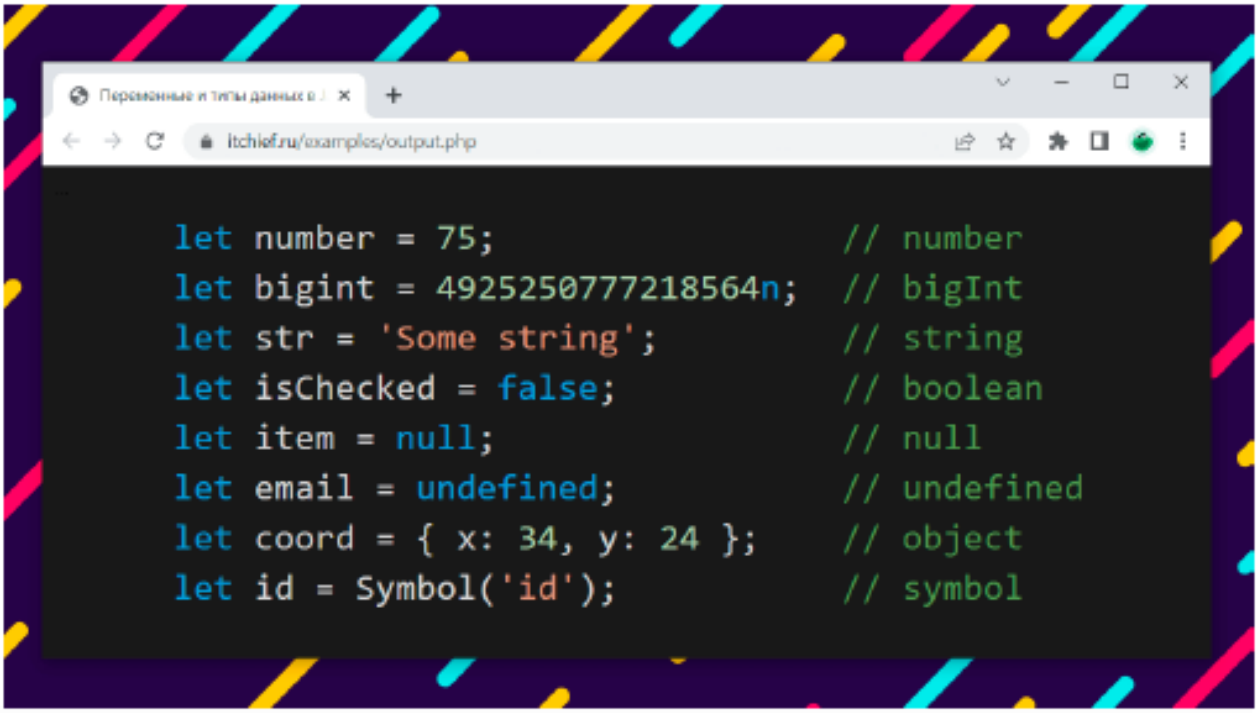
Лабораторная работа №2. Выражения, переменные и типы данных в JavaScript



Содержание:

На этом занятие мы напишем первую простую программу на JavaScript, разберём из чего она состоит и как работает. После чего изучим выражения, переменные, ключевые слова для их объявления, типы данных, что такое динамическая типизация и что делает оператор `typeof`.

Первая программа

Изучение JavaScript начнём с создания простой программы, состоящей из одной строчки, которая будет выводить в консоль браузера сообщение «Hello, World!»:

```
JavaScript

console.log('Hello, World!');
```

Полный текст HTML документа:

