# Урок 3.2. Типы данных. Преобразование типов. Выражения

## Материалы

- Приоритет операторов MDN

## Summary

### Термины

- Выражение - часть кода, содержащая вычисление какого либо значения.
- Оператор - часть выражения, показывающая, какое именно вычисление необходимо сделать с данными операндами
- Операнд - то, к чему применяется оператор. В выражении `10 * 2` 10 и 2 - операнды.
- Унарный оператор - оператор, требующий только одного операнда: `+5`
- Бинарный оператор - оператор, требующий двух операндов: `5 + 25`
- Общий вид выражения в JS, использующего бинарный оператор - `operand1 operator operand2`

### Математика

- В JS существуют стандартные математические операторы: `+`, `-`, `*`, `/`
- Также есть математические операторы "не из курса арифметики":
  - `%` - вычислить остаток от деления (Например, результатом `26 % 5` будет `1`)
  - `**` - возведение в степень. `5 ** 2` - это 5 в квадрате. Слева число, которое возводим, справа - степень.

## Строки

- Оператор `+` используется не только в математике, но и в строках
- Оператор выполняет склейку двух строк.
- Если хотя бы один из операндов - строка (независимо, справа или слева от `+`) - выполнится именно строковое сложение

## Унарный +

- Кроме бинарного `+`, существует также унарный.
- Унарный `+` выполняет приведение операнда к числовому типу

```
console.log(+'526'); // 526
console.log(+'wewe'); // NaN
console.log(+true); // 1
console.log(+false); // 0
```

## Приоритет операторов

- Как и в математике, у каждого оператора есть свой приоритет для того, чтобы знать в каком порядке вычислять выражение
- Например, умножение приоритетнее сложения и т.д.
- Вся таблица приоритетов в материалах

## Присваивание

- Присваивание - это тоже оператор
- У присваивания приоритет `3` - низкий. Чтобы сначала вычислялось значение выражения, и только потом выполнялось присваивание:

```
// сначала вычислится выражение справа от знака =, и только потом присвоится константе `num`
const num = 5 + 5 * 5;
```

## Сокращённая арифметика с присваиванием

```
let n = 1;
n = n * 2;
n = n + 2;
```

Выражения выше можно заменить на более короткие: домножение и досложение:

```
let n = 1;
n *= 2; // домножаем на 2
n += 2; // прибавляем к n 2
```

Приоритет у такой операции такой же, как у присваивания:

```
let n = 4;// поскольку приоритет как у присваивания, сначала выполнится 5 + 5,
// и затем n домножится на 10
n *= 5 + 5;
```

## Инкремент / декремент

- Инкремент увеличивает переменную на 1
- Декремент уменьшает переменную на 1
- Эти операторы могут применяться только к числовым переменным

```
let n = 0;
n++;
console.log(n); // n = 1
n--;
console.log(n); // n = 0
```

- Существует префиксная `++i` и постфиксная `i++` форма записи у обоих операторов
- Разница состоит в возвращаемом значении (если мы используем его сразу)

```
let n = 0;
console.log(n++); // выведется 0, так как постфиксная форма возвращает предыдущее значение
```

```
console.log(n); // выведется 1
```

```
let n = 0;
console.log(++n); // выведется 1, так как префиксная форма возвращает новое значение
console.log(n); // выведется 1
```

# Преобразование типов

- Три основных вида преобразования: числовые, логические, строковые
- У каждого вида - два типа: явные и неявные
- Явные делаете вы в коде сами с помощью специальных методов
- Неявные делает сам JS под капотом при вычислении каких-либо выражений

```
// Числовые преобразования
const str = '123';
const num = Number(str); // преобразует str к числу 123
// ------ //
const str = 'qwe';
const num = Number(str); // ошибки не будет, но результат будет NaN - not a number (так ка
к преобразуемая строка - не число)
// ------ //
const str = '     12345      ';
const num = Number(str); // корректно преобразуется в число 12345 (пробелы по краям строки
убираются)
// ------ //
const bool = true;
const num = Number(bool); // Булевское true преобразуется к 1, false - к 0.
// ------ //
const str = '123';
const num = +str; // неявное преобразование к числу
```

```
// Строковые преобразования
const bool = true;
const str = String(bool); // булевское приводится к строке 'true' или 'false'
// ------ //
const num = 25;
const str = String(num); // Число приводится к строке '25'
// ------ //
const num = 25;
const str = '' + num; // неявное преобразование в строку - прибавляем пустую строчку.
```

```
// Логические преобразования
// Все значения преобраазуются в true, за исключением falsey-значений:
// undefined, NaN, null, 0, ''
const str = '';
const bool = Boolean(str); // false;
// ------ //
const str = 'adad';
const bool = Boolean(str); // true
// ------ //
const str = '2';
const bool = !!str; // неявное приведение к boolean
```