Конспект. Переменные и типы данных

В конспекте мы повторяем всю теоретическую информацию из урока.

Добавление JS на страницу

Добавить код JavaScript на HTML-страницу можно двумя способами.

Код JS в HTML-файле

Код может находиться прямо в HTML файле между тегами `<script></script>`.

Рекомендуем помещать эти теги перед закрывающим тегом `</body>`, чтобы убедиться, что весь HTML загружен перед выполнением скрипта.

Это самый простой вариант, который редко используется для больших скриптов, потому что превращает HTML файл в огромную трудночитаемую простыню кода.

Пример такой вставки:

```
<!DOCTYPE html>
<html>
<head>
   <title>Пример страницы</title>
</head>
<body>
   ...
   <script>
      alert('Привет, мир!');
   </script>
</body>
</html>
```

Код JS во внешнем файле

В этом варианте код располагается в отдельном файле с расширением .js, а внутри тегов `<script></script>` находится ссылка на этот файл.

У тега `<script>` в этом варианте добавляется атрибут `src`.

Пример содержимого файла `script.js`:

```
alert('Привет, мир!');
```

Как подключить в HTML:

```
<!DOCTYPE html>
<html>
<head>
   <title>Пример страницы</title>
</head>
<body>
   ...
   <script src="script.js"></script>
</body>
</html>
```

Всплывающие подсказки в JS

JavaScript предлагает несколько встроенных функций для взаимодействия с пользователем:

1. **Alert**: показывает сообщение и ждёт, пока пользователь нажмёт "ОК".

   ```
   alert("Это простое сообщение.");
   ```

2. **Confirm**: показывает сообщение и ждёт, пока пользователь нажмёт "ОК" или "Отмена". Возвращает `true` (если нажато "ОК") или `false` (если нажата "Отмена").

   ```
   if (confirm("Вы уверены?")) {
      // Пользователь нажал ОК
   } else {
      // Пользователь нажал Отмена
   }
   ```

3. **Prompt**: показывает сообщение с полем для ввода текста и кнопками "ОК" и "Отмена". Возвращает текст, введённый пользователем, или `null`, если была нажата "Отмена".

```
let age = prompt("Сколько вам лет?", 18);
```

Переменные

Представьте контейнер. В нем можно хранить самые разные вещи — яблоки, одежду или игрушки. И еще у него есть определенный объем, который он занимает на полке.

Переменные — как такой вот контейнер. Они могут хранить различные данные и занимают в Java Script определенный объем памяти.

Переменная — это хранилище простых данных.

В переменную мы можем положить возраст пользователя, название товара или его цену.

Создание переменной

Чтобы создавать переменные, в JS существует три ключевых слова: `var`, `let` и `const`.

Var — это старый способ объявление переменной. Сегодня мы поговорим только о `let` и `const`, как более современных и безопасных способах.

Чтобы создать переменную, нужно написать ключевое слово (`let` или `const`), затем название переменной. После этого можно присвоить ей значение через знак равенства.

```
let product = "Кресло";
const productPrice = 3000;
```

Имя переменной может быть любым: главное делать его понятным. В большинстве случаев названия переменных пишутся на английском языке с соблюдением принципа **camelCase**.

camelCase означает, что, если название переменной состоит из нескольких слов (product и price) то они пишутся слитно, причем второе и последующие слова начинаются с большой буквы: `productPrice`.

Принцип **camelCase** (с англ. «верблюжий регистр») получил такое название, потому что заглавная буква второго и следующих слов напоминает горб (или горбы) верблюда.

Ключевое слово `let` мы используем, когда знаем, что значение переменной может меняться, а const — когда нам нужно зафиксировать значение так, чтобы оно не менялось.

Например, для даты рождения пользователя лучше всего использовать `const`, а вот для его возраста — `let`. Возраст меняется с течением времени, а день рождения всегда один и тот же.

Комбинируем объявление переменной и вызов подсказки

Давайте спросим у пользователя нашей программы возраст, а затем выведем его:

```
let age = prompt("Сколько вам лет?"); // Запрашиваем возраст
alert(age); // Выводим возраст
```

В коде выше сначала у нас идет строка, где мы объявляем переменную age и присваиваем ей значение, которое вводит пользователь при помощи функции prompt. Затем это значение выводится при помощи функции alert.

Если вы хотите уточнить или переопределить значение переменной, вы просто обращаетесь к уже объявленной переменной по имени и присваиваете ей новое значение. Например:

```
// Переопределяем значение переменной
age = prompt("Укажите ваш возраст еще раз");
alert(age);
```

Типы данных

JavaScript — это язык с динамической типизацией. Типы данных переменных определяются автоматически во время выполнения программы. JavaScript сам анализирует и определяет тип данных на основе присваиваемого значения.

