ТИНЬКОФФ

Основы JavaScript

Сафонов Олег

Март 2023

Преподаватель

Сафонов Олег

Ведущий разработчик Tinkoff
12 лет опыта
Профильное образование MSIT-SE

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Почему JavaScript?

- ~65% разработчиков пишут на JS
- > 1.5 миллиардов сайтов используют JS
- > 800 000 npm пакетов

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Инструменты

- Редакторы кода
  - WebStorm
  - Visual Studio Code
  - Sublime Text
- Стиль и форматирование
  - Eslint
  - Prettier
- Отладка
  - Chrome\Firefox DevTools

Инструменты - 2

- Песочницы
  - StackBlitz
  - CodeSandbox
  - CodePen
  - JsFiddle
- Документация
  - Mozilla Developer Network
  - DevDocs
  - Can I Use

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Вводная

JS - слабо типизированный язык с динамической типизацией

```
1  // Динамическая типизация
2  let dynamicTypeSystem = [
3      // ??
4      // ??
5  ];
6
7
8
9
10
11
```

Динамическая типизация

JS - слабо типизированный язык с динамической типизацией

```
// Динамическая типизация
let dynamicTypeSystem = [
    "Тип переменной определяется не при объявлении, а при присваивании значения",
    "Одна переменная может использоваться для хранения разных типов данных"
];
```

Ключевое слово let

Переменная - именованная область памяти. Позволяет читать и записывать значение.

```
1  let name; // Объявление переменной
2
3  name = "Олег"; // Присвоение значения переменной
4
5  // Чтение значения переменной
6  console.log(name); // "Олег"
7
8  let surname = "Сафонов"; // Можно объединить объявление и присваивание значения
9
10 // Нельзя объявить ещё одну переменную с таким же названием
11 let name;
```

Ключевое слово const

Константа - именованная область памяти. Позволяет читать значение, но не менять его.

```
1  const SECONDS_IN_MINUTE = 60; // Объявление константы и присваивание значения
2
3  // Чтение значения константы
4  console.log(SECONDS_IN_MINUTE); // 60
5
6  // Объявить константу без значения нельзя
7  const SECONDS_IN_MINUTE; ❌
8
9  // Изменить значение константы нельзя
10 SECONDS_IN_MINUTE = 50; ❌
11
12 // Нельзя объявить ещё одну переменную с таким же названием
13 const SECONDS_IN_MINUTE = 50; ❌
```

Ключевое слово var

Переменная - именованная область памяти. Позволяет читать и записывать значение.

```
var name; // Объявление переменной

name = "Олег"; // Присвоение значения переменной

// Чтение значения переменной
console.log(name); // "Олег"

var surname = "Сафонов"; // Можно объединить объявление и присваивание значения

var name; // Можно объявить ещё одну переменную с таким же названием
```

Ключевые слова

	let	const	var
Можно объявить несколько раз	✗	✗	✓
Можно переприсвоить	✓	✗	✓
Обязательно задать значение	✗	✓	✗

https://learn.javascript.ru/let-const

Правила именования

- ✔ Буквы
- ✔ Цифры (не в начале)
- ✔ Символы _ и $
- ✖ Цифры в начале
- ✖ Зарезервированные слова

```
1  // Правильно
2  let _ = 1;
3  let $ = 1;
4  let o_o = 1;
5  let test1 = 1;
6
7  // Неправильно
8  let 666 = "Я начинаюсь с цифры"
9  let const = "const зарезервировано";
```

Зарезервированные слова

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Reserved words

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Типы данных

	string	number	bigInt	boolean	undefined	null	symbol	object
Тип	Строка	Число	Большое целое	Логический	Неопределённый	Null	Символ	Структура
Примитив	✔	✔	✔	✔	✔	✔	✔	✖
typeof	“string”	“number”	“bigint”	“boolean”	“undefined”	“object”	“symbol”	“object”
Пример	“привет”	3.14 1.1e32 0xFF	1924924124n	true false	undefined	null	Symbol()	{ name: “Batman” }

✔ Функция - особый, вызываемый объект

```
1  function someFunction() {};
2  console.log(typeof someFunction); // "function"
```

Что такое примитивный тип?

