Операторы и выражения

Что такое оператор?

- Оператор это символ который представляется как **вид операции над данными** в процессе выполнения
  - Требует **один или больше аргументов** (операндов)
  - Вычисляет **новое значение**
- **Операторы имеют приоритет**
- Выражения представляют собой последовательности операндов и операторов которые вычисляют определенное значение

Операторы в JavaScript

- Бывают:
  - **Унарные** – требует один операнд
  - **Бинарные** – требует два операнда
  - **Тернарные** (`?:`) – требует три операнда
- За исключением операторов присваивания, все операторы лево-ассоциативные
- Операторы присваивания и условный оператор являются право-ассоциативными

Категории операторов в JavaScript

Категория	Операторы
Арифметические	`+` `-` `*` `/` `%` `++` `--`
Логические	`&&` `||` `^` `!`
Бинарные	`&` `|` `^` `~` `<<` `>>` `>>>`
Сравнения	`==` `!=` `<` `>` `<=` `>=` `===` `!==`
Присваивания	`=` `+=` `-=` `*=` `/=` `%=` `|=` `^=` `<<=` `>>=`
Конкатенации	`+`
Другие	`.` `[]` `()` `?:` `new` `in` `,` `delete` `void` `typeof` `instanceof` `...`

Арифметические операторы

- Арифметические операторы `+`, `-`, `*`, `/` такие же как в математике
  - Оператор деления `/` возвращает число, `Infinity` или `NaN`
  - Деление в JavaScript возвращает число с плавающей точкой (i.e. 5 / 2 = 2.5)
- Оператор остатка `%` возвращает остаток от деления на число
- Оператор инкремента `++` увеличивает (когда как `--` уменьшает) на единицу число хранимое в переменной

```
const squarePerimeter = 17;
const squareSide = squarePerimeter / 4;
const squareArea = squareSide * squareSide;

console.log(squareSide); // 4.25
console.log(squareArea); // 18.0625

let a = 5;
let b = 4;

console.log(a + b); // 9
console.log(a + b++); // 9
console.log(a + b); // 10
console.log(a + (++b)); // 11
console.log(a + b); // 11

console.log(12 / 3); // 4
console.log(11 / 3); // 3.6666666666666665
```

```
console.log(11 % 3);   // 2
console.log(11 % -3);  // 2
console.log(-11 % 3);  // -2

console.log(1.5 / 0.0);  // Infinity
console.log(-1.5 / 0.0); // -Infinity
console.log(0.0 / 0.0);  // NaN

const x = 0;
console.log(5 / x);
```

Логические операторы

- Логические операторы принимают булевы значения и возвращают булевы значения
- Оператор `!` инвертирует `true` до `false` и `false` до `true`
- Операторы сравнения `&&`, `||` и `^` (`1 == true`, `0 == false`):

```
let a = true;
let b = false;

console.log(a && b); // False
console.log(a || b); // True
console.log(a ^ b); // True
console.log(!b); // True
console.log(b || true); // True
console.log(b && true); // False
console.log(a || true); // True
console.log(a && true); // True
console.log(!a); // False
console.log((5 > 7) ^ (a == b)); // False
```

`true`-подобные и `false`-подобные значения

- JavaScript, слабо типизированный язык, может использоваться любое значение,такое как `true` или `false`
- Любое значение может быть приведено к логическому с применением оператора приведения - `!!`

```
console.log(
  !!'', // пустая строка false
  !!'0', // не пустая строка true
  !!0, // ноль false
  !!35, // не нулевое число true
  !![], // обьект true
  !!NaN, // NaN  false
  !!'true', // true
  !!'false' // true,
  !!null, // любое нулевое и неопределенное false
  !!undefined
);
```

Битовые операторы

- Битовые операторы используют числовые значения. Они применяются побитово.
- Оператор `~` переводит `0` к `1` и `1` к `0`
  - Подобно `!` как в булевах применяется побитово
- Операторы `|`, `&` и `^` так же как и `||`, `&&` и `^` в булевах только побитово
- `<<` и `>>` сдвигают (перемещают) биты в лево или вправо

```
let a = 3;               // 00000000 00000011
let b = 5;               // 00000000 00000101

console.log(a | b);      // 00000000 00000111
console.log(a & b);      // 00000000 00000001
console.log(~a & b);     // 00000000 00000100
console.log(a ^ b);      // 00000000 00000110

console.log(true << 1);  // 00000000 00000010
console.log(true >> 1);  // 00000000 00000000
```

Операторы сравнения

- Применяются для сравнения переменных
  - `==` , `<` , `>` , `>=` , `<=` , `!=` , `===` , `!==`
  - Для определения эквивалентности `===` и `!==` предпочтительны

```
let a = 5;
let b = 4;

console.log(a >= b);   // True
console.log(a != b);   // True
console.log(a == b);   // False
console.log(0 == '');  // True
console.log(0 === ''); // False
```

Операторы присваивания

- присваивают знаение переменной
  - `=` , `+=` , `-=` , `|=` , ...

```
let x = 6;
let y = 4;

console.log(y *= 2); // 8

let z = y = 3;       // y=3 and z=3

console.log(z);      // 3
console.log(x |= 1); // 7
console.log(x += 3); // 10
console.log(x /= 2); // 5
```

Другие операторы

- Оператор конкатенации строка `+`

- Если второй из операндов не является строкой, то он приводится к ней.

```
let first = "First";
let second = "Second";

console.log(first + second); // FirstSecond

let output = "The number is : ";
let number = 5;

console.log(output + number); // The number is : 5
```

- Оператор доступа к члену `.` используется для доступа к члену объекта
- Квадратные скобки `[]` используются как индексы для доступа к члену обьекта или массива
- Скобки `(` `)` используются для переопределения стандартного потока очередности вычислений или вызова функций
- Оператор условия `?:` имеет форму
  - (if `b` true то результат `x` иначе результат `y`)

```
b ? x : y
```

- `new` оператор создания нового объекта
- `typeof` оператор возвращающий тип обьекта

```
let a = 6;
let b = 4;

console.log(a > b ? 'a > b' : 'b >= a'); // a > b

let c = b = 3; // b = 3;

console.log(c); // 3
console.log(new Number(6) instanceof Number); // true
console.log(6 instanceof Number); // false
console.log((a + b) / 2); // 4
console.log(typeof c); // number
console.log(void(3 + 4)); // undefined
```

Приоритет операторов

- Если сомневаетесь можете обратиться к справочнику MDN Precedence chart
- Операторы скобок всегда имеют высший приоритет
- **Считается хорошей практикой применять круглые скобки, даже если это не обязательно**

- Делают код более читаемым

Выражения

- Выражения это последовательности операторов, литералов и переменных, которые возвращают результат вычислений

```
let r = (150 - 20) / 2 + 5; // r = 70

// Выражения для высисления площади круга
let surface = Math.PI * r * r;

// Выражение для вычисления периметра окружности
let perimeter = 2 * Math.PI * r;
```

- Выражения имеют:
  - **Тип** (integer, real, boolean, ...)
  - **Значение**

```
let a = 2 + 3; // a = 5

let b = (a + 3) * (a - 4) + (2 * a + 7) / 4;  // b = 12

let greater = (a > b) || ((a == 0) && (b == 0));
```