HTML

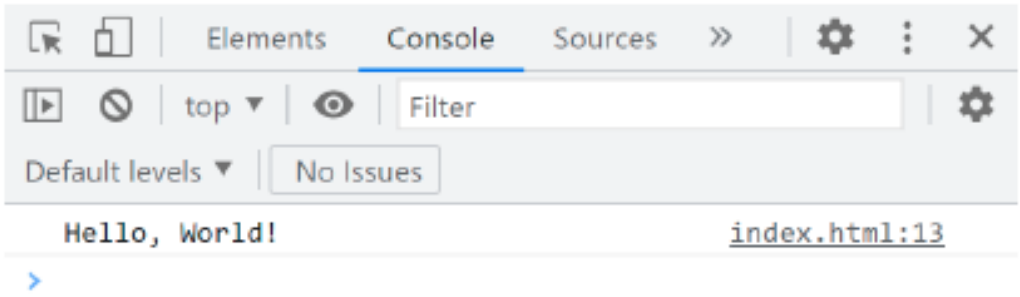
```
<!doctype html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>

    <!-- Сценарий JavaScript -->
    <script>
      console.log('Hello, World!');
    </script>
  </body>
</html>
```

Для его создания можно воспользоваться любым текстовым редактором, например, Visual Studio Code.

Для этого откройте текстовый редактор, вставьте в него этот код и сохраните его в файл с расширением html (например, «index.html»).

После этого откройте его в браузере, например, Google Chrome. Далее перейдите в «Инструменты разработчика», а в них на вкладку «Console»:



Здесь мы увидим вывод нашего сообщения «Hello, World!».

Программа на JavaScript находится между открывающим и закрывающим тегом `script`. Как уже было отмечено выше эта программа состоит всего из одной строчки кода:

JavaScript

```
console.log('Hello, World!');
```

Этот код выполняет очень простое действие: **выводит сообщение, указанное в круглых скобках в консоль**.

Чтобы понять как эта строчка работает, рассмотрим из чего она состоит:

- `console` – это **объект**. Объект в JavaScript – это набор свойств. Каждое свойство состоит из имени и значения («имя: значение»), имена также ещё называют ключами.
- `log` – это одно из свойств объекта `console`, а точнее его **метод**. Т.к. в качестве его значения выступает **функция**.
- `.` (точка) – это оператор, который мы используем **для доступа** к свойствам и методам объекта. В данном случае мы с помощью точки получаем доступ к методу `log` объекта `console`.
- `()` (скобки) – это часть синтаксиса JavaScript, с помощью которого в данном случае мы **вызываем функцию** `log`, которая является методом объекта `console`.
- `'Hello, World!'` – аргумент типа String (строка), т.е. **значение, которое мы передаём в метод** `log()`.

Таким образом, эта строчка `console.log('Hello, World!');` вызывает метод `log()` и передаёт ему в качестве аргумента значение `'Hello, World!'`. `log()` – это метод объекта `console` и поэтому чтобы к нему обратиться мы используем точку.

В этом примере мы познакомились с одними из ключевых понятий JavaScript: **объектом и функцией**. Здесь объектом является `console`, а функцией - метод `log()`.

В заключении отметим, что строка `console.log('Hello, World!')` является **выражением**.

Выражения

Выражение – это тоже одно из ключевых понятий в JavaScript. Оно представляет собой некоторый кусок кода, который **всегда возвращает значение**.

Если вернутся к предыдущему примеру, то `console.log('Hello, World!')` как мы уже отметили выше является выражением, т.к. возвращает значение:

```
JavaScript

console.log('Hello, World!');
```

В качестве значения данное выражение возвращает `undefined`. В этом легко убедиться если его обернуть в ещё одну конструкцию `console.log()`:

```
JavaScript

console.log(console.log('Hello, World!')); // undefined
```

Примеры выражений:

- `'Alexander'` – это выражение, которое представляет собой строку; результатом этого выражения будет эта же строка;
- `7 + 3` – это выражение, в котором используется оператор `+`; результатом этого выражения будет число `10`;
- `'Alexander' + ' ' + 'Maltsev'` – результатом этого выражения будет новая строка `'Alexander Maltsev'`;
- `name = 'Tim'` – это выражение, в котором используется оператор присваивания; здесь переменной `name` присваивается значение `'Tim'`; результатом этого выражения будет строка `'Tim'`, т.е. то значение, которое мы присваиваем переменной;
- `index++` – здесь используется оператор `++`, который увеличивает значение переменной на 1; результатом этого выражения будет значение переменной `index`, увеличенное на 1;
- `2 > 3 || 2 > 1` – это выражение, в котором используются несколько операторов; оно возвращает значение `true`;
- `sum(3, 5)` – вызов функции также является выражением, т.к. функция всегда возвращает какое-либо значение.

Некоторые примеры этих выражений имеют *side effects* (побочные действия). Т.е. они не только возвращают значение, но выполняют также некоторые дополнительные действия.

Например, выражение `name = 'Tim'` не только возвращает значение `'Tim'`, но также выполняет присвоение переменной `name` значения `'Tim'`.

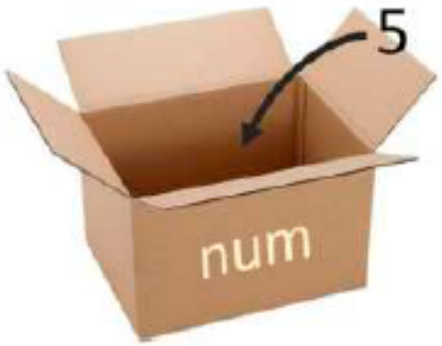
Если взять выражение `index++`, то оно тоже кроме возвращения значения ещё увеличивает значение переменной `index` на единицу.

Если внутри функции будут выполняться некоторые другие действия кроме возврата значения, то это выражение также будет иметь *side effects*.

Переменные

Переменная – это именованный участок памяти для хранения данных. Они используются когда необходимо сохранить некоторое значение, чтобы к нему можно было обратиться позже.

Представить переменную можно как коробку, на которой написано некоторое название (имя переменной). В коробку мы можем положить некоторое значение.



В дальнейшем мы можем обратиться к этому значению, в данном случае по имени `num` и выполнить с ним какие-либо действия. Затем мы можем поместить в переменную другое значение. Но в процессе выполнения программы переменная в определённый момент времени всегда имеет какое-то одно определённое значение.

Именование переменных

Имя переменной можно составлять из букв, цифр и символов `$` и `_`. При этом первый символ переменной не должен быть цифрой. Кроме этого, в качестве имени переменной нельзя использовать зарезервированные слова JavaScript.

```JavaScript
// юбъявление двух переменных: phone и message
let phone, message;
```

Регистр букв в имени переменной имеет значение. Т.е., например, `phone` и `Phone` – это две разные переменные.

Несмотря на то, что вы можете выбирать любые названия для переменных, хорошей практикой является придерживаться следующих рекомендаций при составлении имен:

- `PascalCase` – для именования типов и классов;
- `SELECTOR_ITEM` – для именования констант (значения которых известно до запуска программы и не меняются на протяжении её выполнения);
- `camelCase` – во всех остальных случаях.

`PascalCase` – имя переменной всегда начинается с заглавной буквы. Другие слова, которые являются частью названия переменной тоже начинаются с большой буквы. Не допускается пробелов, тире, т.е. всё пишется слитно.

JavaScript

```
class SimpleSlider { ... };
```

Именование констант (например, `SELECTOR_ITEM`) осуществляется прописными буквами и нижнего подчеркивания для разделения слов.

JavaScript

```
const SELECTOR_ITEM = '.slider';
```

`camelCase` – тоже самое что `PascalCase` за исключением того, что первая буква пишется в нижнем регистре.

JavaScript

```
const SELECTOR_ITEM = '.slider';
```

Кроме этого, желательно также давать именам четкие названия, чтобы было очень просто понять, что хранится в такой переменной.

Объявление переменных

В JavaScript доступно 3 ключевых слова, с помощью которых можно объявить переменные: `let`, `const` и `var`.

Ключевые слова `let` и `const` появились в языке с приходом ECMAScript 6, который вышел в 2015 году. `var` присутствовало в языке JavaScript с самого начала и до ECMAScript 6 использовалось только оно.

В настоящее время не рекомендуется использовать ключевое слово `var`, а только `let` и `const`.

Пример объявление переменной с названием `email`:

JavaScript

```
let email;
```

Перед названием переменной расположено ключевое слово `let`. Это слово дает инструкцию интерпретатору JavaScript создать новую переменную с именем email.

При создании переменной ей можно сразу же присвоить некоторое значение. Эту операцию называют **инициализацией**. Выполняется присвоение значения только что объявленной переменной с помощью оператора `=`.

```JavaScript
let email = 'no-reply@astr.org';
```

Присвоим переменной, объявленной до этого новое значение:

```JavaScript
email = 'support@astr.org';
```

Для того чтобы получить значение переменной к ней нужно просто обратиться по имени.

```JavaScript
// выведем в консоль браузера значение переменной email
console.log(email);
```

Переменная, которая объявлена без инициализации имеет по умолчанию значение `undefined`.

```JavaScript
let phone;
console.log(phone); // undefined
```

С помощью одного ключевого слова можно объявить сразу несколько переменных:

```JavaScript
let price = 78.55, quantity = 10, message;
```

Отделение друг от друга осуществляется посредством запятой.

Отличие let от const

Основное отличие `let` от `const` заключаются в том, что значения переменным, объявленных с помощью ключевого слова `let`, можно переприсваивать, т.е. задавать им новые значения.

```JavaScript
let price;
// присвоим переменной price новое значение
price = 76.10;
```

Если вы используете `const`, то необходимо при объявлении переменной сразу же присвоить ей значение:

JavaScript

```
const COLOR_RED = '#ff0000';
```

Если это не сделать, то получите ошибку:

JavaScript

```
const COLOR_RED; // Uncaught SyntaxError: Missing initializer in const declaration
COLOR_RED = '#ff0000';
```

При попытке присвоить новое значение, вы также получите ошибку:

JavaScript

```
const COLOR_RED = '#ff0000';
COLOR_RED = '#f44336'; // Uncaught TypeError: Assignment to constant variable.
```

Таким образом, переменной, объявленной с помощью `const` нельзя присваивать новые значения. Это основное отличие `let` от `const`.

Если в коде вы не планируете переменной присваивать новые значения, то тогда рекомендуется её объявлять с помощью `const`.

Также необходимо отметить, что если вы попытаетесь получить доступ к переменной, которая ещё не объявлена, то получите ошибку:

JavaScript Копировать

JavaScript

```
console.log(price); // Uncaught ReferenceError: Cannot access 'price' before initialization
let price;
```

Кроме этого, получите ошибку, если попытаетесь объявить переменную ещё один раз в её области видимости:

JavaScript

```
let price;
console.log(price);
let price = 76.10; // Uncaught SyntaxError: Identifier 'price' has already been declared
```

При работе с `const` имеется один очень интересный момент. Если присвоить переменной значение ссылочного типа, то вы не сможете изменить только саму ссылку на этот объект. Но при этом сам объект будет доступен для изменения.

JavaScript

```
// присвоим переменной массив (массив в JavaScript – это объект)
const colors = ['#ff0000', '#00ff00', '#00ff00'];
// присвоить новое значение переменной нельзя
colors = []; // Uncaught TypeError: Assignment to constant variable.
```

Изменить объект, ссылка на который содержится в переменной, объявленной с помощью `const` можно:

JavaScript

```
// присвоим переменной массив (массив в JavaScript – это объект)
const colors = ['#ff0000', '#00ff00', '#00ff00'];
// добавим в массив новое значение
colors.push('#4caf50');
// выведем значение COLORS в консоль
console.log(colors); // ["#ff0000", "#00ff00", "#00ff00", "#4caf50"]
```

Ключевое слово var

Как уже было отмечено выше, использовать `var` для объявления переменных сейчас не рекомендуется.