- Не объект, конкретное значение
- Не может быть изменён (иммутабельный).
  При изменении создаётся новый объект
- Не имеет методов.
  При вызове метода создаётся объект-обёртка

```
1  const name = "batman";
2  console.log(name); // "batman"
3
4  // Методы строки не мутируют строку
5  name.toUpperCase();
6  console.log(name); // "batman"
7
8  // Методы строки возвращают новую строку
9  const upperName = name.toUpperCase();
10 console.log(upperName); // "BATMAN"
11
12 // Под капотом создаётся объект-обёртка
13 const upperName2 = new String(name).toUpperCase();
14 console.log(upperName2); // "BATMAN"
```

Undefined

```
let minds = [
      "Тип переменной определяется при присваивании значения",
      "Можно объявить переменную без присваивания ей значения"
];
```

Undefined

Неопределённое значение

- Автоматически присваивается переменным, для которых не указано значение
- Автоматически присваивается аргументам функции, для которых не были переданы значения
- Если функция ничего не возвращает, то она возвращает undefined
- Несуществующие свойства объекта имеют значение undefined

```
let value;
console.log(typeof value); // "undefined"

// Переменная name нигде не определена

if (typeof name === "undefined") // "OK"
if (name === undefined) // "ReferenceError"
```

Null

Отсутствие значения

✔ Не присваивается автоматом

```
1  console.log(typeof null === "object"); // true
```

Boolean

Логическое значение

- ✔ Примитив boolean и объект-обёртка Boolean
- ✔ Принимает значение true (истина) или false (ложь)

```
1  const isValid = true;
2  const isOk = false;
```

Symbol

Уникальный идентификатор

Используется как идентификатор для скрытых свойств объектов

```
1  console.log(Symbol.for("a")); // Symbol(a)
2  console.log(Symbol.for("b")); // Symbol(b)
3
4  console.log(Symbol("a")); // Symbol(a)
5  console.log(Symbol("b")); // Symbol(b)
```

String

Последовательность символов для представления текста

- Примитив string и объект-обёртка String
- Значение пишется в кавычках

```
// Кавычки двух видов
const singleQuotes = 'Привет';
const doubleQuotes = "Привет";
const quote = "'Привет' - это слово";

console.log(singleQuotes); // "Привет"
console.log(doubleQuotes); // "Привет"
console.log(quote); // "'Привет' - это слово"

// Шаблонные строки
const name = 'Пользователь';
const greeting = `Привет, ${name}`;
console.log(greeting); // "Привет, Пользователь"
```

Работа со строками

```
1  // Конкатенация
2  const concatenatedString = "Собираем" + "строку";
3  console.log(concatenatedString); // "Собираемстроку"
4
5  // Методы
6  const login = "MyLogin";
7  console.log(login.length); // 7
8  console.log(login[0]); // "M"
9  console.log(login.toUpperCase()); // "MYLOGIN"
10 console.log(login.toLowerCase()); // "mylogin"
```

Number

- Примитив number и объект-обёртка Number
- Числовой тип в формате 64-битного числа двойной точности с плавающей запятой
- 64 бита, от -10^{307} до $+10^{307}$

```
1  const result = 0.1 + 0.2;
2
3  console.log(result);
```

Floating Point Math

Арифметические операции

```
1  console.log(3 + 2); // 5
2  console.log(3 - 2); // 1
3  console.log(3 * 2); // 6
4  console.log(3 / 2); // 1.5
5  console.log(3 % 2); // 1
6
7  console.log(3 / 0); // ??
```

Infinity

```
1  console.log(3 / 0); // Infinity
2  console.log(-3 / 0); // -Infinity
3
4  console.log(3 / Infinity); // 0
5  console.log(-3 / Infinity); // -0
6
7  console.log(Infinity * Infinity); // Infinity
8  console.log(Infinity * 0); // ??
```

NaN (Not a Number)

```
1  console.log(Infinity * 0); // NaN
2  console.log(Infinity - Infinity); // NaN
3  console.log(Infinity / Infinity); // NaN
4  console.log(Infinity % 5); // NaN
5
6  console.log(typeof NaN); // ??
7  console.log(NaN === NaN); // ??
```

NaN (Not a Number) - 2

```
1  console.log(Infinity * 0); // NaN
2  console.log(Infinity - Infinity); // NaN
3  console.log(Infinity / Infinity); // NaN
4  console.log(Infinity % 5); // NaN
5
6  console.log(typeof NaN); // "number"
7  console.log(NaN === NaN); // false
8
9  console.log(Number.isNaN(NaN)); // true
10 console.log(Number.isNaN("строка")); // false
```

Работа с числами

```
// Статические методы Number
console.log(Number.MIN_VALUE); // 5e-324
console.log(Number.MAX_VALUE); // 1.7976931348623157e+308

// Math - математические операции и функции
console.log(Math.PI); // 3.141592653589793
console.log(Math.sqrt(16)); // 4
console.log(Math.pow(2, 10)); // 1024

// Методы
const price = 10;

console.log(price.toFixed(1)); // "10.0"
console.log(price.toFixed(2)); // "10.00"
console.log(price.toFixed(3)); // "10.000"
```

BigInt

- Только целые числа
- Диапазон зависит от конкретной реализации и окружения
- Нельзя смешивать с Number и использовать с Math
- Используется в финансовых технологиях, в высокоточных метках времени

```
1  const bigInt = 10n;
2  const anotherBigInt = BigInt(10);
3
4  console.log(bigInt === anotherBigInt); // true
5
6  const invalidBigInt = BigInt(10.5);
```

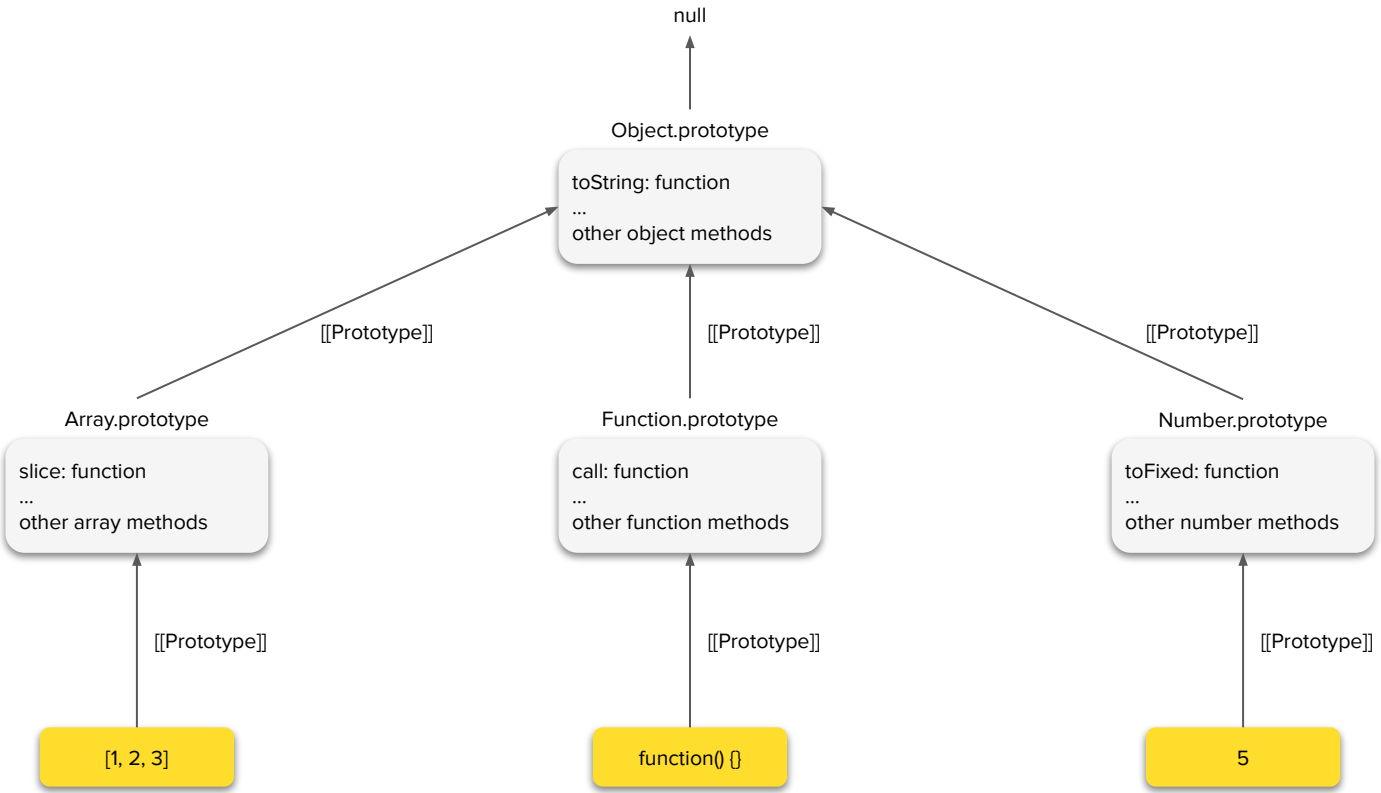
Object

- Не примитив
- Набор свойств различных типов (включая вложенные объекты)
- Получает методы не через объект-обёртку, а через прототип
- Значение в памяти, можно ссылаться

```
const hero = {
    name: "Batman",
    tools: [
        { name: "Batarang" },
        { name: "Batmobile" },
    ],
    isReady: function() {
        return true;
    },
    birthDay: new Date(1915, 03, 07)
};

console.log(hero.name); // "Batman"
console.log(hero.isReady()); // true
console.log(hero.birthDay.toDateString()); // "Apr 07 1915"
```

Object


null
Object.prototype
toString: function
...
other object methods
[[Prototype]]
[[Prototype]]
[[Prototype]]
Array.prototype
slice: function
...
other array methods
Function.prototype
call: function
...
other function methods
Number.prototype
toFixed: function
...
other number methods
[[Prototype]]
[[Prototype]]
[[Prototype]]
[1, 2, 3]
function() {}
5

Массив

Особый тип объекта, предназначенный для работы с упорядоченным набором элементов

- Length возвращает последний индекс + 1
- toString() возвращает перечисление элементов через запятую
- Обращение к элементам по индексу через квадратные скобки

```
1  const array = ["first", "second"];
2
3  const anotherArray = new Array("first", "second");
4
5  console.log(array.length); // 2
6  console.log(array[0]); // "first"
```

Функция

```
function getHelloMessage(name) {
    return `Привет, ${name}!`;
}

function displayMessage(message) {
    console.log(message);
}

displayMessage(getHelloMessage("Иван")); // "Привет, Иван!"
```

Динамическая типизация

Тип переменной определяется при присваивании значения
Одна переменная может использоваться для хранения разных типов данных

```
1  // Типизация
2  let hero;
3  console.log(typeof hero); // "undefined"
4
5  hero = "Batman";
6  console.log(typeof hero); // "string"
7
8  hero = { name: "Batman" };
9  console.log(typeof hero); // "object"
```

Слабая типизация

JS - слабо типизированный язык с динамической типизацией

```
1  // Динамическая типизация
2  let dynamicTypeSystem = [
3      "Тип переменной определяется не при объявлении, а при присваивании значения",
4      "Одна переменная может использоваться для хранения разных типов данных"
5  ];
6
7  // Слабая типизация
8  let weakTypeSystem = [
9      // ??
10     // ??
11 ];
```

Слабая типизация - 2

JS - слабо типизированный язык с динамической типизацией

```
// Динамическая типизация
let dynamicTypeSystem = [
    "Тип переменной определяется не при объявлении, а при присваивании значения",
    "Одна переменная может использоваться для хранения разных типов данных"
];

// Слабая типизация
let weakTypeSystem = [
    "Можно смешивать в выражениях различных типы",
    "Поддерживаются неявные преобразования типов"
];
```

Слабая типизация

Можно смешивать в выражениях различных типы
Поддерживаются неявные преобразования типов

```
1  // Приведение типов
2  const hour = 10; // Число
3  const name = "Олег"; // Строка
4
5  console.log(hour + name); // ??
6
7  const hero = { name: "Batman" };
8  console.log(hour + hero); // ??
```

Преобразование к строке

```
1  // Явное приведение
2  console.log(String(null)); // "null"
3
4  // Неявное приведение
5  console.log("Строка" + undefined); // "Строкаundefined"
6  console.log(10 + {}); // "10[object Object]"
7  console.log(10 + []); // "10"
8  console.log(10 + [1]); // "101"
9  console.log(10 + ["Строка"]); // "10Строка"
```

Преобразование к числу

```
1  // Явное приведение
2  console.log(Number(null)); // ??
3  console.log(Number("")); // ??
4  console.log(Number(undefined)); // ??
5  console.log(Number({})); // ??
6  console.log(+"10"); // ??
7  console.log(-"1"); // ??
8
9  // Неявное приведение
10 // Рассмотрим дальше
```

Преобразование к числу - 2

```
1  // Явное приведение
2  console.log(Number(null)); // 0
3  console.log(Number("")); // 0
4  console.log(Number(undefined)); // NaN
5  console.log(Number({})); // NaN
6  console.log(+"10"); // 10
7  console.log(-"1"); // -1
8
9  // Неявное приведение
10 // Рассмотрим дальше
```

Преобразование к логическому значению

```
1  // Явное приведение
2  console.log(Boolean(0)); // false
3  console.log(Boolean(undefined)); // false
4  console.log(Boolean(null)); // false
5  console.log(Boolean("")); // false
6  console.log(Boolean(NaN)); // false
7
8  console.log(!0); // true
9  console.log(!!0); // false
10
11 // Невное приведение
12 // Рассмотрим дальше
```

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Инкремент и декремент

Увеличивают или уменьшают на единицу значение переменной и возвращают новое (префиксный) либо исходное (потсфиксный) значение

```
let x = 0;
let y = x++;
console.log(x, y); // 1, 0

let x = 0;
let y = x--;
console.log(x, y); // -1, 0

let x = 0;
let y = ++x;
console.log(x, y); // 1, 1

let x = 0;
let y = --x;
console.log(x, y); // -1, -1
```

Операторы сравнения

- Конвертируются в примитив number с хинтом “number” (Symbol.ToPromitive(hint))
- Если оба операнда - строки, то сравниваются как строки
- Если хотя бы один операнд не является строкой, то операнды приводятся к числам
- Если хотя бы один NaN, возвращается false

```
1  console.log("a" <= "b"); // true
2  console.log("a" < "a"); // false
3  console.log("a" < "3"); // false
4
5  console.log("5" <= 3); // false
6  console.log("3" <= 5); // true
7
8  console.log("hello" >= 5); // false
9  console.log(5 >= "hello"); // false
10
11 console.log(false > true); // false
12 console.log(true > false); // true
```

Операторы сравнения - 2

	===	!==	==	!=
Тип	Строгое	Строгое	Нестрогое	Нестрогое
Приводит типы	✗	✗	✓	✓
Используем?	✓	✓	✗	✗

```
1  console.log(false == ""); // true
2  console.log(true == 1); // true
3  console.log("1" == 1); // true
4
5  console.log(false === ""); // false
6  console.log(true === 1); // false
7  console.log("1" === 1); // false
```

Логические операторы

&& - логическое И
|| - логическое ИЛИ
! - логическое НЕТ

A	B	A && B	A || B	!A	!B
true	true	true	true	false	false
true	false	false	true	false	true
false	true	false	true	true	false
false	false	false	false	true	true