> Всего в JS **восемь** типов данных, мы поговорим об основных **шести.**

JavaScript отличается тем, что переменные могут хранить данные различных типов, и важно правильно понимать, как с ними работать.

Number

В этом типе данных числа — как целые, так и с плавающей точкой (например, 5 или 3.14).

Есть подтипы: `Infinity` представляет бесконечность, а `NaN` (Not a Number) используется, когда результат операции не является числом.

```
let price = 5.99; // Number
let infinity = 1 / 0; // Infinity
let notANumber = "текст" / 2; // NaN
```

BigInt

Это тип данных для очень больших чисел, которые выходят за рамки обычного типа Number.

```
let bigNumber = 1234567890123456789012345678901234567890n;
```

String

Текстовые данные. Могут быть заключены в одинарные, двойные или обратные кавычки.

```
let greeting = "Привет"; // Обычная строка
let name = 'Мир'; // Строка в одинарных кавычках
let phrase = `Привет, ${name}`; // Шаблонная строка
```

Boolean

Логический тип данных, который может быть либо `true` (истина), либо `false` (ложь).

```
let isLearning = true; // Boolean
```

Null

Специальное значение, которое представляет собой "ничего" или "пустое значение".

```
let emptyBox = null;
```

Undefined

Значение, автоматически присваиваемое переменным, которым не было присвоено какое-либо значение.

```
let unknownValue;
console.log(unknownValue); // undefined
```

Преобразование типов

В JavaScript можно изменять тип данных одной переменной на другой.

> Например: все данные, полученные от пользователя через текстовые поля ввода в форме (например, HTML формы), по умолчанию являются **строками.** Если эти данные — на самом деле числа, которые мы хотим использовать в математических вычислениях, нужно их преобразовать из строк в числа.

Преобразование в строку:

```
let value = String(123); // Становится "123" (строка)
```

Преобразование в число:

```
let numberFromString = Number("123"); // Становится 123 (число)
```

Преобразование в логическое значение:

```
let truthyValue = Boolean(1); // true
let falsyValue = Boolean(0); // false
```

Полезные замечания

- Пустые строки (`""`), ноль (`0`), `null`, `undefined`, и `NaN` в логическом контексте будут преобразованы в `false`.
- Почти любое другое значение (включая все объекты и массивы) преобразуется в `true`.

> JavaScript поддерживает динамическую типизацию. Тип переменной может быть автоматически изменен во время выполнения программы.

Ссылочные переменные

В JS всего один ссылочный тип данных — **Object**, или объект. Объекты в JS используются часто — это основа языка.

> **Объект** — это самостоятельная единица, имеющая свойства и определенный тип.

Объекты в JavaScript похожи на объекты из реальной жизни. Так же как у обычной тарелки есть цвет, форма, вес и материал, из которого она сделана — так и у объектов в JavaScript есть свойства.

Объект может быть создан с помощью фигурных скобок `{…}` с необязательным списком *свойств*.

Свойство — это пара «ключ : значение», где ключ — это строка (также называемая «именем свойства»), а значение может быть чем угодно.

Свойством объекта может быть переменная, закрепленная за объектом. Свойства объекта будут определять его характеристики.

В реальной жизни у собаки есть кличка, порода, вес, возраст и т.п. У объекта `dog` будут характеристики в виде переменных `name`, `breed`, `weight`, `age`.

```
const dog = {
   name: 'Шарик',
   breed: 'корги',
   weight: 6,
   age: 2
}
```

Такой способ объявления объекта называется **литеральным синтаксисом**. У каждого свойства есть ключ (также называемый «имя» или «идентификатор»). После имени свойства следует двоеточие `:`. Затем указывается значение свойства. Если в объекте несколько свойств, то они перечисляются через запятую.

Обращение к свойствам объекта

Мы можем прочесть свойства объекта в любой момент. Для обращения к свойствам они упоминаются вместе с именем объекта — через точку (`dog.name`, например):

```
console.log(dog.name); // 'Шарик'
console.log(dog.age); // 2
```

Добавить свойство также можно в любой момент. Например, давайте добавим свойство `isPedigree` (имеет родословную):

```
dog.isPedigree = true;
```

Для удаления свойства используется оператор `delete`:

```
delete dog.weight;
```

Обращаться к свойствам объекта можно не только через точку, но и при помощи квадратных скобок:

```
console.log(dog['name']); // 'Шарик'
console.log(dog['age']); // 2
dog['isPedigree'] = true;
delete dog['weight'];
```

Обращение к свойствам с помощью квадратных скобок приведет к таким же результатам, что и обращение через точку.