Отличия `var` от `let` и `const`:

1. `var` в отличие от `let` и `const` имеет функциональную область видимости, т.е. если её объявить внутри функции, то она будет видна во всем коде этой функции. При этом видимость `let` и `const` ограничена блоком, т.е. они видны только в пределах фигурных скобок `{ ... }`.

JavaScript

```
{
  var myAge = 36;
  let myName = 'John';
}
console.log(myAge); // 36
console.log(myName); // Uncaught ReferenceError: myName is not defined
```

2. Переменные, объявленные с помощью `var` поднимаются к началу текущего контекста. Называется это *hoisting*. К такой переменной можно обратиться до её объявления:

JavaScript

```
console.log(myAge); // undefined
var myAge = 36;
console.log(myAge); // 36
```

Кроме этого, мы даже можем присвоить значение переменной до её объявления:

JavaScript

```
myAge = 27;
console.log(myAge); // 27
var myAge = 36;
console.log(myAge); // 36
```

В случае с `let` или `const` такое не получится, т.к. её использовать можно только после объявления.

3. Если строгий режим не используется, то создать переменную с начальным значением можно без ключевого слова `var`:

JavaScript

```
// создадим переменную price со значением без использования var
price = 250;
console.log(price); // 250
```

Создавать переменные без `var` не рекомендуется.

В строгом режиме мы получим ошибку:

JavaScript

```
// создадим переменную price со значением без использования var
price = 250; // Uncaught ReferenceError: price is not defined
console.log(price);
```

Типы данных

Значение в JavaScript имеют определённый тип. Когда переменной мы присваиваем значение, она соответственно тоже принимает этот тип.

Все типы данных в JavaScript можно разделить **на примитивные и ссылочный**.

В JavaScript выделяют 7 примитивных типов:

- number (обычные числа);
- bigint (большие целые числа);

- string (строки);
- boolean (логический) – имеет всего два значения: `true` и `false`;
- null – имеет одно значения: `null`;
- undefined – имеет одно значения: `undefined`;
- symbol (символ) – для создания уникальных значений;

Когда вы присваиваете переменным значения примитивных типов, то они будут непосредственно содержать значения в памяти компьютера, т.е. явно.

```JavaScript
let coordX = 77;
let coordY = coordX;
coordX = 55;
console.log(coordX); // 55
console.log(coordY); // 77
```

В этом примере мы присвоили переменной `coordY` значение, находящееся в `coordX`. `coordY` будет после этого непосредственно содержать это значение в памяти.

Если мы переопределим значение переменной `coordX`, то значение `coordY` не изменится. Потому что значения примитивных типов хранятся непосредственно в переменных. Т.е. переменная `coordY` в таком случае получит собственную копию этого значения.

Ссылочный тип в JavaScript только один – это object (объект). Объект – это одно из ключевых понятий JavaScript. Т.к. в этом языке всё практически является объектами. Функция, массив, дата и т.д. – это объекты. Даже примитивные типы данных ведут себя как объекты, хотя ими не являются.

1. Object (объект)

Объект – это набор свойств «имя: значение». Имена часто также называют ключами.

Пример объекта записанного с помощью литерального синтаксиса, т.е. фигурных скобок:

JavaScript

```
const person = {
  firstName: 'Alexander',
  lastName: 'Maltsev',
  age: 37,
  pets: {
    dog: 'husky',
    cat: 'chartreux',
    fish: 'neon tetra'
  },
  getFullName: function () {
    return `${this.firstName} ${this.lastName}`
  }
}
```

В этом примере свойства `firstName`, `lastName` и `age` содержат значения примитивных типов данных. Значение свойства `pets` – объект. Такие объекты называются вложенными. У него содержатся свои свойства с соответствующими значениями.

Свойство `getFullName` содержит в качестве значения функцию. Такие свойства в JavaScript называются методами.

Когда мы присваиваем такой объект переменной, в ней будет храниться только ссылка (например, `0xD30F`) на этот объект. А сам объект будет хранится в области памяти на которую указывает эта ссылка.

Если некоторой переменной присвоить значение переменой `person`, то она будет тоже содержать ссылку на этот объект:

JavaScript

```
const myPerson = person;
```

Таким образом, у нас получатся две переменные, которые ссылаются на один и тот же объект.

Если добавим новое свойство в объект с помощью переменной `myPerson`, то мы увидим изменение в этом объекте и посредством переменной `person`, т.к. они указывают на один и тот же объект в памяти:

JavaScript

```
// добавим в объект новое свойство
myPerson.hobbies = ['ski', 'books', 'fitness'];
// выведем в консоль его значение с помощью переменной person
console.log(person.hobbies); // ['ski', 'books', 'fitness']
```

Для обращения к свойству `hobbies` в этом примере мы использовали точку. Кроме

точечной записи (dot notation), вы можете также работать со свойствами объектами используя квадратные скобки:

```JavaScript
console.log(person['hobbies']); // ['ski', 'books', 'fitness']
```

2. Number (число)

Числовой тип в JavaScript является универсальным. Он используется для представления как целых, так и чисел с плавающей точкой.

```JavaScript
let int = 5; // целое число
let float = 5.98; // число с плавающей точкой
```

Формат представления чисел в JavaScript осуществляется в соответствии со стандартом IEEE 754-2008.

Целые числа в JavaScript можно задавать не только в десятичной системе счисления, но и в **восьмеричной (0)** или **шестнадцатеричной системе счисления (0x)** с использованием префиксов, указанных в круглых скобках:

```JavaScript
let int = 010; // 8
int = 055; // 45
int = 0xFF; //255
int = 0xB8; // 184
```

Записывать числа также можно в **экспоненциальной форме**:

```JavaScript
let num = 2e3; // экспоненциальная запись числа 2*10^3 (2000)
num = 2e-3; // экспоненциальная запись числа 2*10^-3 (0,002)
num = 3.2e3; // 3200
num = 1.5e-2; // 0.015
```

Числовой тип данных кроме чисел содержит ещё **специальные числовые значения**:

- `Infinity` (положительная бесконечность);
- `-Infinity` (отрицательная бесконечность);
- `NaN` (Not a Number – не число).

Специальное значения `Infinity` означает очень большое положительное число, т.е. число, которое не может быть представлено в JavaScript по причине того, что оно слишком велико.

Специальные значения `-Infinity` означает, наоборот, очень большое отрицательное число, т.е. число, которое не может быть представлено JavaScript по причине того, что оно тоже слишком велико.

Пример выражений, в результате вычисления которых будет **возвращены специальные числовые значения**:

```JavaScript
5/0; // Infinity
-5/0; // -Infinity
Math.pow(10,399); // Infinity (10399)
Math.pow(10,399); // -Infinity (-10399)
```

Значение `NaN` возвращается в результате выполнения математических операций, которые JavaScript не может вычислить.

```JavaScript
5 - 'Hi'; // NaN (от числа 5 отнять строку)
1000 / '20px'; // NaN (число поделить на строку)
true * '1rem'; // NaN (логическое значение true умножить на строку)
```

При этом очень интересным является то, что значение `NaN` в JavaScript не равно ничему включая себя.

```JavaScript
NaN == NaN; // false
NaN === NaN; //false
```

3. BigInt (большое целое число)

BigInt – это числовой тип, который предоставляет возможность работать с большими целыми числами. Появился этот тип в спецификации ECMAScript 2020 (11 редакции).

Примитив `bigint` создается путём добавления `n` в конец числа:

```JavaScript
const bigint = 8950422584340004n;
console.log(bigint); // 8950422584340004n
```

4. String (строка)

String (строка) – это тип данных, который используется в JavaScript для представления текста.

