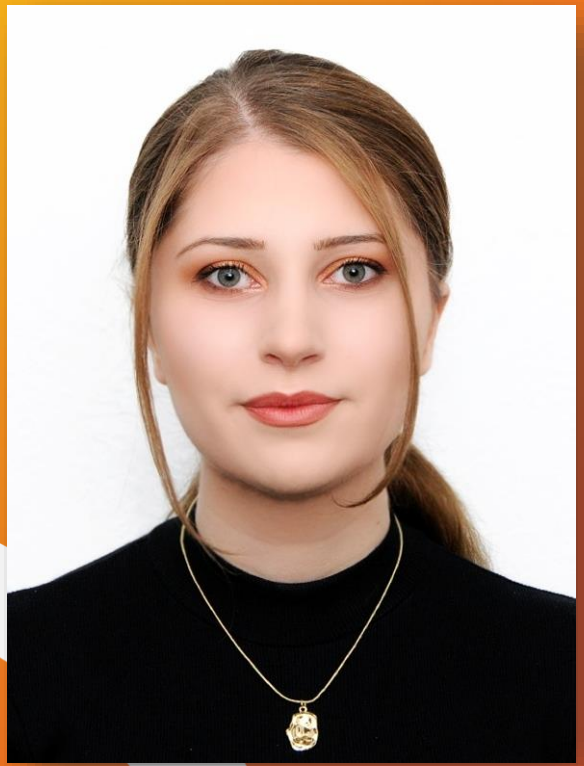
TEL-RAN
by Starta Institute

Frontend Basic
LESSON 9

Anna Khachaturyan



TEL-RAN
by Starta Institute

Анна Хачатурян

Front End/Gen Tech Teacher

- Since 2018 in IT
- Full Stack Developer at Web Magnat
- QA Engineer/Web Developer at Central Bank of RA
- Lecturer at Plekhanov Russian University of Economics
- TA at Picsart Academy
- Teacher at Tel-Ran

https://t.me/anny_khachaturyan

ВАЖНО:

- Если у Вас возник вопрос в процессе занятия, пожалуйста, поднимите руку и дождитесь, пока преподаватель закончит мысль и спросит Вас, также можно задать вопрос в чате или когда преподаватель скажет, что начался блок вопросов.
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях.
- Вести себя уважительно и этично по отношению к остальным участникам занятия.
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя.

TEL-RAN
by Starta Institute

ПЛАН ЗАНЯТИЯ

1. Повторение изученного
2. Вопросы по повторению – HTML/CSS
3. Введение в JavaScript
4. Практика

TEL-RAN
by Starta Institute

1 ПОВТОРЕНИЕ

Повторение

TEL-RAN
by Starta Institute

- **HTML** - это язык гипертекстовой разметки текста. Он нужен, чтобы размещать на веб-странице элементы: текст, картинки, таблицы и видео.
- **Теги** - базовые элементы HTML-разметки. Определяют, как браузеры отображают содержимое страницы, определяют формат и назначение содержимого. Бывают двойные (парные) и одинарные.
- **Атрибут** - дополнительная характеристика тега. Используется внутри открывающего тега для управления поведением элемента.
- **CSS** - язык стилей, который позволяет применять стиль (например, шрифты и цвет) к элементам HTML-документам. CSS отвечает за то, как выглядят элементы на странице.
- **Основные теги**: h1-h6, p, pre, span, br

Повторение

TEL-RAN
by Starta Institute

- Выделяют две основные категории HTML-элементов, которые соответствуют типам их содержимого и поведению в структуре веб-страницы — **блочные** и **строчные** элементы.
- Тег ** ** в HTML является **блочным** элементом верстки. Используется как **контейнер** для содержимого. С помощью тега обычно создается каркас и внутренняя структура страницы.
- Свойства: **margin, padding, border, text-align, line-height, font-weight, font-style**

Повторение

TEL-RAN
by Starta Institute

- **HTML-списки** используются для группировки связанных между собой фрагментов информации. Существует три вида списков:

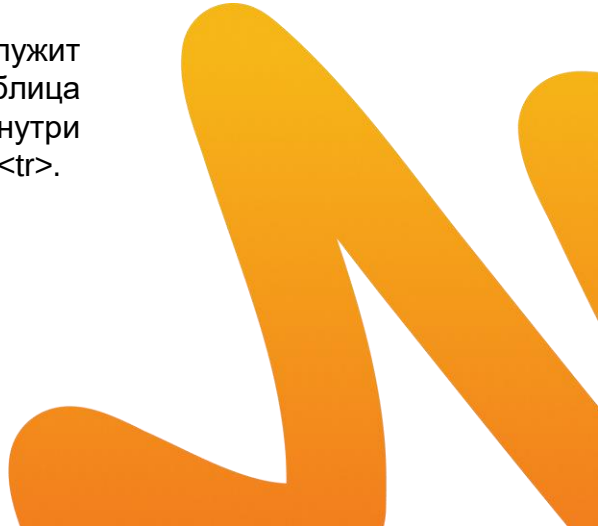
  - **маркированный список** — — каждый элемент списка <li> отмечается маркером,
  - **нумерованный список** — — каждый элемент списка <li> отмечается цифрой,
  - **список определений** — <dl> — состоит из пар термин <dt> — <dd> определение.

  Каждый список представляет собой контейнер, внутри которого располагаются элементы списка или пары термин- определение.

- **Таблица** создаётся при помощи элемента <table></table>, элемент <table> служит контейнером для элементов, определяющих содержимое таблицы. Любая таблица состоит из **строк** и **ячеек**, которые задаются с помощью тегов <tr> и <td>. Внутри <table> допустимо использовать следующие элементы: <caption>, <td>, <th>, и <tr>.

- Объединить ячейки в таблице можно с помощью атрибутов **colspan** и **rowspan**

- Свойства: **list-style, border, border-radius, border-collapse**



Повторение

TEL-RAN
by Starta Institute

- Свойство **box-shadow** добавляет элементу одну или более теней.
- Свойство **box-sizing** применяется для изменения алгоритма расчета ширины и высоты элемента.
- Псевдокласс **hover** определяет стиль элемента при наведении на него курсора мыши, но при этом элемент еще не активирован, иными словами кнопка мыши не нажата.

Повторение

TEL-RAN
by Starta Institute

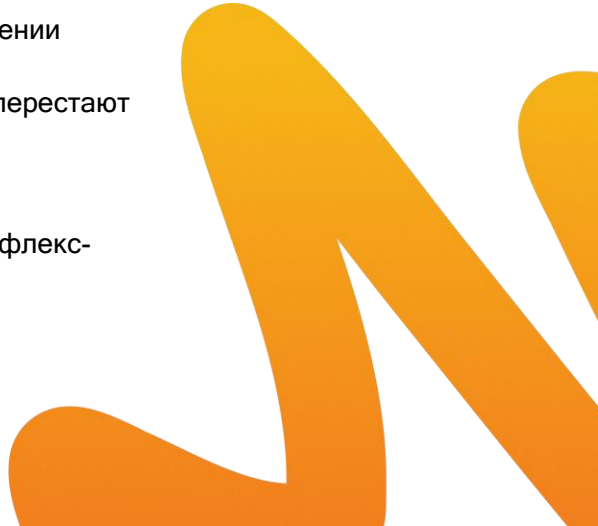
- **Display = block** будет занимать всю доступную ширину; Можно указать ширину и высоту явно.

  Блочные элементы располагаются один над другим, вертикально.
- **Display = inline** Элементы располагаются на той же строке, последовательно. Ширина и высота элемента определяются по содержимому. Поменять их нельзя.
- **Display = Inline-block** Это значение - означает элемент, который продолжает находиться в строке (inline), но при этом может иметь важные свойства блока
- **Display = none** Элемент полностью удален

