پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم.

عملگرهای پایه ای و ریاضیات

در مدرسه ، تعداد زیادی عملگر ریاضی را آموخته ایم. مواردی مانند : جمع (+ addition) ، ضرب (* multipilication) ، تفریق (- subtraction) و ...

در این فصل با عملگرهای ساده شروع می‌کنیم سپس به سراغ مفاهیم خاص جاوا اسکریپت می رویم که در ریاضیات مدرسه وجود ندارد.

@alithecodeguy

unary (تکی) ، binary (دوتایی) ، operand (عملوند)

قبل از ادامه ، به تعریف بعضی واژه ها بپردازیم :

operator (عملوند) : چیزی است که عملگر روی آن اجرا می‌شود. برای مثال در عبارت 2*5 دو عملوند وجود دارد. عملوند سمت چپ 5 و عملوند سمت راست 2 است. بعضی از افراد به جای واژه operand از argument هم استفاده می کنند.

unary (تکی) : یک اپراتور unary است اگر تنها یک عملوند داشته باشد. برای مثال عملگر منفی ساز – یک عملگر unary است :

```
let x = 1;
x = -x;
alert( x ); // -1, unary negation was applied
```

binary (دوتایی) : یک عملگر دوتایی است اگر دو عملوند داشته باشد. برای مثال عملگر تفریق – یک عملگر binary است :

```
let x = 1, y = 3;
alert( y - x ); // 2, binary minus subtracts values
```

در مثال های بالا عملگر – دو نوع کاربرد دارد ، یکی به عنوان منفی ساز و دیگری به عنوان تفریق.

ریاضیات

عملیات های ریاضی زیر توسط جاوا اسکریپت پشتیبانی می شود :

جمع (addition) : +

@alithecodeguy

تفریق (subtraction) : -

ضرب (multiplication) : *

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

تقسیم (division) : /

باقی مانده (remainder) : %

توان (exponentiation) : **

چهار عملگر اول مشخص است ولی دو مورد آخر نیاز به توضیحات بیشتری دارد .

باقی مانده (% remainder)

عملگر باقی‌مانده با % نمایش داده می‌شود و ربطی به درصد ندارد .

نتیجه عبارت a%b باقی مانده تقسیم صحیح a بر b می باشد .

برای مثال :

```
alert( 5 % 2 ); // 1, a remainder of 5 divided by 2
alert( 8 % 3 ); // 2, a remainder of 8 divided by 3
```

توان (** exponentiation)

عبارت a**b یعنی a به توان b

برای مثال :

```
alert( 2 ** 2 ); // 4  (2 multiplied by itself 2 times)
alert( 2 ** 3 ); // 8  (2 * 2 * 2, 3 times)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2, 4 times)
```

در به توان رساندن ، اعداد غیر صحیح نیز می‌توانیم به کار ببریم . به عنوان مثال جذر یعنی به توان رساندن یک عدد به عدد نیم :

```
alert( 4 ** (1/2) ); // 2 (power of 1/2 is the same as a square root)
alert( 8 ** (1/3) ); // 2 (power of 1/3 is the same as a cubic root)
```

تجمیع رشته ای با عملگر دوتایی +

@alithecodeguy

ویژگی ای که در ادامه خواهیم گفت ، خارج از ریاضیات مدرسه می باشد .

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

معمولا از عملگر + برای جمع اعداد استفاده می‌شود ولی در صورتی که این عملگر روی رشته ها اعمال شود ، آنها را با یکدیگر ترکیب می کند .

```
let s = "my" + "string";
alert(s); // mystring
```

توجه داشته باشید که اگر یکی از عملوندها رشته باشد ، عملوند دیگر نیز به رشته تبدیل می‌شود . برای مثال :

```
alert( '1' + 2 ); // "12"
alert( 2 + '1' ); // "21"
```

@alithecodeguy

بنابراین تفاوتی ندارد که عملوند اول رشته باشد یا دومی .