Возвращаемое значение

&& - возвращает первое falsy значение или последнее
|| - возвращает первое truthy значение или последнее

```
1  console.log("строка" && null); // null
2  console.log("строка" && 0); // 0
3  console.log(null && stringValue); // null
4  console.log(false && nullValue); // false
5
6  console.log(true && "строка"); // "строка"
7  console.log("строка" && true); // true
8
9  console.log("строка" || 0); // "строка"
10 console.log(0 || "строка"); // "строка"
11 console.log(0 || null); // null
12 console.log(null || 0); // 0
13
14 console.log(true || "строка"); // true
15 console.log("строка" || true); // "строка"
```

Тернарный оператор

условие ? выражение1 : выражение2

```
1  console.log(true ? "Истина" : "Ложь"); // "Истина"
2
3  const isReady = true;
4  const status = isReady ? "Готов" : "Не готов";
5
6  console.log(status); // "Готов"
```

Оператор опциональной последовательности

Позволяет безопасно получить значение свойства на любом уровне вложенности

```
const config = {
    ui: {
        size: {
            width: 400
        }
    }
};

// Сейчас
console.log(config.ui?.size?.width); // 400
console.log(config.ui?.size?.height); // undefined
console.log(config.default?.size?.height); // undefined

// Раньше (<2020)
console.log(config.default && config.default.size && config.default.size.height); // undefined
```

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

SyntaxError

Синтаксическая ошибка, JS движок не смог распарсить код

```
1  const login;
2
3
4
```

```
Uncaught SyntaxError: Missing initializer in const declaration.
    at script.js:1:7
```

TypeError

Невозможность выполнить операцию, чаще всего из за некорретного типа

```
1  const login = "login";
2
3  login = "secret_login";
4
```

```
Uncaught TypeError: Assignment to constant variable
    at script.js:3:7
```

ReferenceError

Обращение к несуществующей переменной

```
1  const MAX_VLAUE = 123;
2
3  console.log(MAX_VALUE);
4
```

```
Uncaught ReferenceError: MAX_VALUE is not defined
    at script.js:3:13
```

Логические ошибки

Программа работает неправильно

```
1  let first = 10; // Число
2  let second = "50"; // Строка
3
4  console.log(first + second); // "1050"
```

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Блок try - catch

Ловим исключения

```
try {
    const result = myObject.test();
} catch (exception) {
    console.log("Произошла ошибка");
    console.log(exception);
} finally {
    console.log("Блок try catch выполнен");
}
```

Условие if

Выполняет код в случае выполнения условия

```
1  const count = 0;
2
3  if (count > 0) {
4      console.log("Привет"); // Этот код не выполнится
5  }
```

Условие if - блок else

Выполняет одну из веток, в зависимости от выполнения условия

```
1  const count = 0;
2
3  if (count > 0) {
4      console.log("Привет"); // Этот код не выполнится
5  } else {
6      console.log("Привет"); // Этот код выполнится
7  }
```

Условие if - блок else if

Выполняет одну из веток, в зависимости от выполнения условия

```
1  const count = 0;
2
3  if (count > 0) {
4      console.log("Привет"); // Этот код не выполнится
5  } else if (count > 10) {
6      console.log("Привет"); // Этот код не выполнится
7  } else {
8      console.log("Привет"); // Этот код выполнится
9  }
```

switch

```
1  const count = 0;
2  let message;
3
4  switch (count) {
5      case 1: // Если count === 1
6          message = "Один";
7          break;
8      case 2:
9      case 3:
10     case 4: // Если count === 2 || count === 3 || count === 4
11         message = "Несколько";
12         break;
13     default: // Во всех остальных случаях
14         message = "Много";
15         break;
16 }
17
18 console.log(message); // "Много"
```

Циклы while и do...while

```
1  // Выполняем действие до проверки условия
2  // do { ... } while (...);
3
4  // Выполняем действие после проверки условия
5  // while (...) do { ... };
6
7  let count = 0;
8
9  while (count < 5) {
10     count++;
11     console.log(count); // 1 2 3 4 5
12 }
```