Строка JavaScript может состоять из 0 или большего количества символов.

В качестве формата строки в JavaScript всегда используется кодировка Unicode.

Создание строки (литерала строки) выполняется посредством **заключения текста в одинарные или двойные кавычки**.

```JavaScript
let str1 = 'ECMAScript';
let str2 = "JavaScript";
```

В JavaScript нет разницы между одинарными и двойными кавычками.

Но, в некоторых случаях есть смысл использовать именно одинарные кавычки, а не двойные и наоборот.

Например, когда строка содержит двойные кавычки, её более удобно заключить в одинарные. Это избавит от необходимости экранирования в ней двойных кавычек.

```JavaScript
let str1 = '"ECMAScript"'; // без экранирования (с использованием одинарных кавычек)
let str2 = ""ECMAScript""; // с экранированием
```

Строка в JavaScript может содержать специальные символы. Например, (перевод строки), (табуляция), (возврат каретки) и др.

```JavaScript
 console.log('Это предложение,
А это тоже предложение, но оно будет выведено на новой строке.');
```

Со строками можно производить операцию сложения (объединения) или, другими словами, конкатенацию. Для этого используется оператор `+`. Смысл данной операции заключается в присоединении второй строки к концу первой.

```JavaScript
console.log('Я люблю ' + 'JavaScript'); // Я люблю JavaScript
```

5. Boolean (логический тип данных)

Boolean – примитивный тип данных, который имеет всего два значения: `true` (истина) и `false` (ложь).

```JavaScript
let isMoving = true;
let hasTransformed = false;
```

6. Тип null

null – тип данных, который имеет одно значение: `null`.

Он используется, когда вы хотите явно указать, что на данном этапе у переменной нет значения, т.е. оно отсутствует.

```JavaScript
let direction = null;
```

7. Тип undefined

undefined - тип данных, который имеет одно значение: `undefined`. Такое значение подразумевает, что значение у той или иной переменной нет, т.е. оно не определено.

Такое значение содержит объявленная переменная, которой ещё не присвоено значение.

```JavaScript
let num;
console.log(num); // undefined
```

Значение `undefined` также возвращается при доступе к свойству объекта, которого у него нет:

```JavaScript
const objA = {
  x: 5,
  y: 2
};
console.log(objA.z); // undefined
```

console.log(objA.z); // undefined

Также функция которая явно не возвращает значение:

JavaScript

```
function myFunc(str) {
  console.log(str);
}

const result = myFunc('JavaScript!');
console.log(result); // undefined
```

Аналогично будет использование оператора `return` без указания значения:

JavaScript

```
function myFunc() {
  return;
  console.log(str);
}

const result = myFunc();
console.log(result); /// undefined
```

Разница между `null` и `undefined` заключается в том, что значение `null` мы присваиваем переменной сами.

8. Symbol (символ)

Symbol (символ) предназначен для создания уникальных идентификаторов. Появился этот тип данных в ECMAScript 2015 (6 редакции). Создание символа выполняется с помощью функции `Symbol()`:

JavaScript

```
const id = Symbol('id')
```

Эта функция всегда возвращает уникальное значение символа. В круглых скобках при необходимости можно указать описание символа, которое также называется его именем. В этом примере в качестве него задана строка `'id'`.

Символы в основном применяются для создания скрытых свойств объекта, а также для изменения встроенного поведения объектов с использованием системных символов.

Динамическая типизация

JavaScript является языком с динамической типизацией. Это означает, что при объявлении переменной ей не нужно указывать тип данных, который она может принимать. Вы должны просто задекларировать переменную с помощью ключевого слова `let` или `const`.

Таким образом, вы сначала можете присвоить переменной значение с одним типом данных, а позже переприсвоить ей значение, имеющее другой тип данных.

