Function Expressions

در جاوا اسکریپت ، فانکشن یک ساختار جادویی نیست ولی نوع خاصی از داده و مقدار محسوب می‌شود.

در آموزش قبل ، به شکل زیر یک فانکشن را ایجاد کردیم :

```
function sayHi() {
  alert( "Hello" );
}
```

راه دیگری نیز برای ایجاد یک فانکشن نیز وجود دارد که به آن function expression می‌گویند و به شکل زیر نوشته می‌شود:

```
let sayHi = function() {
  alert( "Hello" );
};
```

در اینجا ، یک فانکشن ، ایجاد شده و صراحتا به یک متغیر نسبت داده شده است مانند هر متغیر دیگری. مهم نیست که تابع چگونه تعریف شود ، با آن به شکل یک مقدار برخورد می‌شود که در متغیر sayHi ذخیره می‌گردد.

حتی می‌توانیم آن متغیر را نمایش دهیم :

```
function sayHi() {
  alert( "Hello" );
}
alert( sayHi ); // shows the function code
```

توجه داشته باشید که خط آخر ، فانکشن را اجرا نمی‌کند چرا که بعد از sayHi پرانتز گذاشته نشده است. زبانهایی وجود دارند که در آنها ، نوشتن اسم فانکشن باعث اجرای آن فانکشن می‌شود ولی جاوا اسکریپت اینگونه نیست.

در جاوا اسکریپت ، یک فانکشن یک مقدار است پس به شکل یک مقدار می‌توانیم با آن رفتار کنیم. مثال فوق ، کد فانکشن را به شکل یک رشته نمایش می‌دهد.

اگرچه فانکشن یک مقدار مخصوص محسوب می‌شود ولی همچنان می‌توانیم آن را به شکل sayHi() فراخوانی کنیم.

می‌توانیم یک فانکشن را درون متغیر دیگری کپی کنیم :

```
function sayHi() {   // (1) create
  alert( "Hello" );
}
let func = sayHi;    // (2) copy
```

```
func(); // Hello     // (3) run the copy (it works)!
sayHi(); // Hello    //     this still works too (why wouldn't it)
```

در مثال بالا ، اتفاقات زیر رخ داده است :

- در قسمت 1 فانکشن ، تعریف شده و درون متغیر sayHi قرار گرفته است .
- در قسمت 2 ، مقدار sayHi درون متغیر func ریخته می‌شود . توجه داشته باشید که هیچ پرانتزی بعد از sayHi قرار نگرفته است . اگر پرانتز استفاده می کردیم ، فانکشن ، اجرا شده و نتیجه آن درون متغیر func ریخته می‌شد نه خود فانکشن .
- حال فانکشن را می توانیم بوسیله هر دو متغیر به شکل sayHi() یا func() تعریف کنیم .

توجه داشته باشید که فانکشن را به شکل زیر نیز می‌توانستیم بنویسیم :

```
let sayHi = function() {
  alert( "Hello" );
};
let func = sayHi;
// ...
```


در این صورت نیز همه چی به شکل معمول کار خواهد کرد .

ممکن است برایتان سوال پیش آمده باشد که چرا در انتهای function expression سیمی کالن قرار می گیرد ولی در انتهای function declaration قرار نمی گیرد؟

```
function sayHi() {
  // ...
}
let sayHi = function() {
  // ...
};
```


جواب آن ساده است :

- در انتهای بلاک های کد و ساختارهای گرامری که به شکل if{...} ، for(){} ، function f(){} و . . . نوشته می شوند به سیمی کالن نیازی نیست .
- function expression به شکل یک عبارت نوشته می شود : let sayHi = ...; و به عنوان مقدار در نظر گرفته می‌شود و یک بلاک کد نیست بلکه یک انتساب است و گذاشتن سیمی کالن در انتهای عبارات توصیه می شود . بنابراین سیمی کالن انتهای آن ، نه به خاطر فانکشن ، بلکه به خاطر عبارت است .

---

callback functions

در ادامه ، مثال های بیشتری از برخورد با فانکشن به شکل مقدار و استفاده از function expression را بررسی می کنیم .

فانکشنی به شکل ask(question , yes , no) با سه پارامتر نوشته ایم :

- question : متن سوال
- yes : فانکشنی که در صورت صحیح بودن جواب ، اجرا خواهد شد .
- no :فانکشنی که در صورت غلط بودن جواب ، اجرا خواهد شد .

فانکشن باید سوال question را پرسیده و با توجه به جواب ، یکی از فانکشن های yes() یا no() را اجرا کند :

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
function showOk() {
  alert( "You agreed." );
}
function showCancel() {
  alert( "You canceled the execution." );
}
// usage: functions showOk, showCancel are passed as arguments to ask
ask("Do you agree?", showOk, showCancel);
```


در عمل ، چنین تابع هایی ، بسیار کارا هستند .

آرگومان های showOk و showCancel فانکشن های callback خوانده می‌شوند .

مفهوم آن به این شکل است که فانکشنی را به فانکشن دیگر ارسال کرده تا فانکشن ارسالی با توجه به شرایط خاصی اجرا شود .

در مثال فوق ، در صورتی که جواب سوال صحیح بود showOk اجرا می‌شد و در صورت غلط بودن showCancel اجرا می‌شد .

با استفاده از function expression می توانستیم مثال فوق را به شکل کوتاه تری نیز بنویسیم :

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}
```

```
ask(
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```


در این مثال ، فانکشن ها مستقیما درون فراخوانی ask(...) تعریف شده اند . این فانکشن ها نامی ندارد بنابراین به آنها anonymous می گویند . چنین فانکشن های خارج از ask در دسترس نیستند چرا که به متغیری منتسب نشده اند و این دقیقا چیزی است که در این مثال می‌خواستیم . استفاده از چنین فانکشن هایی بسیار طبیعی است چرا که در بطن خود جاوا اسکریپت قرار دارند .

در حقیقت ، فانکشن ، مقداری است که عملکردی را توصیف می کند .

مقادیر معمولی مانند اعداد و رشته ها ، داده ها را توصیف می کنند . فانکشن یک عملکرد را توصیف می کند . ما می‌توانیم در کنار متغیرها ، فانکشن ها را نیز ارسال کنیم تا در صورت لزوم از آنها استفاده شود .

مقایسه function declaration در مقابل function expression

در این بخش در مورد تفاوت اساسی function declaration و function expression صحبت خواهیم کرد .

ابتدا گرامر و نحوه نوشتن آنها را بررسی می کنیم :

- function declaration : فانکشن به صورت یک بخش مجزا و در بدنه اصلی کد نوشته می شود .

```
// Function Declaration
function sum(a, b) {
  return a + b;
}
```

- function expression : فانکشن درون یک عبارت یا ساختار دیگر ساخته می‌شود . در مثال زیر ، فانکشن در سمت راست عملگر انتساب ساخته می‌شود :

```
// Function Expression
let sum = function(a, b) {
  return a + b;
};
```

تفاوت نامحسوس دیگر زمانی است که فانکشن توسط انجین جاوا اسکریپت ساخته می‌شود.

یک function expression هنگامی ساخته می‌شود که اجرای برنامه به آن برسد و فقط از همان لحظه به بعد می‌توان از آن استفاده کرد.

در مثال فوق ، هنگامی که اجرای برنامه به قسمت راست عملگر انتساب برسد ، فانکشن ، ساخته شده و از آن لحظه به بعد قابل استفاده است.

یک function declaration حتی قبل از تعریف آن نیز قابل استفاده است.

برای مثال ، یک global function declaration فارغ از جایی که تعریف شده است ، در همه جای اسکریپت قابل استفاده است. این خاصیت بواسطه معماری و الگوریتم داخلی جاوا اسکریپت صورت می‌پذیرد.

هنگامی که جاوا اسکریپت آماده می‌شود تا اسکریپتی را اجرا کند ، ابتدا دنبال global function declaration می‌گردد تا آنها را بسازد. می‌توانید این مرحله را ، مرحله "پیش از آغاز" نیز تصور کنید. سپس بعد از اینکه همه function declaration ها ساخته شد ، کد اصلی اجرا می شود. بنابراین ، کد اصلی به فانکشن ها دسترسی دارد. برای مثال :