مثال :

```
alert(2 + 2 + '1' ); // "41" and not "221"
```

در این مثال عملگرها به ترتیب اجرا می‌شوند . عملگر + اول دو عدد را جمع می کند و عدد 4 را برمی گرداند سپس عملگر + بعدی رشته 1 را به آن اضافه می کند و در نهایت مقدار آن رشته 41 می شود .

عملگر دوتایی + تنها عملگری است که با رشته ها بدین صورت برخورد می کند .

باقی عملگرهای ریاضی تنها با اعداد کار می کنند و همیشه عملوندهای خود را به عدد تبدیل می کنند . مثال :

```
alert( 6 - '2' ); // 4, converts '2' to a number
alert( '6' / '2' ); // 3, converts both operands to numbers
```

تبدیل عددی ، عملگر تکی +

عملگر + به دو صورت وجود دارد : به صورت دوتایی که در مثال قبل دیدیم و به صورت تکی . عملگر دوتایی + که روی یک مقدار تنها اعمال می‌شود ، بر روی عددها کار خاصی انجام نمی‌دهد ولی اگر عملوند ، رشته باشد ، آن را تبدیل به عدد می کند . مثال :

```
// No effect on numbers
let x = 1;
alert( +x ); // 1
let y = -2;
alert( +y ); // -2
// Converts non-numbers
alert( +true ); // 1
alert( +"" );   // 0
```

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

در واقع عملگر تکی + همان کاری را انجام می‌دهد که فانکشن Number(...) انجام می‌دهد ، ولی کوتاهتر است .

نیاز به تبدیل رشته به عدد ، به ندرت پیش می آید . برای مثال ، inputfield ها معمولا رشته بر می‌گردانند . اگر بخواهیم انها را جمع کنیم چطور؟

عملگر دوتایی + آنها را به صورت رشته ای تجمیع می‌کند :

```
let apples = "2";
let oranges = "3";
// both values converted to numbers before the binary plus
alert( +apples + +oranges ); // 5
// the longer variant
// alert( Number(apples) + Number(oranges) ); // 5
```

از نقطه نظر ریاضی ، اینکه عملگر + چندین کار انجام می‌دهد ممکن است عجیب به نظر برسد ولی از دیدگاه برنامه نویسی ، این موضوع عجیب نیست . عملگر تکی + ابتدا روی عملوندها اعمال می‌شود و آنها را به عدد تبدیل می کند سپس عملگر دوتایی + عملیات جمع را انجام می‌دهد . چرا عملگر تکی ، قبل از عملگر دوتایی روی عملوندها اعمال می‌شود؟ همانطور که در ادامه خواهیم دید ، دلیل این امر ، اولویت بالاتر آن است .

@alithecodeguy

اولویت عملگرها

اگر عبارتی بیشتر از یک عملگر داشته باشد ، ترتیب اجرای آنها بر اساس اولویت آنها انجام می‌شود .

از دوران مدرسه به خاطر داریم که در عبارت 1 + 2 * 2 عملیات ضرب پیش از جمع انجام می‌شود و این دقیقا همان اولویت عملگرهاست . عملیات ضرب نسبت به جمع از اولویت بالاتری برخوردار است .

پرانتز اولویت عملگرها را تغییر می‌دهد بنابراین اگر از اولویت عملگری راضی نیستیم ، می توانیم از پرانتز استفاده کنیم تا اولویت آن را عوض کنیم . برای مثال :

```
(1 + 2) * 2
```

عملگرهای متفاوتی در جاوا اسکریپت وجود دارد . هر عملگر عدد اولویت مختص خودش را دارد . عدد بالاتر نشان دهنده اولویت بالاتر است . اگر اولویت یکسان باشد ، اولویت اجرا از چپ به راست است .