Например, присвоим одной и той же переменной значения разных типов:

JavaScript

```
let output = 'успех'; // строка
output = 28; // число
output = true; // логический тип
```

Динамическая типизация имеет преимущества, так и недостатки:

JavaScript

```
// объявим функцию
function sum(a, b) {
  console.log(a + b);
}
// вызовем функцию
sum(2, 3); // 5
// присвоим переменной sum новое значение
sum = 4;
// вызовем функцию ещё раз
sum(3, 7); // Uncaught TypeError: sum is not a function
```

В этом примере мы объявили функцию с помощью ключевого слова `function`. `sum` - это название функции. Когда мы первый раз вызывали функцию, переменная `sum` содержала её. После этого мы присвоили переменной число `4`. Теперь когда попытаемся вызвать функцию ещё раз, мы получим ошибку. Т.к. переменная `sum` на данном этапе содержит просто число.

Чтобы таких ошибок было меньше в коде, рекомендуется где это возможно использовать ключевое слово `const`:

JavaScript

```
// присвоим переменной стрелочную функцию
const sum = (a, b) => {
  console.log(a + b);
}
// вызовем функцию
sum(2, 3); // 5
// присвоим переменной sum новое значение
sum = 4; // Uncaught TypeError: Assignment to constant variable.
// вызовем функцию ещё раз
sum(3, 7);
```

В этом примере функция также содержится в переменной, но в объявленной с

помощью `const`. А так как переменная объявлена с помощью `const`, то мы не можем ей присвоить новое значение, в данном случае число `4`.

Оператор typeof

Оператор `typeof` позволяет определить тип того или иного значения. При этом у него имеется два синтаксиса, с круглыми скобками и без них:

JavaScript

```
typeof operand
typeof(operand)
```

В качестве результата он возвращает строку, которая будет показывать тип этого значения:

JavaScript

```
let name;
const age = 37;
const email = 'v3@gmail.com';
const isLicense = true;
const interest = null;
const num = 3n;
const id = Symbol('id');
const lastExperience = {
  [id]: 37216,
  period: 'June 2011 - June 2018',
  place: 'ISACA, Moscow',
  position: 'Web designer'
};
const getExperience = () =gt; {
  return `${lastExperience.period} (${lastExperience.position} - ${lastExperience.place})`;
};

console.log(typeof name);           // undefined
console.log(typeof age);            // number
console.log(typeof email);          // string
console.log(typeof isLicense);      // boolean
console.log(typeof interest);       // object (1)
console.log(typeof num);            // bigint
console.log(typeof id);             // symbol
console.log(typeof lastExperience); // object
console.log(typeof getExperience);  // function (2)
```

В этом примере:

(1) - При получении типа данных для значения `null`, оператор `typeof` возвращает `object`. Это ошибка, которая присутствует в языке, начиная с его первой реализации. Она не была исправлена в целях сохранения

совместимости и это необходимо учитывать, при написании сценариев. null - это примитивный тип данных, он не является объектом.

(2) - Очень удобно, что `typeof` выделяет функции отдельно. Это позволяет нам очень просто проверить является ли указанная переменная функцией. Но функции в JavaScript являются объектами:

```JavaScript
console.log(getExperience instanceof Object); // true
```

Оператор `typeof` очень часто используется когда нужно узнать есть ли значение у переменной:

```JavaScript
let a;
if (typeof a === 'undefined') {
  console.log('У переменной a нет значения!');
} else {
  console.log(`Переменная a имеет значение: ${a}!`);
}
```

Также `typeof` применяется когда вы не уверены какой тип у переменной и вам нужно его проверить перед тем как выполнить какие-то действия:

```JavaScript
let sum = 0;
[1, 3, 'a', null].forEach((item) => {
  if (typeof item === 'number') {
    sum += item;
  }
});
console.log(sum); // 4
```

В этом примере мы будем прибавлять к `sum` значение переменной `item` только если она является числом, т.е. имеет тип `number`.