

JS Level 3

Node.js



Node.js

Эта лекция может вам показаться достаточно простой, в чём-то даже "очевидной".

Она является вводной в курс и содержит ряд ключевых идей, которые мы будем использовать на протяжении всего курса, поэтому изучите её внимательно (это относится и к остальным лекциям курса).



ЦЕЛИ КУРСА



JavaScript

JavaScript (далее – JS) на сегодняшний день является самым распространённым языком программирования. На нём можно делать практически всё – от веб-приложений, до программ для смартфонов и компьютеров (desktop-приложения) и даже микроконтроллеров.

Поэтому мы продолжим изучать именно его и начнём с приложений, которые могут запускаться вне браузера.



ИНСТРУМЕНТЫ



Инструменты

Для прохождения основной части курса вам понадобятся два инструмента: Node.js и редактор VS Code (для дополнительных лекций понадобятся также другие инструменты).

Кроме того, важны следующие три момента:

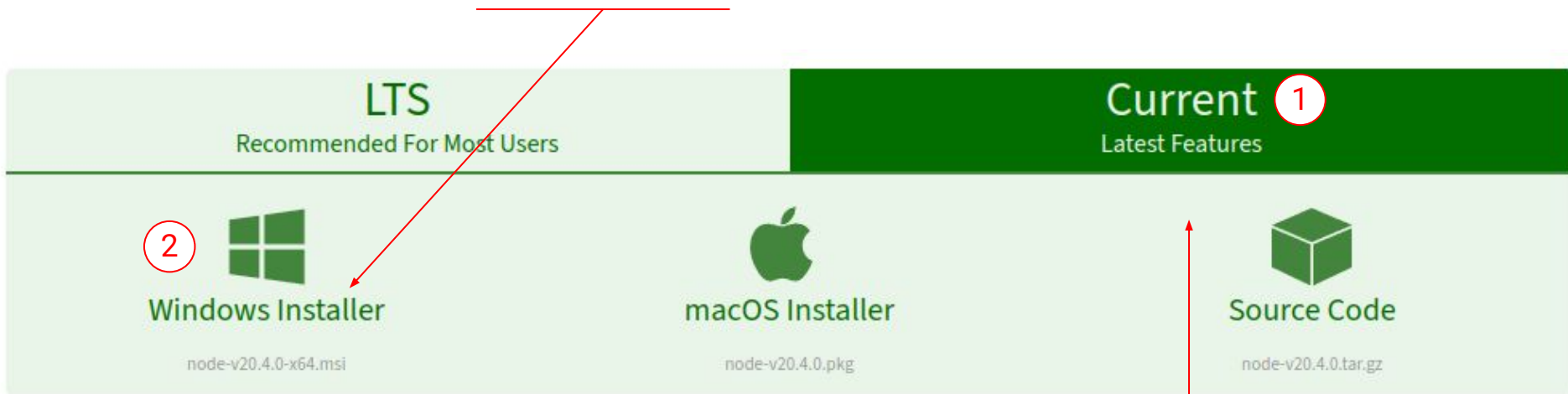
1. У вас должны быть права администратора на компьютере (чтобы вы могли устанавливать программы)
2. Ваш пользователь должен называться по-английски, без пробелов в имени (если это не так – переименуйте)
3. Создавайте все проекты где-нибудь на диске **C:**, например, в каталоге **projects** (следите за тем, чтобы в именах каталогов и файлов не было пробелов, не английских символов и т.д.)

В любом случае, если у вас возникнут проблемы, пишите в канал курса.



Node.js

Для установки Node.js перейдите по адресу <https://nodejs.org/en/download/> и выберите установочный файл для вашей операционной системы (далее – ОС). Например, для Windows, нужно выбрать Windows Installer:

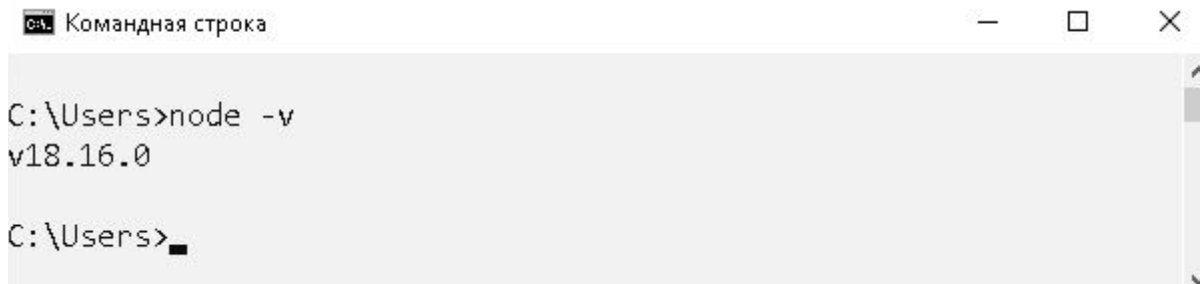


Выбирайте последнюю версию (Current, а не LTS).



