

JS

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

Изучить, какие есть примитивные типы, операторы, условные операторы

ПЛАН ЗАНЯТИЯ

- Установка node.js
- Установка расширения для VSCode - code runner
- const, let
- Разница между строгими и нестрогими языками
- JS types
- typeof

ЗАГРУЗКА Node.js

Переходим по ссылке -
<https://nodejs.org/en/download>

Скачиваем

Проверка версии Node.js

`node -v`

Download Node.js®

Download Node.js the way you want.

Prebuilt Installer Prebuilt Binaries Package Manager Source Code

I want the version of Node.js for running

 **Download Node.js v20.12.1**

Node.js includes npm (10.5.0) ↗.

[Read the changelog for this version ↗](#)

[Read the blog post for this version ↗](#)

[Learn how to verify signed SHASUMS ↗](#)

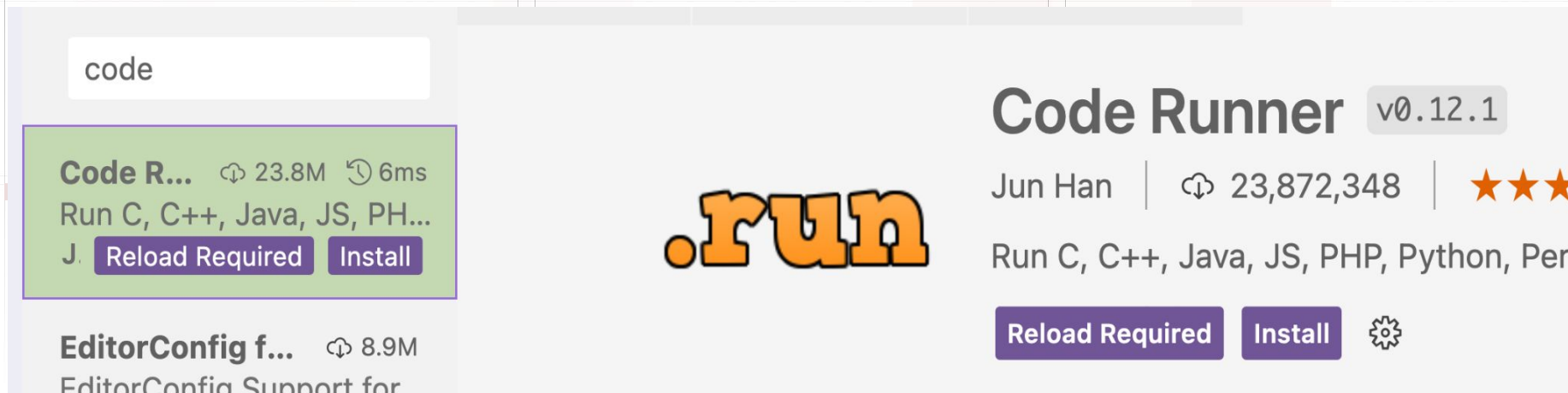
[Check out all available Node.js download options ↗](#)

Установка расширения Code Runner

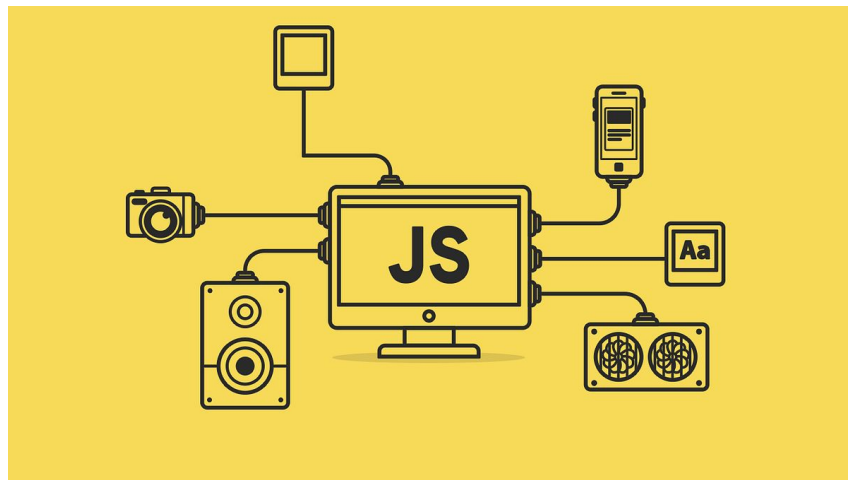
В левом меню VSCode вы можете увидеть значок “расширения”, он похож на кубик Рубика

Кликните по нему и введите в поисковой строке “Code Runner”

Нажмите на кнопку Install



Знакомство с JavaScript



Изначально JavaScript был создан, чтобы «сделать веб-страницы ЖИВЫМИ».

