

حلقه ها : while و for (loops)

@alithecodeguy

ما معمولا نیاز داریم که عملی را تکرار کنیم. چاپ کردن آیتم های یک لیست یکی پس از دیگری یا انجام عملیات خاصی برای هر یک از اعداد یک تا ده. حلقه ها ، روش انجام و تکرار چندین باره یک کد هستند .

حلقه while

حلقه while به شکل زیر نوشته می شود:

```
while (condition) {  
    // code  
    // so-called "loop body"  
}
```

هر گاه مقدار condition ، truthy باشد ، کد موجود در بدنه حلقه اجرا می شود.

برای مثال ، حلقه زیر تا زمانی که i کوچکتر از 3 باشد ، i را نمایش می دهد :

```
let i = 0;  
while (i < 3) { // shows 0, then 1, then 2  
    alert(i);  
    i++;  
}
```

@alithecodeguy

به هر بار تکرار یک حلقه ، یک iteration می گویند . حلقه مثال بالا سه iteration دارد .

اگر در مثال بالا i++ را حذف می کردیم ، حلقه (از نظر منطقی) تا بینهایت ادامه پیدا می کرد. در عمل ، مرورگر از ادامه پیدا کردن چنین حلقه هایی جلوگیری می کند. در جاوا اسکریپت سمت سرور نیز ، می توانیم پروسس مربوطه را kill کنیم. نه تنها عبارات مقایسه ای ، بلکه هر متغیر و عبارتی می تواند به عنوان شرط حلقه در نظر گرفته شود. شرط مورد نظر ، محاسبه شده و توسط while به صورت بولین تبدیل می شود. برای مثال ، کوتاه شده عبارت while (i != 0) به این شکل است : while (i)

```
let i = 3;  
while (i) { // when i becomes 0, the condition becomes falsy, and the loop stops  
    alert(i);  
    i--;  
}
```

اگر بدنه حلقه فقط یک خط داشته باشد ، نیازی نیست که از آکولاد (curly braces) استفاده کنیم .

```
let i = 3;
while (i) alert(i--);
```

حلقه do...while

با استفاده از حلقه do...while شرط حلقه می تواند به انتهای حلقه منتقل شود و به شکل زیر نوشته می شود :

```
do {
    // loop body
} while (condition);
```

حلقه در ابتدا بدنه را اجرا می کند ، سپس شرط را چک می کند و اگر درست بود دوباره حلقه را تکرار می کند . برای مثال :

```
let i = 0;
do {
    alert(i);
    i++;
} while (i < 3);
```

@alithecodeguy

این نوع حلقه هنگامی استفاده می شود که بخواهیم قبل از چک کردن شرط حلقه ، بدنه حلقه را حداقل یک بار اجرا کنیم . معمولا استفاده از حلقه while به این روش ترجیح داده می شود .

حلقه for

حلقه for نسبت به روش های پیشین کمی پیچیده تر بوده ولی پرکاربردترین حلقه می باشد و به روش زیر نوشته می شود :

```
for (begin; condition; step) {
    // ... loop body ...
}
```

با مثالی به شرح این حلقه می پردازیم . حلقه زیر alert(i) را برای i از 0 تا 2 فراخوانی می کند .

```
for (let i = 0; i < 3; i++) { // shows 0, then 1, then 2
    alert(i);
}
```

@alithecodeguy

حال مثال فوق را جزء به جزء بررسی می‌کنیم :

begin	<code>i = 0</code>	Executes once upon entering the loop.
condition	<code>i < 3</code>	Checked before every loop iteration. If false, the loop stops.
body	<code>alert(i)</code>	Runs again and again while the condition is truthy.
step	<code>i++</code>	Executes after the body on each iteration.

الگوریتم کلی حلقه به شکل زیر است :

Run begin

→ (if condition → run body and run step)

→ (if condition → run body and run step)

→ (if condition → run body and run step)

→ ...

مرحله begin شروع شده و هر iteration بعد از چک کردن شرط ، صورت پذیرفته و در آخر هر کدام مرحله step اجرا می‌شود. اگر تازه با حلقه ها آشنا شده اید پیشنهاد می‌شود که به مثال فوق برگردید و مراحل آن را تک به تک روی کاغذ بنویسید .

در مثال ، دقیقاً این اتفاق صورت می‌پذیرد :

```
// for (let i = 0; i < 3; i++) alert(i)
```

```
// run begin
```

```
let i = 0
```

```
// if condition → run body and run step
```

```
if (i < 3) { alert(i); i++ }
```

```
// if condition → run body and run step
```

```
if (i < 3) { alert(i); i++ }
```

```
// if condition → run body and run step
```

```
if (i < 3) { alert(i); i++ }
```

```
// ...finish, because now i == 3
```

در مثال زیر متغیر `i` درون حلقه تعریف شده است . به این نوع تعریف inline declaration می‌گویند . چنین متغیرهایی فقط داخل بدنه حلقه قابل دسترس اند .

```
for (let i = 0; i < 3; i++) {
```

```
  alert(i); // 0, 1, 2
```

```
}
```

```
alert(i); // error, no such variable
```

به جای تعریف متغیر جدید ، می‌توانیم از متغیرهایی که قبلاً تعریف شده اند نیز استفاده کنیم :

```
let i = 0;
for (i = 0; i < 3; i++) { // use an existing variable
  alert(i); // 0, 1, 2
}
alert(i); // 3, visible, because declared outside of the loop
```

نادیده گرفتن بخش های مختلف

هر یک از بخش های حلقه را می توان نادیده گرفت. برای مثال ، اگر به انجام کار خاصی در شروع حلقه نیاز نداریم ، می توانیم بخش begin را نادیده بگیریم. مثال :

```
let i = 0; // we have i already declared and assigned
for (; i < 3; i++) { // no need for "begin"
  alert(i); // 0, 1, 2
}
```

همچنین می توانیم قسمت step را نیز حذف کنیم :

```
let i = 0;
for (; i < 3;) {
  alert(i++);
}
```

حلقه فوق مشابه حلقه while(i<3) شده است . در واقع می توانیم همه چیز را حذف کنیم و یک حلقه بینهایت بسازیم.

```
for (;;) {
  // repeats without limits
}
```

دقت کنید که دو علامت ; را نمی شود حذف کرد و در صورت حذف ، خطا خواهد داد.

شکستن و توقف حلقه (break)

در حالت عادی ، یک حلقه زمانی پایان می یابد که شرط آن falsy بشود. ولی می توانیم با استفاده از دستور مخصوص break هر زمان که بخواهیم از حلقه خارج شویم.

برای مثال ، حلقه زیر از کاربر سری ای از اعداد را می پرسد و در صورتی که کاربر هیچ عددی را وارد نکند حلقه پایان می یابد :

```
let sum = 0;
while (true) {
  let value = +prompt("Enter a number", "");
  if (!value) break; // (*)
  sum += value;
```

```
}
```

```
alert( 'Sum: ' + sum );
```

اگر کاربر چیزی وارد نکند یا کنسل کند ، کلمه **break** در خط * باعث توقف حلقه و خروج از آن می شود و برنامه را از خط بعد از حلقه ادامه می دهد و در واقع در مثال فوق ، **alert** اجرا می شود .

ترکیب حلقه بینهایت و کلمه **break** برای موقعیت هایی که نیاز داریم شرایط خاصی در میانه حلقه چک شود ، بسیار کارا است .

انجام iteration بعدی

کلمه **continue** نسخه سبک تری از **break** می باشد چرا که باعث توقف کامل حلقه نشده بلکه iteration جاری را خاتمه داده و در صورتی که شرط **truthy** باشد به سراغ iteration بعدی می رود .

در شرایطی که کارمان با iteration جاری تمام شده و می خواهیم به سراغ iteration بعدی برویم می توانیم از **continue** استفاده کنیم . مثال زیر از **continue** استفاده می کند تا فقط اعداد فرد را نمایش دهد .

```
for (let i = 0; i < 10; i++) {
```

```
    // if true, skip the remaining part of the body
```

```
    if (i % 2 == 0) continue;
```

```
    alert(i); // 1, then 3, 5, 7, 9
```

```
}
```

در مثال فوق ، هنگامی که **i** زوج است ، **continue** اجرای حلقه را به iteration بعدی منتقل می کند ، بنابراین فقط اعداد فرد نمایش داده می شود .

continue از تودرتو شدن جلوگیری می کند .