جدول اولویت به شرح ذیل می باشد :

(نیازی نیست که این جدول را حفظ کنید ولی توجه داشته باشید که عملگرهای تکی اولویت بالاتری نسبت به عملگرهای دوتایی دارند .)

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم.

علامت	نام انگلیسی	نام فارسی	اولویت
...	...	...	..
+	unary plus	مثبت	17
-	unary negation	منفی	17
**	exponentiation	توان	16
*	multiplication	ضرب	15
/	division	تقسیم	15
+	addition	جمع	13
-	subtraction	تفریق	13
...	...	...	...
=	assignment	انتساب	3
...	...	...	...

همانطور که میبینیم ، علامت تکی + اولویت 17 دارد در صورتی که علامت دوتایی + اولویت 13 دارد. به همین دلیل است که ابتدا علامت تکی + اعمال می‌شود.

@alithecodeguy

انتساب

توجه داشته باشید که انتساب نیز یک عملگر محسوب می شود و اولویت آن در جدول اولویت ، 3 است. به خاطر همین است که در عبارتی همچون :

```
x = 2 * 2 + 1
```

ابتدا محاسبات انجام شده سپس انتساب (=) صورت می گیرد و در x ذخیره می شود.

@alithecodeguy

عملگر انتساب یک مقدار برمیگرداند.

عملگر انتساب (=) رفتار جالبی دارد. اکثر عملگرهای جاوا اسکریپت یک مقدار برمی‌گردانند. این رفتار برای + و – منطقی است ولی عملگر = نیز همچین رفتاری دارد.

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

عبارت x = value مقدار value را داخل x ذخیره کرده سپس آن را بر می گرداند .

مثال :

```
let a = 1;
let b = 2;
let c = 3 - (a = b + 1);
alert( a ); // 3
alert( c ); // 0
```

@alithecodeguy

در مثال فوق ، نتیجه عبارت (a = b + 1) همان مقداری است که در a ذخیره می شود و در محاسبه باقی عبارت استفاده می شود .

جالب است ، نه؟ ما باید با نحوه کار آن آشنا باشیم چرا که گاهی اوقات در بعضی از کتابخانه ها به آن برخورد می کنیم .

اگرچه نباید کد را به این شکل نوشت چرا که خوانایی و وضوح کد را افزایش نمی دهد .

انتساب زنجیره ای

ویژگی جالب دیگر ، انتساب زنجیره ای است .

```
let a, b, c;
a = b = c = 2 + 2;
alert( a ); // 4
alert( b ); // 4
alert( c ); // 4
```

انتساب زنجیره ای از راست به چپ اجرا می‌شود . ابتدا مقدار 2 + 2 محاسبه می‌شود و در c ، سپس در b و در نهایت در a قرار می گیرد . در نهایت مقدار همه متغیرها یکسان می‌شود . برای خوانایی بیشتر ، پیشنهاد می‌شود ، کد در چندین خط نوشته شود :

```
c = 2 + 2;
b = c;
a = c;
```

نوشتن کد به این صورت راحت تر است مخصوصا هنگامی که نگاه سریع به کد می اندازیم .

@alithecodeguy

اصلاح درجا

معمولا نیاز داریم تا عملگری را روی متغیری اعمال کنیم سپس نتیجه را در همان متغیر ذخیره کنیم . برای مثال :

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

```
let n = 2;
n = n + 5;
n = n * 2;
```

با استفاده از += و *= می توان عبارت فوق را به شکل کوتاهتری نوشت :

```
let n = 2;
n += 5; // now n = 7 (same as n = n + 5)
n *= 2; // now n = 14 (same as n = n * 2)
alert( n ); // 14
```

عملگرهای کوتاه modify-and-assign برای همه عملگرهای ریاضی و بیتی وجود دارد . مانند /= ، -= و . . .

اینچنین عملگرهایی ، اولویت یکسان با انتساب عادی متناظر آن دارند بنابراین این انتساب ها بعد از محاسبه باقی عبارت انجام می شود .

```
let n = 2;
n *= 3 + 5;
alert( n ); // 16  (right part evaluated first, same as n *= 8)
```

@alithecodeguy

افزایش / کاهش (decrement/increment)

کاهش یا افزایش یک واحدی ، یکی از معمول ترین عملگرهای عددی در جاوا اسکریپت است . عملگرهای مخصوصی برای آن وجود دارد :

- افزایش : ++ متغیر را یک واحد افزایش می دهد .

```
let counter = 2;
counter++;        // works the same as counter = counter + 1, but is shorter
alert( counter ); // 3
```

- کاهش : -- متغیر را یک واحد کاهش می دهد .

```
let counter = 2;
counter--;        // works the same as counter = counter - 1, but is shorter
alert( counter ); // 1
```

نکته : عملگرهای ++ و -- فقط روی متغیرها کار می کند و استفاده از آن به شکل : 5++ منجر به خطا می شود .

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

عملگرهای ++ و — هم قبل از متغیر و هم بعد از متغیر می تواند استفاده شود ولی در هر کدام نحوه کار متفاوتی دارند .

- وقتی عملگر ++ بعد از متغیر می‌آید ، به آن حالت postfix میگویند :

```
counter++
```

- وقتی عملگر قبل از متغیر می آید ، به آن حالت prefix میگویند :

```
++counter
```

هر دو حالت کار یکسانی انجام می‌دهند و آن افزایش متغیر به اندازه 1 واحد است .

آیا این دوحالت با یکدیگر تفاوتی دارند؟ بله . ما هنگامی متوجه این موضوع خواهیم شد که از مقدار بازگشتی آنها استفاده کنیم .

می دانیم که همه عملگرها مقداری را بر می گردانند و ++ و — از این قضیه مستثنی نیستند . حالت prefix مقدار جدید متغیر و حالت postfix مقدار قدیم متغیر را برمی‌گرداند . برای مثال :

```
let counter = 1;
let a = ++counter; // (*)
alert(a); // 2
```

در خطی که با * علامت زده شده است ، مقدار counter که 1 است یک واحد افزایش داده شده و مقدار 2 برگردانده می شود و alert آن را نمایش می دهد .

مثال دیگر :

```
let counter = 1;
let a = counter++; // (*) changed ++counter to counter++
alert(a); // 1
```

در خطی که با * علامت زده شده است ، مقدار counter برگردانده شده سپس یک واحد افزایش پیدا می کند ، بنابراین alert مقدار 1 را نمایش می‌دهد .

به طور خلاصه می توانیم بگوییم :

- اگر از مقدار برگردانده شده توسط ++ و — استفاده نمی‌کنید ، فرقی ندارد که از کدام حالت استفاده کنید .

```
let counter = 0;
counter++;
++counter;
alert( counter ); // 2, the lines above did the same
```

@alithecodeguy

- اگر می خواهید متغیری را افزایش دهید و بلافاصله از آن استفاده کنید از حالت prefix استفاده نمایید :

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

```
let counter = 0;
alert( ++counter ); // 1
```

- اگر می خواهید متغیری را افزایش داده ولی از مقادیر پیش از افزایش آن استفاده نمایید از حالت postfix استفاده نمایید .

```
let counter = 0;
alert( counter++ ); // 0
```

عملگر ++ و -- در بین دیگر عملگرها

عملگر ++ و - - می تواند درون عبارت های مختلف استفاده شود . اولویت اجرای آنها از اکثر عملگرهای ریاضی بالاتر است .

برای مثال دو قطعه کد زیر را مقایسه کنید :

```
let counter = 1;
alert( 2 * ++counter ); // 4
```

@alithecodeguy

```
let counter = 1;
alert( 2 * counter++ ); // 2, because counter++ returns the "old" value
```

از نظر تکنیکی دو قطعه کد بالا درست هستند ولی این نوع استفاده از عملگرها خوانایی برنامه را کاهش می دهد چرا که یک خط چندین کار انجام می دهد .

در هنگام مطالعه کد ، تفاوت دو قطعه کد بالا در یک نگاه قابل تشخیص نیست .

به خاطر مواردی که ذکر شد است بهتر است طوری کد نوشته شود که هر خط فقط یک کار انجام دهد :

```
let counter = 1;
alert( 2 * counter );
counter++;
```

عملگرهای بیتی (Bitwise)

عملگرهای بیتی با آرگومان ها به شکل اعداد صحیح ۳۲ بیتی رفتار می‌کنند و در مبنای دودویی با آنها رفتار می کنند . این عملگرها مختص جاوا اسکریپت نیستند و توسط اکثر زبان های برنامه نویسی پشتیبانی می‌شوند .

لیست عملگرهای بیتی :

- AND (&)
- OR (|)
- XOR (^)
- NOT (~)

@alithecodeguy

پیشاپیش از هرگونه حمایت شما از جمله subscribe درکانال یوتیوب و کانال تلگرام سپاسگزارم .

- LEFT SHIFT (<<)
- RIGHT SHIFT (>>)
- ZERO-FILL RIGHT SHIFT (>>>)

این عملگرها به ندرت استفاده می شوند . ما به این عملگرها نیازی نخواهیم داشت چرا که در برنامه نویسی وب کاربردی ندارند ولی در بعضی از حوزه ها مانند رمزنگاری بسیار پرکاربرد هستند . برای اطلاعات بیشتر در مورد این عملگرها می‌توانید داکیومنت آن را روی MDN مطالعه کنید .

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators

کاما (comma)

عملگر کاما (,) عملگری غیرعادی است که خیلی به ندرت استفاده می‌شود . گاهی اوقات از این عملگر برای کوتاه تر کردن عبارات استفاده می‌شود ، بنابراین باید از نحوه کار آن اطلاع داشته باشیم .

عملگر کاما این امکان را به ما می‌دهد چندین عبارت که با کاما از هم جدا شده اند را جداگانه محاسبه کنیم . همه آنها محاسبه می‌شود ولی فقط مقدار محاسبه شده پایانی برگردانده می‌شود . برای مثال :

```
let a = (1 + 2, 3 + 4);
alert( a ); // 7 (the result of 3 + 4)
```

در مثال فوق مقدار 1+2 محاسبه شده و دور انداخته می‌شود . سپس مقدار 3+4 محاسبه شده و به عنوان نتیجه بازگردانده می‌شود .

کاما اولویت بسیار پایینی دارد .

توجه داشته باشید که اولویت کاما حتی از = پایین تر است بنابراین هنگام استفاده از آن ، مهم است که از پرانتز استفاده کنیم . اگر در مثال بالا از کاما استفاده نکنیم مقدار a = 3,7 محاسبه شده و a = 3 شده و باقی عبارت نادیده گرفته می شود . در واقع به این شکل عمل می کند :

```
(a = 1 + 2), 3 + 4
```

چرا باید از عملگری استفاده کنیم که همه مقادیر به جز مقدار آخر را دور می‌اندازد؟

بعضی از برنامه نویسان از آنها در محاسبات پیچیده و محاسبه چندین مقدار در یک خط استفاده می‌کنند . برای مثال :

```
// three operations in one line
for (a = 1, b = 3, c = a * b; a < 10; a++) {...
}
```

@alithecodeguy

همچنین نکته هایی در بسیاری از فریم ورک های جاوا اسکریپت استفاده می‌شود به همین خاطر به آن اشاره کردیم ولی به طور کلی نوشتن چنین عبارت هایی خوانایی کد را کاهش می‌دهد بنابراین توصیه می‌شود تنها در صورت لزوم از آنها استفاده کنید .

@alithecodeguy