری بازگشتی `undefined` می شود.
- برای خوانایی و فهم آسان کد، پیشنهاد می شود به طور کلی به حای استفاده از متغیرهای بیرونی، از متغیرهای محلی استفاده کنید.
- فهم تابعی که پارامتر می گیرد و مقداری را بر می گرداند، آسان تر از فانکشنی است که پارامتر نمی گیرد و از متغیرهای بیرونی استفاده می کند.
- نام یک فانکشن باید گویای عملکرد آن باشد.
- فانکشن یک عملکرد است، پس نام آن معمولاً "فعل" است.
- نام های معمولی برای فانکشن ها وجود دارد از جمله `create`، `show`، `get` و ...
- فانکشن ها، بلاک های اصلی یک اسکریپت هستند. در این آموزش ما مفاهیم پایه ای آن را توضیح دادیم ولی این تازه ابتدای کار است! در طول آموزش بارها به سراغ فانکشن ها آمده و ویژگی های پیشرفته تری از آن را بررسی خواهیم کرد.

@alithecodeguy