مثال فوق را به شکل زیر می شود نوشت :

```
for (let i = 0; i < 10; i++) {
```

```
    if (i % 2) {
```

```
        alert(i);
```

```
    }
```

```
}
```

از نظر منطقی ، عملکرد هر دو یکسان است . ولی مشکلی که در این حالت پیش می آید این است که بدنه **if** نوشته شده ممکن است بیشتر از یک خط باشد و همین باعث تودرتو شدن کد و کاهش خوانایی برنامه می شود .

break و **continue** را نمی توانید همراه با عملگر **?** به کار ببرید . به عنوان مثال ، کد زیر اشتباه است :

```
(i > 5) ? alert(i) : continue;
```

break و continue برای label

گاهی اوقات نیاز داریم که از چند حلقه تودرتو به صورت همزمان خارج شویم :

به عنوان مثال ، در مثال زیر از دو حلقه تودرتو و `i` و `j` استفاده می کنیم تا مقدار متناظر با مختصات `(0,0)` تا `(2,2)` را نمایش دهیم :

```
for (let i = 0; i < 3; i++) {  
  for (let j = 0; j < 3; j++) {  
    let input = prompt('Value at coords ({i},{j})', '');  
    // what if we want to exit from here to Done (below)?  
  }  
}  
alert('Done!');
```

به راهی نیاز داریم تا در صورت کنسل شدن `input` توسط کاربر ، پروسه متوقف شود .

استفاده از `break` به صورت عادی بعد از `input` فقط باعث خروج از حلقه داخلی می شود . در این مواقع می توانیم از `label` استفاده کنیم . `label` شناسه ای همراه با : است که قبل از حلقه استفاده می کنیم :

```
labelName: for (...) {  
  ...  
}
```

عبارت `break <labelName>` در حلقه زیر باعث خروج از `label` می شود :

```
outer: for (let i = 0; i < 3; i++) {  
  for (let j = 0; j < 3; j++) {  
    let input = prompt('Value at coords ({i},{j})', '');  
    // if an empty string or canceled, then break out of both loops  
    if (!input) break outer; // (*)  
    // do something with the value...  
  }  
}  
alert('Done!');
```

در مثال فوق عبارت `break outer` به سراغ `label` ای به نام `outer` رفته و از آن حلقه خارج می شود . بنابراین روند اجرای کد از * مستقیماً به `alert` می رود . همچنین می توانیم `label` را در خط جداگانه ای نیز بنویسیم :

```
outer:  
for (let i = 0; i < 3; i++) { ... }
```

`label` را همراه با `continue` نیز می توانستیم استفاده کنیم . در این حالت اجرای `iteration` جاری متوقف شده و به سراغ `iteration` بعدی حلقه متناظر با `label` می رود .

از `break` و `continue` فقط می شود درون حلقه استفاده کرد و `label` باید در بالای آنها قرار گرفته باشد .

نکته : در استفاده از `label` افراط نکنید و حتی المقدور از آن استفاده نکنید .

خلاصه :

در مورد سه نوع حلقه صحبت کردیم :

@alithecodeguy

- while : شرط حلقه قبل از هر iteration چک می شود.
- do...while : شرط حلقه بعد از هر iteration چک می شود.
- for(;;) : شرط حلقه قبل از هر iteration چک می شود و امکانات بیشتری در اختیار ما قرار می دهد.

برای ایجاد حلقه بی نهایت می توانید از while(true) استفاده کنید و می توانید بوسیله کلمه break از آن خارج شوید .
اگر کارمان با iteration جاری تمام شده و می خواهیم به iteration بعدی برویم می توانیم از کلمه continue استفاده کنیم .
از label می توانیم همراه با continue و break استفاده کنیم . label تنها راه خروج از حلقه های تو در تو بوسیله break یا continue است .