```
sayHi("John"); // Hello, John
function sayHi(name) {
  alert( `Hello, ${name}` );
}
```


فانکشن sayHi ، پیش از آغاز اجرای اسکریپت ساخته شده و در طول اجرای اسکریپت در همه جای آن قابل استفاده است. در مثال فوق اگر به جای function declaration از function expression استفاده می‌کردیم ، کد ما با خطا مواجه می‌شد :

```
sayHi("John"); // error!
let sayHi = function(name) {  // (*) no magic any more
  alert( `Hello, ${name}` );
};
```

خاصیت دیگر function declaration ، block scope آن است.

در حالت مدرن ، هنگامی که یک function declaration داخل یک بلاک کد است ، در همه جای آن قابل دسترس است ولی خارج از آن بلاک کد نمی‌توان از آن استفاده کرد.

فرض کنید نیاز داریم تا فانکشنی به نام welcome بسازیم که از متغیر age که در زمان اجرا مقدار دهی می‌شود ، استفاده می‌کند. سپس می‌خواهیم از این فانکشن در قسمت دیگری استفاده کنیم :

```
let age = prompt("What is your age?", 18);
// conditionally declare a function
if (age < 18) {
```

```
  function welcome() {
    alert("Hello!");
  }
} else {
  function welcome() {
    alert("Greetings!");
  }
}
// ...use it later
welcome(); // Error: welcome is not defined
```


خطای رخ داده در این مثال بابت این است که function declaration فقط در بلاکی که در آن قرار دارد ، در دسترس است.

حال به مثال زیر توجه کنید :

```
let age = 16; // take 16 as an example
if (age < 18) {
  welcome();               // \  (runs)
 function welcome() {      //  |
    alert("Hello!");       //  |  Function Declaration is available
  }                        //  |  everywhere in the block where it's declared
 welcome();                // /  (runs)
} else {
  function welcome() {
    alert("Greetings!");
  }
}
// Here we're out of curly braces,
// so we can not see Function Declarations made inside of them.
welcome(); // Error: welcome is not defined
```


در این مثال نیز با خطا مواجه می‌شویم چرا که نمی‌توان از فانکشن welcome خارج از بلاک if استفاده کرد.

راه حل مناسب برای مثال فوق این است که از function expression استفاده کنیم و فانکشن welcome را به متغیری خارج از بدنه if منتسب کنیم.

کد زیر به درستی اجرا خواهد شد :

```
let age = prompt("What is your age?", 18);
```

```
let welcome;
if (age < 18) {
  welcome = function() {
    alert("Hello!");
  };
} else {
  welcome = function() {
    alert("Greetings!");
  };
}
welcome(); // ok now
```


برای پیاده سازی ساده تر کد فوق ، از عملگر ? نیز می‌توانیم استفاده کنیم :

```
let age = prompt("What is your age?", 18);
let welcome = (age < 18) ?
  function() { alert("Hello!"); } :
  function() { alert("Greetings!"); };
welcome(); // ok now
```

چه زمانی از function declaration و چه زمانی از function expression استفاده کنیم؟

به عنوان یک قانون کلی ، روش پیش فرض ساخت یک فانکشن ، function declaration است . ساختن یک فانکشن به این روش ، آزادی بیشتری به ما می‌دهد چرا که می‌توانیم هر جایی از کد که می‌خواهیم آن را تعریف کرده و هرجایی که می‌خواهیم فراخوانی کنیم . همچنین این روش خوانایی کد را نیز افزایش می‌دهد . ولی در مواردی که function declaration پاسخگوی نیاز ما نیست (مانند مثال فوق) ، می‌توانیم از function expression استفاده کنیم .