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной и зависеть от пользовательского ввода:

```
const user = {
   name: 'Bob',
   age: 32
};

let info = prompt("Что вы хотите узнать о пользователе?", "name");

console.log(user[info]);
// Если пользователь введет name, то получим 'Bob'
// Если age, то 32, во всех остальных случаях undefined
```

Стоит отметить, что в объекте ключи могут состоять из 2 и более слов, тогда такие ключи необходимо брать в кавычки:

```
const user = {
   name: 'Bob',
   age: 32,
   "likes dogs": true
};
```

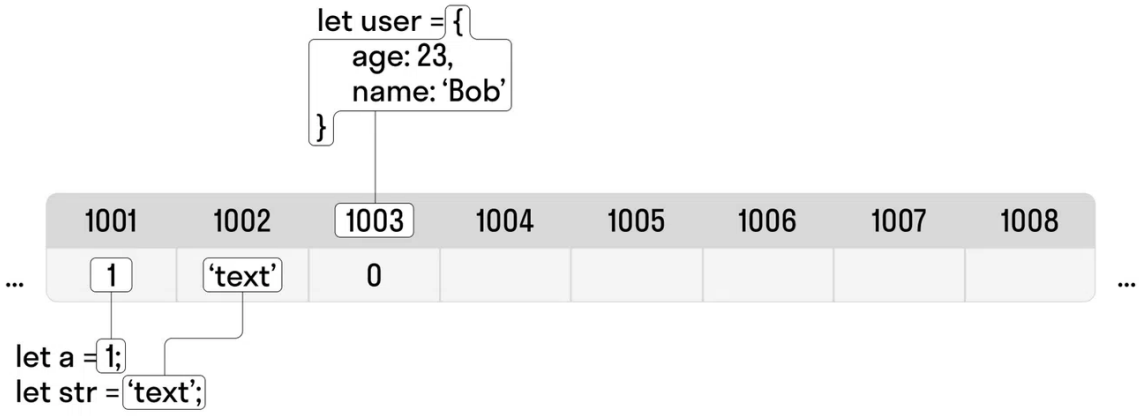
В этом случае обратиться к свойству через точку не удастся, нужно обращаться к свойству, используя квадратные скобки.

```
console.log(user.likes dogs); // Вызовет синтаксическую ошибку
```

```
console.log(user['likes dogs']); // true
```

Отличие примитивных типов данных от ссылочных

Примитивные данные хранят в себе конкретное значение из памяти. Ссылочные — ссылку на ячейку памяти.



Память — это ячейки, которые имеют свои адреса (…, 1001, 1002, 1003, …). При создании переменной конкретное значение записывается в свободную ячейку.

Примитивными типы переменных хранят в себе значение из ячейки памяти.

Переменные-свойства объекта хранят ссылку/индекс/адрес ячейки. Для каждого свойства в объекте необходима своя ячейка памяти.

Базовые операторы

Многие операторы вам уже известны и будут понятны интуитивно — на основе имеющегося знания математики.

Операторы — это символы или команды, которые передают компьютеру указания, например: сложить числа или присвоить некое значение.

Арифметические операторы

- **Сложение (+)**:

```
let sum = 5 + 3; // 8
```

- **Вычитание (-)**:

```
let difference = 5 - 3; // 2
```

- **Умножение (*)**:

```
let product = 5 * 3; // 15
```

- **Деление (/)**:

```
let quotient = 6 / 3; // 2
```

- **Модуль от деления (%)** (остаток от деления):

```
let remainder = 5 % 3; // 2
```

- **Показатель степени ()**

```
let power = 2 ** 3; // 8
```

Операторы инкремента и декремента

- **Инкремент (++): увеличение на единицу**:

```
let i = 1;
i++; // теперь i равно 2
```

- **Декремент (--): уменьшение на единицу**:

```
let j = 2;
j--; // Теперь j равно 1
```

Операторы присваивания

- **Простое присваивание (=)**:

```
let x = 5; // x теперь 5
```

- **Присваивание с операцией**:

- Сложение и присваивание (+=):

```
x += 4; // Эквивалентно x = x + 4;
```

- Вычитание и присваивание (-=):

```
x -= 2; // Эквивалентно x = x - 2;
```

- Умножение и присваивание (*=):

```
x *= 3; // Эквивалентно x = x * 3;
```

- Деление и присваивание (/=):

```
x /= 5; // Эквивалентно x = x / 5;
```

Понимание приоритетов операторов:

Так же, как и в арифметике, операторы имеют свой порядок действий. Например, умножение и деление имеют более высокий приоритет, чем сложение и вычитание. Если вам нужно изменить порядок выполнения операций, используйте скобки:

```
let result = (2 + 3) * 4; // 20, а не 14
```