Программы на этом языке называются скриптами.

Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы.

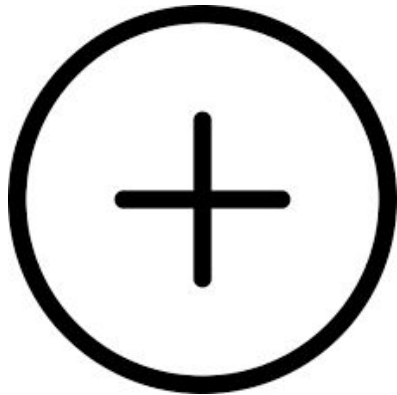


Что делает JavaScript особенным?

- Полная интеграция с HTML/CSS.
- Простые вещи делаются просто.
- Поддерживается всеми основными браузерами и включён по умолчанию.



Добавление JavaScript



Для добавления JavaScript в HTML, вы можно использовать тег <script>. Есть несколько способов включения JavaScript в HTML-документ.

1. Внутренний скрипт (в теле HTML):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript в HTML</title>
</head>
<body>

  <!-- Внутренний скрипт -->
  <script>
    // Ваш JavaScript код здесь
    console.log("Привет, это JavaScript внутри HTML!");
  </script>

</body>
</html>
```

2. Внешний файл скрипта (в конце тела HTML):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JavaScript в HTML</title>
</head>
<body>

  <!-- Содержимое тела документа -->

  <!-- Внешний файл скрипта в конце тела документа -->
  <script src="script.js"></script>
</body>
</html>
```

3. Внешний файл скрипта (в голове HTML):

Добавить в тег `<head></head>`

```
<script src="путь к вашему файлу с расширением js" defer></script>
```

Переменные



Переменные

JavaScript-приложению обычно нужно работать с информацией.

Например:

1. Интернет-магазин – информация может включать продаваемые товары и корзину покупок.
2. Чат – информация может включать пользователей, сообщения и многое другое.

Переменные используются для хранения этой информации.

Переменная – это «именованное хранилище» для данных. Мы можем использовать переменные для хранения товаров, посетителей и других данных.

Для создания переменной в JavaScript используйте ключевое слово `let`. Приведённая ниже инструкция создаёт (другими словами, объявляет) переменную с именем «message»:

```
1 let message;
```


Теперь можно поместить в неё данные (другими словами, определить переменную), используя оператор присваивания =:

```
1 let message;  
2  
3 message = 'Hello'; // сохранить строку 'Hello' в переменной с именем message
```

Строка сохраняется в области памяти, связанной с переменной. Мы можем получить к ней доступ, используя имя переменной:

```
1 let message;  
2 message = 'Hello!';  
3  
4 alert(message); // показывает содержимое переменной
```

Аналогия из жизни

Например, переменную `message` можно представить как коробку с названием `"message"` и значением `"Hello!"` внутри:

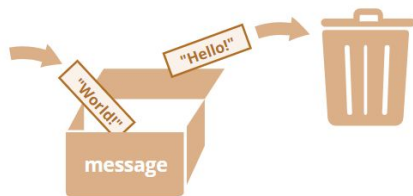
Мы можем положить любое значение в коробку.

Мы также можем изменить его столько раз, сколько захотим:



```
1 let message;  
2  
3 message = 'Hello!';  
4  
5 message = 'World!'; // значение изменено  
6  
7 alert(message);
```

При изменении значения старые данные удаляются из переменной:



Мы также можем объявить две переменные и скопировать данные из одной в другую.

```
1 let hello = 'Hello world!';  
2  
3 let message;  
4  
5 // копируем значение 'Hello world' из переменной hello в переменную message  
6 message = hello;  
7  
8 // теперь две переменные содержат одинаковые данные  
9 alert(hello); // Hello world!  
10 alert(message); // Hello world!
```

Имена переменных

В JavaScript есть два ограничения, касающиеся имён переменных:

1. Имя переменной должно содержать только буквы, цифры или символы \$ и _.
2. Первый символ не должен быть цифрой.

```
1 let userName;  
2 let test123;
```

```
1 let $ = 1; // объявили переменную с именем "$"  
2 let _ = 2; // а теперь переменную с именем "_"  
3  
4 alert($ + _); // 3
```

Константы

Чтобы объявить константную, то есть, неизменяемую переменную, используйте `const` вместо `let`:

```
1  const myBirthday = '18.04.1982';
```

Переменные, объявленные с помощью `const`, называются «константами». Их нельзя изменить. Попытка сделать это приведёт к ошибке:

```
1  const myBirthday = '18.04.1982';  
2  
3  myBirthday = '01.01.2001'; // ошибка, константу нельзя перезаписать!
```

Константы в верхнем регистре

Широко распространена практика использования констант в качестве псевдонимов для трудно запоминаемых значений, которые известны до начала исполнения скрипта.