Цикл for

for ([инициализация]; [условие выхода]; [финальное выражение]) { ... }

```
1  const count = 5;
2
3  for (let index = 1; index <= count; index++) {
4      console.log(index); // 1 2 3 4 5
5  }
```

Область видимости

```
1  const count = 0;
2
3  if (true) {
4      const innerCount = 1;
5
6      // Тут доступны обе переменные: count и innerCount
7      console.log(count); // 0
8      console.log(innerCount); // 1
9  }
10
11 // Тут доступна только одна переменные: count
12 console.log(count); // 0
13 console.log(innerCount);
```

Захват переменной

```
1  const createCacheable = function(heavyFunction) {
2      let result;
3
4      return function() {
5          if (!result) {
6              result = heavyFunction();
7          }
8          return result;
9      }
10 };
11 const myHeavyFunction = function() {
12     console.log("Вычисление");
13     return 12;
14 }
15 const cacheable = createCacheable(myHeavyFunction);
16 console.log(cacheable()); // "Вычисление" 12
17 console.log(cacheable()); // 12
18 console.log(cacheable()); // 12
```

```
1  // Вводная
2  let beginning = [
3        "Почему JavaScript?",
4        "Инструменты",
5  ];
6
7  // Основы JavaScript
8  let mainPart = [
9        "Переменные",
10       "Типы данных",
11       "Операторы",
12       "Ошибки",
13       "Управляющие конструкции"
14 ];
15
16 // Рефлексия
17 let finish = [
18       "Рефлексия"
19 ];
```

Рефлексия

- [x] Переменные
- [x] Типы данных
- [x] Операторы
- [x] Ошибки
- [x] Инструкции


Спасибо

Backup slides

Объявление и присваивание - const

```
1  // Объявление
2  const login;
3
4  // Присваивание
5  login = "mylogin";
6
7  // Чтение
8  console.log(login);
9
10 // Объявление и присваивание
11 const username = "mylogin";
12
```

Uncaught SyntaxError: Missing initializer in const declaration.
at script.js:2:7

Uncaught TypeError: Assignment to constant variable.
at script.js:5:7

Преобразование строки в число

```
Number('100'); // 100
Number('100 чисел'); // NaN
Number('3.14'); // 3.14
Number("два 15"); // NaN
Number('123', 8); // 123
Number('FF', 16); // NaN

+'100'; // 100
+'100 чисел'; // 100
+'3.14'; // 3.14
+"два 15"; // NaN
+''; // 0
```

```
Number.parseInt('100'); // 100
Number.parseInt('100 чисел'); // 100
Number.parseInt('3.14'); // 3
Number.parseInt("два 15"); // NaN
Number.parseInt('123', 8); // 83
Number.parseInt('FF', 16); // 255

Number.parseFloat('100'); // 100
Number.parseFloat('100 чисел'); // 100
Number.parseFloat('3.14'); // 3.14
Number.parseFloat("два 15"); // NaN
Number.parseFloat('123', 8); // 83
Number.parseFloat('FF', 16); // 255
```

Операторы

- Количество операндов: 1, 2, 3
- Ассоциативность: левая, правая
- Приоритет

Оператор in

Возвращает true если свойство содержится в объекте или в цепочке прототипов. Бросает исключение при применении на примитиве

```
// Неправильно
var array = ["first", "second", "third"];
1 in array; // true
"first" in array; // false

// Встроенные типы
"PI" in Math; // true

// Пользовательские типы
var hero = { name: "Batman", age: undefined };
"name" in hero; // true
"age" in hero; // true

// Примитивы
"min" in 12; // TypeError
```

Оператор instanceof

Возвращает true если объект является экземпляром указанного класса

```
var simpleString = "Строка";
var objectString = new String("Строка-объект");
var myDate = new Date();
var myObject = {};

console.log(simpleString instanceof Object); // false

console.log(objectString instanceof String); // true
console.log(objectString instanceof Object); // true
console.log(objectString instanceof Date); // false

console.log(myObject instanceof Object); // true

console.log(myDate instanceof Date); // true
console.log(myDate instanceof Object); // true
console.log(myDate instanceof String); // false
```