Повторение

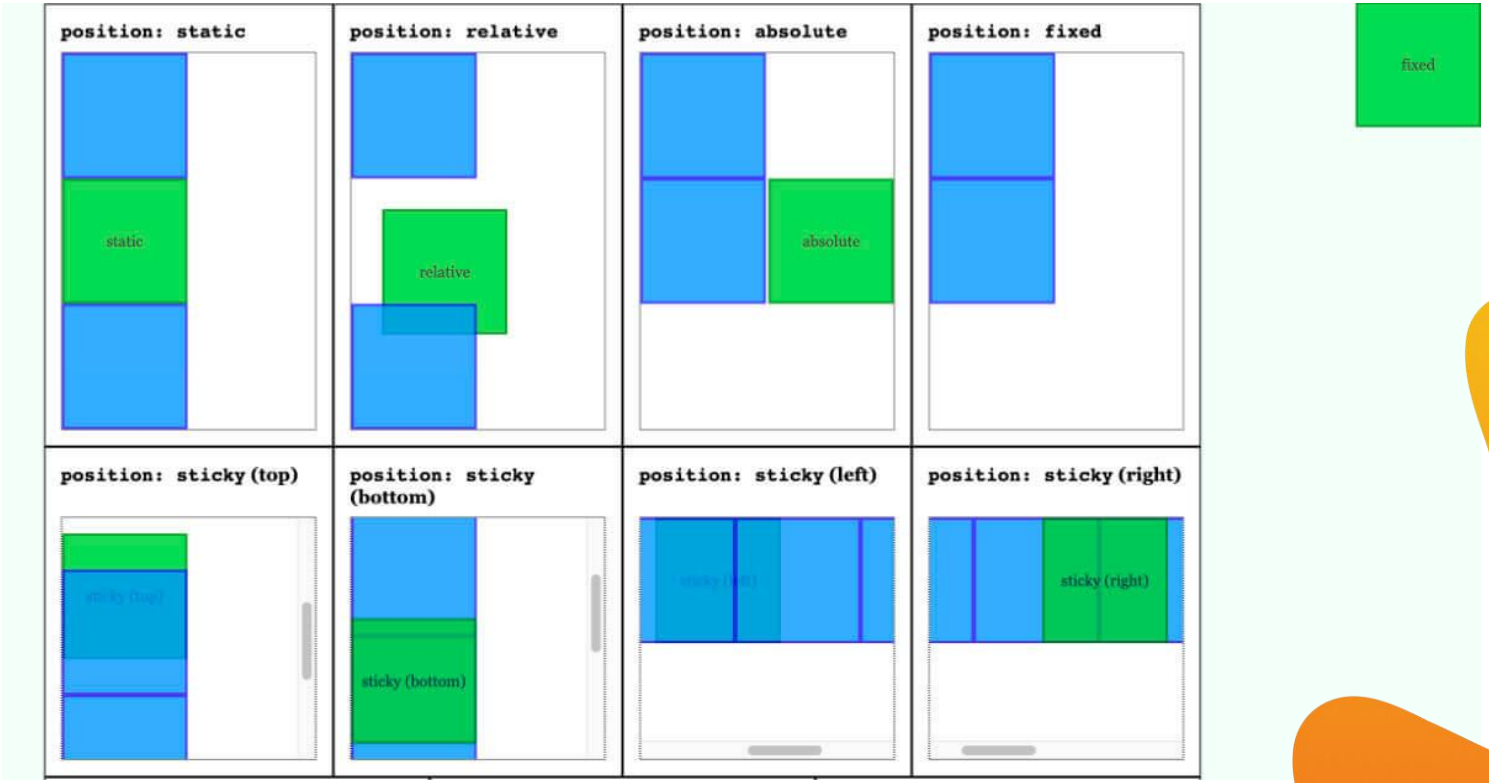
TEL-RAN
by Starta Institute

- **Flexbox** предоставляет инструменты для быстрого создания сложных, гибких макетов, и функции, которые были сложны в традиционных методах CSS.
- **flex-direction** - расставляет элементы в ряд или в колонку.
- **justify-content** - Отвечает за выравнивание элементов по главной оси.
- **align-items** - выравнивает флекс-элементы внутри контейнера в перпендикулярном направлении
- **flex-wrap** - Определяет смогут ли flex-элементы перемещаться на следующие строки, когда перестают помещаться внутри flex-контейнера
- **align-content** устанавливает тип выравнивания строк флекс-элементов по вертикали внутри флекс-контейнера, позволяя управлять свободным пространством.



Повторение

TEL-RAN
by Starta Institute



TEL-RAN
by Starta Institute

2 ВОПРОСЫ ПО ПОВТОРЕНИЮ

TEL-RAN
by Starta Institute

3 ВВЕДЕНИЕ В JAVASCRIPT

JavaScript это язык, который позволяет вам применять сложные вещи на web странице — каждый раз, когда на web странице происходит что- то большее, чем просто её статичное отображение — отображение периодически обновляемого контента, или интерактивных карт, или анимация 2D/3D графики, или прокрутка видео в проигрывателе, и т.д. — можете быть уверены, что скорее всего, не обошлось без JavaScript. Это третий слой слоёного пирога стандартных web технологий, два из которых (HTML и CSS).

Подключение JavaScript к HTML

Чтобы наша первая программа (или сценарий) JavaScript запустилась, ее нужно внедрить в HTML-документ.

Чтобы включить в HTML-документ JavaScript-код из внешнего файла, нужно использовать атрибут **src** (source) тега <**script**>. Его значением должен быть URL-адрес файла, в котором содержится JS-код:

```
    <script src="/scripts/script.js"></script>
```

В этом примере указан абсолютный путь к файлу с именем script.js, содержащему скрипт (из корня сайта). Сам файл должен содержать только JavaScript-код, который иначе располагался бы между тегами <script> и </script>.

Чтобы подключить несколько скриптов, используйте несколько тегов:

```
    <script src="/scripts/script1.js"></script>
    <script src="/scripts/script2.js"></script>
    ...
```

Выводим элемент в консоль

Консоль — это инструмент разработчика, который помогает тестировать код. Если во время выполнения скрипта возникнет ошибка, в консоли появится сообщение о ней. А ещё в консоль можно выводить текстовые подсказки.

Чтобы вывести сообщение в консоль, нужно использовать **console.log**:

```
console.log('Привет от JavaScript!');
```

В результате этой инструкции в консоли появится сообщение Привет от JavaScript!

Объявляем переменную

TEL-RAN by Starta Institute

Переменная — это способ сохранить данные, дав им понятное название.

В JavaScript переменную можно создать/объявить, с помощью ключевого слова **let**. За ним следует имя переменной. После объявления в переменную нужно записать, или присвоить, какое-то значение:

```
let variableName = 'Я значение переменной!';
```

Имя переменной может быть почти любым, но не должно начинаться с цифры, а из спецсимволов разрешены только '_' и '$'. Кроме того, в JavaScript есть зарезервированные слова, которые нельзя использовать для именования переменных. Имена переменных чувствительны к регистру: header, Header и HEADER — это разные переменные.

Можно использовать:

- ***CamelCase*** (с англ. — «ВерблюжийРегистр», также «ГорбатыйРегистр», «СтильВерблюда») — стиль написания составных слов, при котором несколько слов пишутся слитно без пробелов, при этом каждое слово внутри фразы пишется с прописной буквы.
- ***Snake case*** (snake_case) «Змеиный регистр» — заменяет пробелы на символ подчеркивания. Но самое главное — чтобы переменная действительно делала код понятнее, её имя должно описывать то, что в ней хранится.

let имеет блочную область видимости.

Блок — это фрагмент кода, ограниченный фигурными скобками {}. Всё, что находится внутри фигурных скобок, относится к блоку. Таким образом, переменная, объявленная в блоке через let, будет доступна только внутри этого блока.

Давайте рассмотрим на примере:

```
let greeting = "say Hi";
let times = 4;

if (times > 3) {
    let hello = "say Hello instead";
    console.log(hello);// "say Hello instead"
 }
console.log(hello) // hello is not defined
```

Здесь видно, что попытка использовать hello вне блока (фигурных скобок, в рамках которых переменная была определена) возвращает ошибку. Это происходит потому, что переменные типа let являются блочными.

Переменные let могут быть обновлены, но не объявлены повторно.

TEL-RAN
by Starta Institute

Переменные, объявленные через let, можно обновлять внутри их области видимости, но нельзя повторно объявить внутри области видимости. Так что, хотя такое сработает:

```
let greeting = "say Hi";
greeting = "say Hello instead";
```

Этот код уже вернет ошибку:

```
let greeting = "say Hi";
let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been declared
```

Однако, если переменная с одним и тем же именем определена в разных областях видимости, ошибки не будет:

```
let greeting = "say Hi";
   if (true) {
      let greeting = "say Hello instead";  console.log(greeting); // "say
      Hello instead"
   }
console.log(greeting); // "say Hi"
```

Почему же ошибки нет? Так происходит, потому что оба экземпляра рассматриваются как различные переменные, поскольку у них различаются области видимости.

Используя let, вам не нужно беспокоиться, не было ли у вас уже переменной с таким именем раньше, ведь переменная существует только внутри своей области видимости.

Объявление **const** задаёт константу, то есть переменную, которую нельзя менять:

```
const apple = 5;
apple = 10; // ошибка
```

В остальном объявление const полностью аналогично let.