Названия таких констант пишутся с использованием заглавных букв и подчёркивания.

Например, сделаем константы для различных цветов в «шестнадцатеричном формате»:

```
1  const COLOR_RED = "#F00";
2  const COLOR_GREEN = "#0F0";
3  const COLOR_BLUE = "#00F";
4  const COLOR_ORANGE = "#FF7F00";
5
6  // ...когда нам нужно выбрать цвет
7  let color = COLOR_ORANGE;
8  alert(color); // #FF7F00
```

Типы данных



Значение в JavaScript всегда относится к данным определённого типа. Например, это может быть строка или число.

Есть восемь типов данных в JavaScript.

Переменная в JavaScript может содержать любые данные. В один момент там может быть строка, а в другой – число:

```
1 // Не будет ошибкой
2 let message = "hello";
3 message = 123456;
```

1. Число

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

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

```
1 let n = 123;  
2 n = 12.345;
```

Число

- **Infinity** представляет собой математическую бесконечность ∞ . Это особое значение, которое больше любого числа.

Мы можем получить его в результате деления на ноль:

```
1 alert( 1 / 0 ); // Infinity
```

Или задать его явно:

```
1 alert( Infinity ); // Infinity
```

Число

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

```
1 alert( "не число" / 2 ); // NaN, такое деление является ошибкой
```

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

```
1 alert( NaN + 1 ); // NaN
2 alert( 3 * NaN ); // NaN
3 alert( "не число" / 2 - 1 ); // NaN
```

2. BigInt

В JavaScript тип `number` не может безопасно работать с числами, большими, чем 9007199254740991 или меньшими, чем -9007199254740991 для отрицательных чисел.

Чтобы создать значение типа **BigInt**, необходимо добавить `n` в конец числового литерала:

```
1 // символ "n" в конце означает, что это BigInt
2 const bigInt = 1234567890123456789012345678901234567890n;
```

3. Строка

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

```
1 let str = "Привет";  
2 let str2 = 'Одинарные кавычки тоже подойдут';
```

4. Булевый (логический) тип

Булевый тип (**boolean**) может принимать только два значения: **true** (истина) и **false** (ложь).

Такой тип, как правило, используется для хранения значений да/нет: **true** значит «да, правильно», а **false** значит «нет, не правильно».

```
1 let nameFieldChecked = true; // да, поле отмечено
2 let ageFieldChecked = false; // нет, поле не отмечено
```

Булевы значения также могут быть результатом сравнений:

```
1 let isGreater = 4 > 1;
2
3 alert( isGreater ); // true (результатом сравнения будет "да")
```

5. Значение «null»

Специальное значение **null** не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение **null**:

```
1 let age = null;
```

В JavaScript null не является «ссылкой на несуществующий объект» или «нулевым указателем», как в некоторых других языках. Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

6. Значение «undefined»

Специальное значение **undefined** также стоит особняком. Оно формирует тип из самого себя так же, как и null.

Оно означает, что «значение не было присвоено».

Если переменная объявлена, но ей не присвоено никакого значения, то её значением будет **undefined**:

```
1  let age;  
2  
3  alert(age); // выведет "undefined"
```

7. Символы

Символ (**Symbol**) в JavaScript является уникальным и неизменным примитивным типом данных. Каждый созданный символ уникален, и он может быть использован в качестве уникального идентификатора для свойств объекта. Создание символа происходит с использованием функции `Symbol()`.

```
// Создание символа
const mySymbol = Symbol();

// Использование символа в качестве ключа для свойства объекта
const obj = {
  [mySymbol]: 'Hello, Symbol!'
};

console.log(obj[mySymbol]); // Hello, Symbol!
```

8. Объекты

Следующий тип данных - **object**, он используется для более сложных структур данных.

В JavaScript объекты представляют собой коллекции ключ-значение, где ключи являются строками или символами, а значениями могут быть любые типы данных, они относятся к типу данных - **Object**.

```
const person = {  
  name: 'John',  
  age: 30,  
  gender: 'male'  
};
```

Функции и массивы также относят к сложному типу данных - **Object**

Оператор **typeof**:

Оператор **typeof** используется для определения типа значения.

<code>typeof 42;</code>	<code>//</code>	<code>"number"</code>
<code>typeof "Hello";</code>	<code>//</code>	<code>"string"</code>
<code>typeof true;</code>	<code>//</code>	<code>"boolean"</code>
<code>typeof undefined;</code>	<code>//</code>	<code>"undefined"</code>



Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH