Основы JavaScript

Операторы, ветвление и циклы

Кирилл Талецкий

TeachMeSkills
27 июля 2023

Что будет

- Базовые операторы
  - Математические
  - Сравнения
  - Логические
- **Перерыв**
- Условия
- Конструкция switch
- Циклы

Базовые операторы

Глоссарий

- **Операнд** — то, к чему применяется оператор.
  - Например, в выражении `2 + 3` операндами являются числа 2 и 3
- **Унарный оператор** — применяется к одному операнду
  - Например, унарный оператор `-` меняет знак числа на противоположный
- **Бинарный оператор** — применяется к двум операторам
  - Тот же `-` как бинарный оператор выполняет вычитание чисел

Математические операторы

Математические операции

- Сложение +
- Вычитание -
- Умножение *
- Деление /
- Взятие остатка от деления %
- Возведение в степень **
  - Тот же оператор — взятие корня

```
6 % 3 = 0
```

```
7 % 3 = 1
```

```
8  % 3 = 2
```

```
2**2 = 4
```

```
2**3 = 8
```

```
2**4 = 16
```

```
4**1/2 = 2
```

```
8**1/3 = 2
```

```
9**1/2 = 3
```

Конкатенация строк

- Бинарный `+` может складывать строки
- Если один из операндов строка, то второй операнд будет приведён к строке
- Если операндов несколько, то приведение сработает последовательно
- Приведение к строке работает только для `+`, остальные математические операторы всегда приводят операнды к числам

Как можно привести значение к числу?

Ещё один способ

- Унарный `+`
- Работает так же как и Number(value), но запись более короткая

Составное присваивание

`+=` `-=`

`*=` `/=`

```
a += b
```

Эквивалентно

```
a = a + b
```

Инкремент / Декремент

`++` `--`

```
a++
```

Эквивалентно

```
a = a + 1
```

Инкремент / Декремент

Префиксная форма

```
let a = 1;
const b = ++a;

console.log(b) // 2
```

Пригодится, если нужно вернуть значение после инкрементирования

Постфиксная форма

```
let a = 1;
const b = a++;

console.log(b) // 1
```

Пригодится, если нужно вернуть значение до инкрементирования

Логические операторы

Как в PHP сравнить числа?

Сравнение


```
const a = 5
const b = 7

const c = a < b;
console.log(c) // true
```

Операторы сравнения возвращают булевы значения (true/false)

Равенство

`===` `!==`

`a === b`

`a !== b`

```
const a = 5
const b = 7

const c = a === 5;
console.log(c) // true

const d = b !== 7;
console.log(d) // false

const e = a === "5"
console.log(e) // false (!)
```

Операторы равенства возвращают булевы значения (true/false)

Нестрогое равенство

Сначала приводит тип переменных к одному, затем производит сравнение

`==` `!=`

`a == b`

`a != b`

```
const a = 5
const b = 7

const c = a == 5;
console.log(c) // true

const d = b != 7;
console.log(d) // false

const e = a == "5"
console.log(e) // true (!)
```

Можно ли в PHP сравнивать типы отличные от чисел?

Сравнение строк

- Строки в JS сравниваются в “лексиографическом” порядке

```
console.log('Z' > 'A') // true
console.log('cat' > 'cam') // true
console.log('caterpillar' > 'cat') // true
```

- Сначала сравниваются первые символы строк.
- Если первый символ первой строки больше (меньше), чем первый символ второй, то первая строка больше (меньше) второй. Сравнение завершено.
- Если первые символы равны, то таким же образом сравниваются уже вторые символы строк.
- Сравнение продолжается, пока не закончится одна из строк.
- Если обе строки заканчиваются одновременно, то они равны. Иначе, большей считается более длинная строка.

- Для сравнения символов, используется их порядковый номер в юникод таблице (поэтому регистр имеет значение)

Сравнение разных типов

- При сравнении разных типов, JS приведёт их к числу

```
console.log('5' > 3) // true - 5 больше чем 3
console.log('01' == 0) // false - 1 не равно 0

console.log(true == 1) // true
console.log(false == 0) // true
```

- Обратите внимание, что NaN ни с чем не сравнивается (всегда false) и ничему не равно (даже самому себе)
  - Это поведение подчиняется общему стандарту для чисел с плавающей точкой IEEE 754

Сравнение с null и undefined

null и **undefined** равны только сами себе и друг другу

Это специальное правило в **JS!**

Сравнение с null и undefined

null и undefined равны только сами себе и друг другу — это специальное правило в JS!

```
console.log(null == undefined) // true
console.log(null === undefined) // false - типы не совпадают

console.log(null > 0) // false - приведение к числу 0
console.log(null >= 0) // true - приведение к числу 0
console.log(null == 0) // false - нет приведения! null равен только самому себе и undefined;

console.log(undefined > 0) // false - нет приведения
console.log(undefined >= 0) // false - нет приведения
console.log(undefined == 0) // false - нет приведения
```

Как с этим жить

- Всегда, где возможно, используйте только строгое равенство `===` / `!==`
- Не используйте операторы математического сравнения с переменными которые могут быть null и undefined. Если сравнить всё же нужно, добавьте отдельную обработку.

Перерыв

Логические операторы

Логические операторы

||

```
console.log(true || true); // true
console.log(true || false); // true
console.log(false || true); // true
console.log(false || false); // false
```

Возвращает первое truthy значение

&&

```
console.log(true && true); // true
console.log(true && false); // false
console.log(false && true); // false
console.log(false && false); // false
```

Возвращает первое falsy значение

!

```
console.log(!true); // false
console.log(!false); // true
```

Возвращает булево значение

Логические операторы

Оператор нулевого слияния

??

```
console.log(null ?? 0) // 0
console.log(undefined ?? "" ?? false) // false
console.log(null ?? undefined ?? NaN ?? 'kek') // NaN
```

Возвращает первое значение не равное `null` или `undefined`

Условия

Условное ветвление if ()

```
if (a === 0) {
  console.log('zero')
}
```

```
if (a === 0) {
  console.log('zero')
} else {
  console.log('non-zero')
}
```

```
if (a === 0) {
  console.log('zero')
} else if (a > 0) {
  console.log('positive')
} else {
  console.log('negative')
}
```

Для простых выражений возможна также короткая запись без фигурных скобок

В условие можно записать любое JS выражение!

Условное ветвление if ()

- Оператор if (value) всегда приводит значение к булевому типу

Повторим: какие значения других типов будут приведены к false?

Тут стоит вспомнить как работает явное приведение к логическому типу через Boolean(value)

Сталкивались ли вы с понятием “тернарный оператор”?

Какой у него синтаксис в PHP?

Тернарный оператор

```
condition ? true : false
```

```
const result = a === 0 ? 'zero' : 'non-zero'
```

Можно так же писать вложенные тернарные выражения, но это не рекомендуется

Конструкция switch

Есть ли конструкция switch-case в PHP?

Конструкция switch

- Заменяет несколько if ()

```
switch (state) {
  case 'loading':
    console.log('подключаемся...');
    break;
  case 'connected':
    console.log('подключение успешно');
    break;
  case 'error':
    console.log('упс, что-то пошло не так')
    break;
  default:
    console.log('ошибка: неизвестный статус ')
}
```

- Тип имеет значение
- Опция по умолчанию (default) не обязательна, но её рекомендуется использовать
- Возможна группировка нескольких case в один

Циклы

Какие циклы есть в PHP?

While

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Условие

Тело цикла

While

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Условие

Тело цикла

Что будет выведено в консоль?
Как тот же самый цикл записать короче?

While

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

Условие

Тело цикла

Что будет выведено в консоль?
Как тот же самый цикл записать короче?

Интерпретатор не войдёт в цикл, если условие не будет выполнено

Чтобы гарантировано хотя бы раз выполнить тело цикла, используйте do ... while

Контроль потока выполнения в циклах

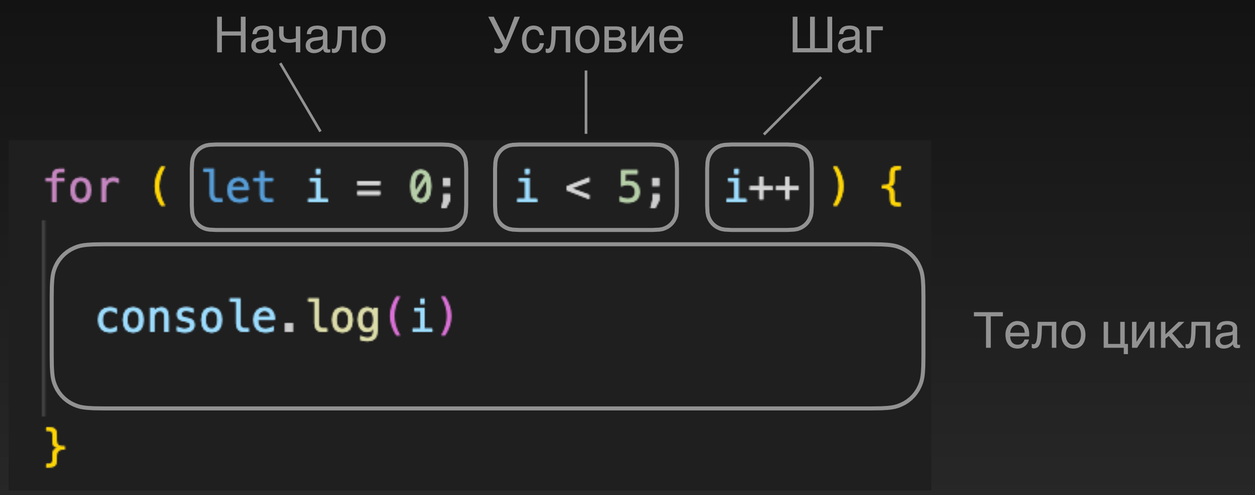
`break`

Немедленно прерывает весь цикл

`continue`

Переход к выполнению следующей итерации без выполнения текущей

For



Обсуждение Домашнего Задания

Формат сдачи материала — Pull Request

Практика