**عملگرها**

ما عملگرهای مختلفی را از زمان مدرسه به خاطر داریم. مانند جمع + ، تفریق - ، ضرب \* ، تقسیم و دیگر عملگرها.

در این بخش ما بر روی زوایایی از عملگرها متمرکز می‎شویم که در مدرسه نیاموخته‌ایم.

**اصطلاحات : unary (یونِری) – binary (باینری) – operand (اُپِرند)**

پیش از ادامه بیایید مفهوم این اصطلاحات را بفهمیم.

یک operand – همان چیزی است که عملگرها بر روی آنها اعمال می‌شوند. برای نمونه در ضرب 5 \* 2 دو operand داریم. برخی آن را آرگمان (argument) نیز می‌خوانند.

یک عملگر زمانی unary است که فقط یک operand داشته باشد. برای نمونه unary منفی کننده‌ی – که علامت یک عدد را برعکس می‌کند.

کد

یک عملگر زمانی binary است که دو operand داشته باشد. همان عملگر منفی کننده در شکل binary هم وجود دارد.

کد

در اصل در اینجا ما در مورد دو عملگر مجزا صحبت کردیم. اولی عملگر unary منفی کننده (که علامت عدد را برعکس می‌کرد) و دیگری عملگر تفریق binary (تفریق دو عدد از هم).

**تلفیق رشته‌ها، عملگر + binary**

معمولا از عملگر + برای جمع اعداد استفاده می‌شود، اما زمانیکه این عملگر به رشته‌‌‌ها اعمال شود، آنها را ادغام می‌کند.

کد

در نظر داشته باشید که اگر یکی از operand ها string باشد، دیگری نیز به string تبدیل می‌شود.

برای نمونه :

کد

همانطور که می‌بینید مهم نیست که operand ابتدا آمده باشد یا نه، قاعده ساده است: اگر هر کدام از operand ها string بود، دیگری نیز به string تبدیل می‌شود.

با این حال در نظر داشته باشید که عملیات از چپ به راست انجام می‌شود. اگر دو عدد در پیش از یک string آمده باشند، ابتدا آن دو عدد جمع می‌شوند و سپس به string تبدیل می‌شوند.

کد

تبدیل و الصاق رشته‌ها یکی از ویژگی‌های خاص عملگر + است. دیگر عملگرهای ریاضیاتی فقط با اعداد کار می‌کنند و همیشه operand خود را به number تبدیل می‌کنند.

برای نمونه :

کد

**تبدیل به عدد ، عملگر unary +**

علامت جمع + به دو شکل وجود دارد: به صورت عملگر binary که بالاتر از آن استفاده کردیم و به صورت عملگر unary .

عملگر + که به یک مقدار اعمال می‌شود، هیچ عملی رو اعداد انجام نمی‌دهد. اما اگر operand یک عدد نباشد، عملگر + آن را به عدد تبدیل می‌کند.

به عنوان مثال :

کد

در واقع این همان کاریست که Number(…) انجام می‌دهد ولی به شکلی کوتاه‌تر.

نیاز به تبدیل رشته‌ به عدد اغلب پیش می‌آید. برای نمونه اگر در حال دریافت مقادیری از اینپوت‌های HTML هستیم معمولا به این موضوع نیاز پیدا می‌کنیم.

اگر خواستیم آنها را جمع ببندیم چطور ؟

عملگر + به صورت رشته‌ای آنها را بهم اضافه می‌کند :

کد

اگر خواستیم با آنها مانند عدد برخورد کنیم، آنها را به عدد تبدیل کرده و سپس آنها را جمع می‌کنیم:

کد

از دیدگاه یک ریاضی‌دان، تعدد علامت + ممکن است عجیب به نظر برسد، اما از دیدگاه یک برنامه‌نویس اینطور نیست. unary plusهایی که در ابتدا آمده‌اند رشته‌ها را عدد تبدیل می‌کنند و سپس binary plus اعداد را با هم جمع می‌کند.

چرا unary plus ها پیش از binary plus ها اعمال شدند ؟ به این دلیل که اولویت آنها بالاتر است.

**اولویت عملگرها**

اگر در یک عبارت بیش از یک عملگر وجود داشته باشد، ترتیب اجرای آنها بر اساس تقدم آنها خواهد بود.

از زمان مدرسه همه ما می‌دانیم که در یک عمل ریاضی مانند 1 + 2 \* 2 ابتدا عمل ضرب انجام می‌شود و سپس عمل جمع. این همان اولویت عملگرها است. اینکه عمل ضرب اولویت بالاتری نسبت به جمع دارد.

پرانتزها بر هر اولویتی، اولویت دارند و زمانی‌که از ترتیب قرارگیری عملگرها راضی نیستیم می‌توانیم با پرانتزها این اولویت را تغییر دهیم.

عملگرهای مختلفی در جاوا اسکریپت وجود دارد و هر کدام اولویت مربوط به خود را دارا می‌باشند. در جدول زیر عملگری که عدد بالاتری دارد اولویتش بالاتر است. همینطور اگر دو عملگر عدد یکسانی داشتند اولویت اجرا از چپ به راست (در کد) می‌باشد.

کد

همانطور که می‌بینیم unary + اولویت 16 دارد که از binary + با اولویت 13 بالاتر است. به همین دلیل است که در عبارت “+apples + +oranges” علامت‌های unary plus پیش از علامت جمع اجرا می‌شوند.

**مقداردهی**

در نظر داشته باشید که مقداردهی با علامت = نیز یک عملگر است. در جدول اولویت‌ها با اولویت پایینی (3) قرار گرفته است.

به همین دلیل است که وقتی متغیری را مقدار دهی می‌کنیم، مانند x = 2 \* 2 + 1 ، ابتدا عملیات محاسباتی انجام شده و سپس مقداردهی صورت می‌گیرد.

کد

همینطور زنجیر وار نوشتن مقداردهی ها امکان پذیر است :

کد

مقداردهی‌های زنجیر وار از سمت راست انجام می‌شوند. یعنی ابتدا 2 + 2 محاسبه می‌شود و در متغیر c سپس b و در آخر در a قرار می‌گیرد. عملا همه متغیرها یک مقدار دارای یک مقدار هستند.

**عملگر = یک مقدار را باز می‌گرداند**

یک عملگر همیشه مقداری را باز می‌گرداند. این موضوع در مورد عملگری چون جمع + یا ضرب \* واضح است، اما برای عملگر = هم چنین قانونی وجود دارد.

عبارت x = value ابتدا value را در x می‌نویسد و سپس آن را باز می‌گرداند.

در اینجا یک نمونه از مقداردهی به عنوان بخشی از یک عبارت پیچیده‌تر را داریم :

کد

در مثال بالا، مقدار (a = b + 1) مقداریست که در متغیر a نوشته قرار می‌گیرد (که برابر 3 است). سپس (در ادامه کد) از 3 کم می‌شود.

کد جالبی‌ست. ما باید طرز کار آن را یاد بگیریم تا زمانی که در کدهای کتابخانه‌های مختلف با آن روبرو می‌شویم بدانیم که چطور کار می‌کند. ولی نباید به این شکل برنامه‌نویسی کنیم چراکه کدهای ما را غیر واضح و غیر قابل خواندن می‌کند.

**عملگر باقی مانده %**

این عملگر بر خلاف ظاهرش، مربوط به درصد نمی‌باشد.

نتیجه محاسبه‌ی a % b برابر خواهد بود با باقی‌مانده‌ی تقسیم عدد a بر b.

برای نمونه :

کد

**عملگر بتوان \*\***

این عملگر اخیرا به جاوا اسکریپت اضافه شد.

برای عدد طبیعی b ، نتیجه محاسبه a \*\* b برابر خواهد بود با به تعداد b عدد a ضرب در خودش می‌شود.

برای نمونه :

کد

این عملگر برای اعداد غیر صحیح نیز کار می‌کند.

برای نمونه :  
کد

**عملگر افزایش / کاهش**

این عملگرها جزو رایج‌ترین نمونه‌های موجود در کار با اعداد هستند.

افزایش ++ : یک واحد به عدد اضافه می‌کند :

کد

کاهش -- : یک واحد از عدد کم می‌کند :

کد

**مهم :**

این عملگرها فقط بر روی اعداد کار می‌کنند و برای نمونه 5++ با خطا مواجه خواهد شد.

عملگرهای ++ و – می‌توانند پیش و پس از متغیر قرار گیرند.

وقتی پس از متغیر قرار بگیرد حالت پسوندی دارد counter++

وقتی پیش از متغیر قرار گیرد حالت پیشوندی دارد ++counter

آیا تفاوتی بین آنها وجود دارد؟ بله، اما فقط با مشاهده‌ی مقدار باز گردانده شده از آن، می‌توانیم این تفاوت را دریابیم.

حالت پیشوندی مقدار جدید را باز می‌گرداند درحالیکه حالت پسوندی مقدار قبلی را باز می‌گرداند.

برای نمونه :

کد

در خط (\*) حالت پیشوندی ++counter متغیر counter را یک واحد افزایش می‌دهد و مقدار 2 را باز می‌گرداند. در نتیجه alert مقدار 2 را نمایش می‌دهد.

کد

در خط (\*) حالت پسوندی counter++ مقدار counter را یک واحد افزایش می‌دهد ولی مقدار قبلی این متغیر را باز می‌گرداند. در نتیجه alert مقدار 1 را نمایش می‌دهد.

در نتیجه :

اگر مقدار بازگشتی از عملگرهای ++ و – مورد استفاده قرار نگیرد، تفاوتی در استفاده از آنها وجود ندارد :

کد

اگر می‌خواهیم مقداری را افزایش داده و بلافاصله از نتیجه عملگر استفاده نماییم، باید از حالت پیشوندی استفاده کنیم :

کد

اگر می‌خواهیم مقداری را افزایش داده و از مقدار قبلی آن استفاده نماییم باید از حالت پسوندی استفاده کنیم :

کد

**عملگرهای افزایش و کاهش در بین دیگر عملگرها**

عملگرهای ++ و – در عبارات (expressions) قابل استفاده هستند. اولویت آنها از اکثر عملگرهای ریاضیاتی بالاتر است.

برای نمونه :

کد

در مقایسه با :

کد

با اینکه از نظر فنی مشکلی ندارد ولی چنین روشی خوانایی کد را کاهش می‌دهد. اینکه یک خط کارهای مختلفی انجام می‌دهد مناسب نیست.

در هنگام خواندن کدها، چشمان به صورت عمودی و با سرعت کدها را می‌خوانند و چیزی مانند counter++ و افزایش مقدار counter به سادگی از چشم پنهان می‌ماند.

ما پیشنهاد می‌کنیم هر عمل را در یک خط بنویسید :

کد