Ввод данных

Функция prompt принимает два аргумента:

```
result = prompt(title, default);
```

Она выводит модальное окно с заголовком title, полем для ввода текста, заполненным строкой по умолчанию default и кнопками OK/CANCEL.

Пользователь должен либо что-то ввести и нажать OK, либо отменить ввод кликом на CANCEL или нажатием Esc на клавиатуре. Вызов prompt возвращает то, что ввёл посетитель – строку или специальное значение null, если ввод отменён.

Safari 5.1+ не возвращает null

Единственный браузер, который не возвращает null при отмене ввода – это Safari. При отсутствии ввода он возвращает пустую строку. Предположительно, это ошибка в браузере.

Пример:

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

console.log('Вам ' + age + ' лет!');
```

Типы данных

TEL-RAN
by Starta Institute

Число

Числовой тип данных (**number**) представляет как целочисленные значения, так и числа с плавающей точкой.

```
let n = 123;
n = 12.345;
```

Существует множество операций для чисел, например, умножение *, деление /, сложение +, вычитание - и так далее.

Кроме обычных чисел, существуют так называемые «специальные числовые значения», которые относятся к этому типу данных: **Infinity**, **-** Infinity и **NaN**.

Infinity представляет собой математическую бесконечность ∞. Это особое значение, которое больше любого числа. Мы можем получить его в результате деления на ноль:

```
console.log( 1 / 0 ); // Infinity

console.log( -1 / 0 ); // -Infinity
```

NaN означает вычислительную ошибку. Это результат неправильной или неопределённой математической операции, например:

```
console.log( "не число" / 2 ); // NaN, такое деление является ошибкой
```

Значение NaN «прилипчиво». Любая математическая операция с NaN возвращает NaN:

```
console.log( NaN + 1 ); // NaN
console.log( 3 * NaN ); // NaN  console.log( "не число" / 2 - 1 ); // NaN
```

Если где-то в математическом выражении есть NaN, то оно распространяется на весь результат (есть только одно исключение: NaN ** 0 равно 1).

Подробно: https://learn.javascript.ru/number

Строка

Строка (**string**) в JavaScript должна быть заключена в кавычки.

```
let str = "Привет";
let str2 = 'Одинарные кавычки тоже подойдут';
let phrase = `Обратные кавычки позволяют встраивать переменные ${str}`;
```

В JavaScript существует три типа кавычек.

- Двойные кавычки: "Привет".
- Одинарные кавычки: 'Привет'.
- Обратные кавычки: `Привет`.

Двойные или **одинарные** кавычки являются «простыми», между ними нет разницы в JavaScript.

Обратные же кавычки имеют расширенную функциональность. Они позволяют нам встраивать выражения в строку, заключая их в **${…}**.
Например:

```
let name = "Иван";
// Вставим переменную
console.log( `Привет, ${name}!` ); // Привет, Иван!
// Вставим выражение
console.log( `результат: ${1 + 2}` ); // результат: 3
```

Выражение внутри ${…} вычисляется, и его результат становится частью строки. Мы можем положить туда всё, что угодно: переменную name, или выражение 1 + 2, или что-то более сложное.

Обратите внимание, что это можно делать только в обратных кавычках. Другие кавычки не имеют такой функциональности встраивания!

```
console.log( "результат: ${1 + 2}" ); // результат: ${1 + 2} (двойные кавычки ничего не делают)
```

Подробно: https://learn.javascript.ru/string

Тип аргумента

Функция typeof возвращает тип аргумента. Это полезно, когда мы хотим обрабатывать значения различных типов по-разному или просто хотим сделать проверку.

Вызов typeof(x) возвращает строку с именем типа:

```
typeof(undefined) // "undefined"

typeof(0) // "number"  typeof(true) // "boolean"

typeof("text") // "string"
```

Базовые математические действия

В JavaScript между числами можно совершать различные математические операции:

```
console.log(2 + 3); //выведет 5
console.log(5 - 1); //выведет 4
console.log(2 * 3); //выведет 6
console.log(6 / 2); //выведет 3
console.log(5 ** 2); //выведет 25 - возведение числа в степень
console.log(11 % 2); //выведет 1 - остаток от деления
```

Конкатенация и интерполяция

TEL-RAN
by Starta Institute

Конкатенация

В веб-разработке программы постоянно оперируют строками. Всё, что мы видим на сайтах, так или иначе представлено в виде текста. Этот текст чаще всего динамический, то есть полученный из разных частей, которые соединяются вместе. Операция соединения строк в программировании называется конкатенацией.

```
// Оператор такой же, как и при сложении чисел
// но здесь он имеет другой смысл (семантику)
console.log('Dragon' + 'stone');
// => 'Dragonstone'
```

Склеивание строк всегда происходит в том же порядке, в котором записаны операнды. Левый операнд становится левой частью строки, а правый — правой.

```
// Конкатенировать можно абсолютно любые строки
console.log("King's" + 'Landing'); // => King'sLanding
```

Как видите, строки можно склеивать, даже если они записаны с разными кавычками.

В последнем примере название города получилось с ошибкой: King's Landing нужно писать через пробел. Но в наших начальных строках не было пробелов, а пробелы в самом коде слева и справа от символа + не имеют значения, потому что они не являются частью строк.

Выхода из этой ситуации два. →

Первый способ:

```
// Ставим пробел в левой части
console.log("King's " + 'Landing'); // => King's Landing
// Ставим пробел в правой части
console.log("King's" + ' Landing'); // => King's Landing
// Пробел — такой же символ, как и другие. Чем больше пробелов, тем шире отступы:
console.log("King's " + ' Landing'); // => King's    Landing

console.log("King's " + '              Landing'); // => King's     Landing
```

Второй способ:

```
console.log("King's" + " " + 'Landing'); //          => King's Landing
```

Интерполяция

Интерполяция - способ соединения строк через вставку значений переменных в строку-шаблон с помощью фигурных скобок. Например:

```
const firstName = 'Joffrey';
const greeting = 'Hello';
// Интерполяция не работает с одинарными и двойными кавычками
console.log(`${greeting}, ${firstName}!`);
// => 'Hello, Joffrey!'
```

Мы просто создали одну строку и «вставили» в неё в нужные места константы с помощью знака доллара и фигурных скобок ${ }. Получился как будто бланк, куда внесены нужные значения. И нам не нужно больше заботиться об отдельных строках для знаков препинания и пробелов — все эти символы просто записаны в этой строке-шаблоне.

Неявные и явные преобразования из строки в число и обратно

TEL-RAN
by Starta Institute

Система преобразования типов в JavaScript очень проста, но отличается от других языков. Всего есть три преобразования:

1.Строковое преобразование.
2.Численное преобразование.
3.Преобразование к логическому значению.

Строковое: **String(value)** – в строковом контексте или при сложении со строкой. Работает очевидным образом.

```
let num = 123;
let str = String(num);  console.log(str); //"123"
```

Численное: Number(value) – в численном контексте, включая унарный плюс +value. Происходит при сравнении разных типов, кроме строгого равенства.

```
let num1 = +"123"; // 123
let num2 = Number("456"); // 456
```

Что будет, если попробовать перемножить, к примеру, число и строку, вот так: 3 * '3'? В результате вы получите число 9. Это значит, что JavaScript автоматически осуществляет преобразование типов при необходимости, вам не нужно за это переживать.

Однако, есть нюанс: если мы попытаемся сложить строку и число, то JavaScript сложит их как строки, а не как числа, вот так: '3' + 3 получится строка '33', а не число 6. В случае, например, с умножением JavaScript понимал, что нельзя перемножить строки, поэтому строки переводил в числа и перемножал их. А случай со сложением можно трактовать двояко: складывать как строки или как числа (плюс-то используется как для сложения строк, так и чисел).

Бороться с этим можно следующем способом: нужно сделать недопустимую для строк операцию, например, так: +'3' + 3 - поставим плюс перед строкой, и она преобразуется к числу.

```
console.log('3' + 3); // '33'
console.log(+'2' + 2); // 4
console.log('9' / 3); // 3
console.log('3' * 4); // 12
console.log('6' - 2); // 4
```

TEL-RAN
by Starta Institute

4 ПРАКТИКА

Задача: Написать программу, которая считывает через prompt число и выводит в консоль ее квадрат

Задача: Написать программу, которая считывает два числа (объявляем две переменные и записываем туда результат работы двух вызовов prompt) и выводит их сумму. Не забудьте преобразовать полученные значения в число.

Ребята!
Вы
просто
молодцы!