Инструменты

Если у вас уже установлен Node.js, то удостоверьтесь, что версия 20+ (в примере 18):



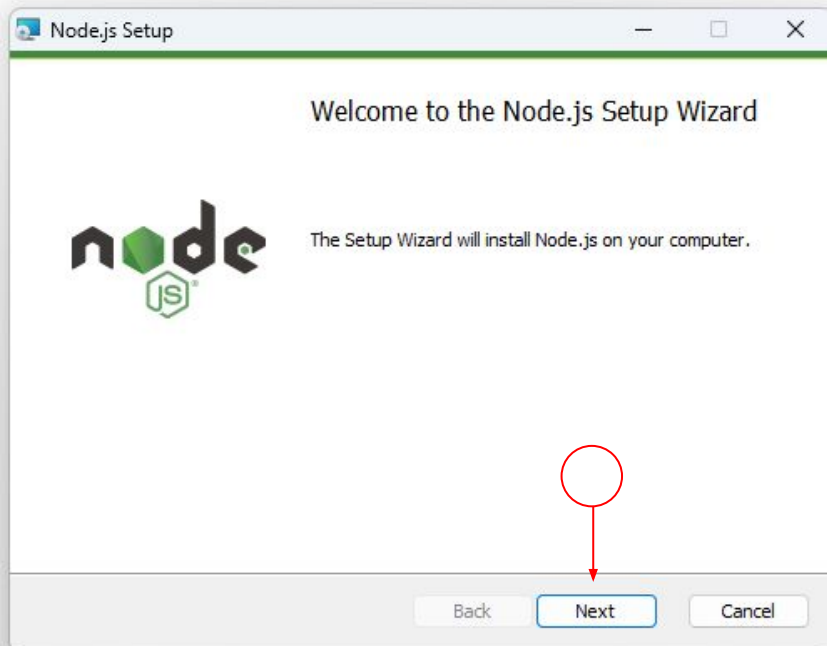
```
Командная строка
C:\Users>node -v
v18.16.0
C:\Users>
```

В противном случае удалите старую версию и установите новую.



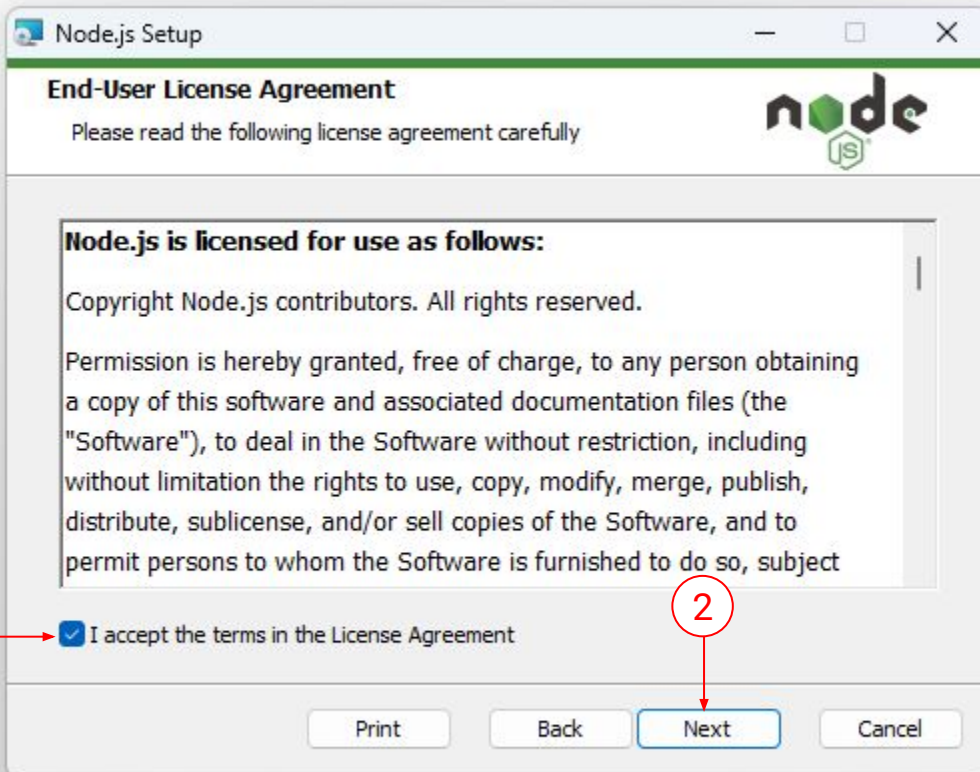
Node.js

Дождитесь активации кнопки **Next** (это может занять несколько минут), после чего нажмите на кнопку **Next**:



Node.js

Прочитайте и примите условия лицензионного соглашения, после чего нажмите на кнопку **Next**:

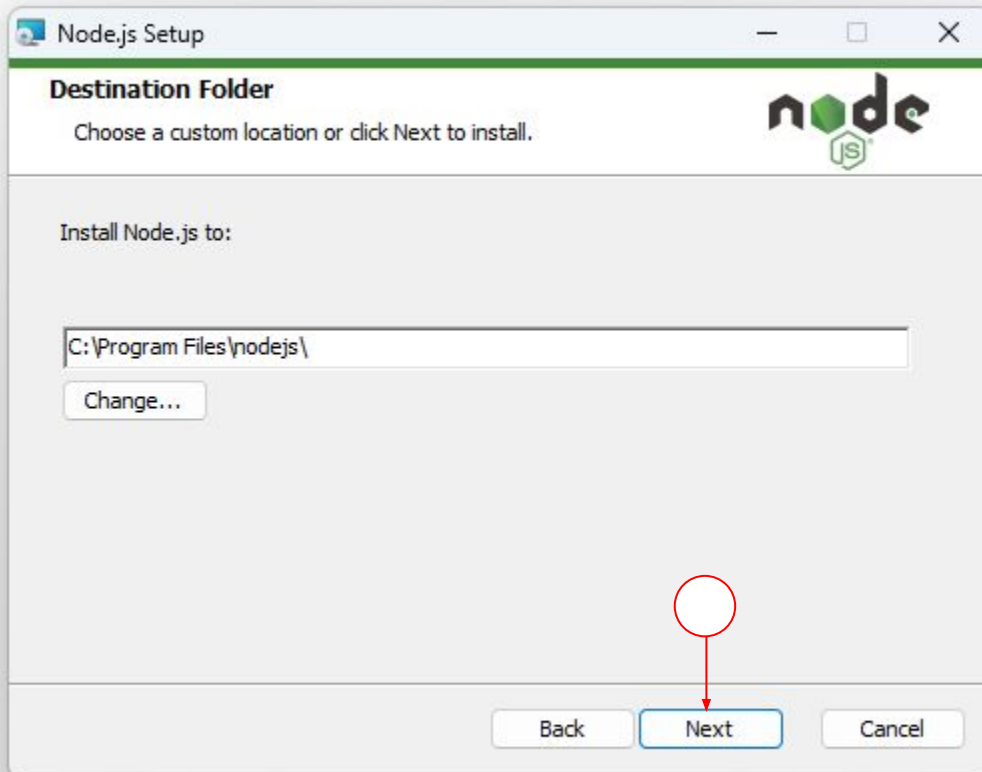


Node.js



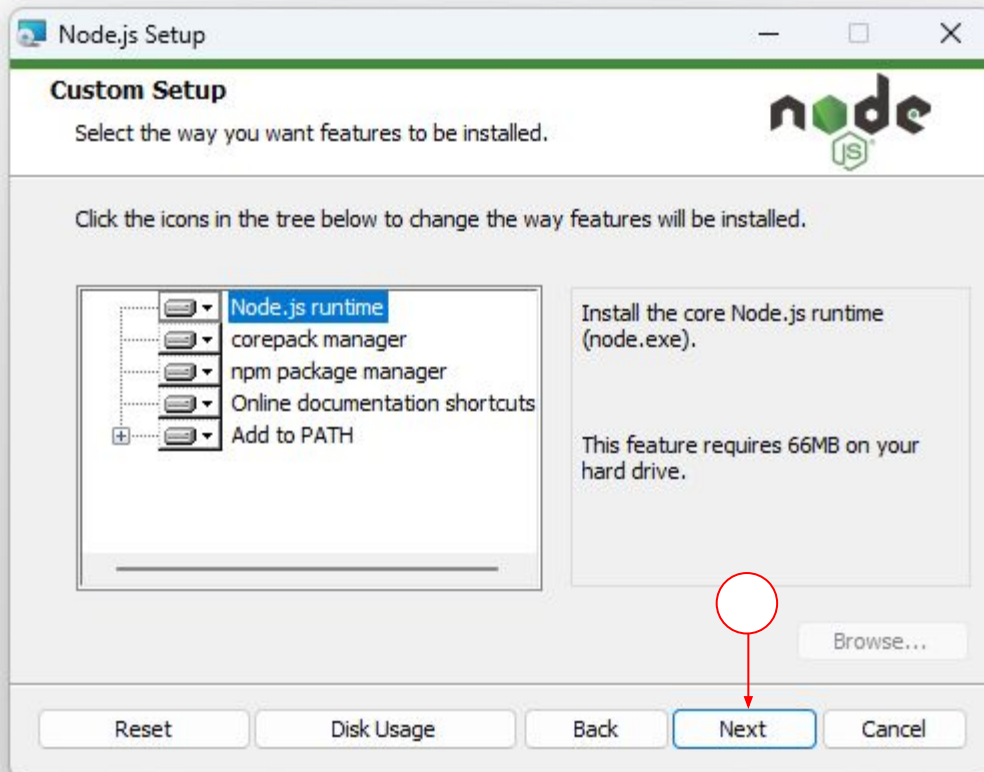
Node.js

Подтвердите установку в указанный каталог:



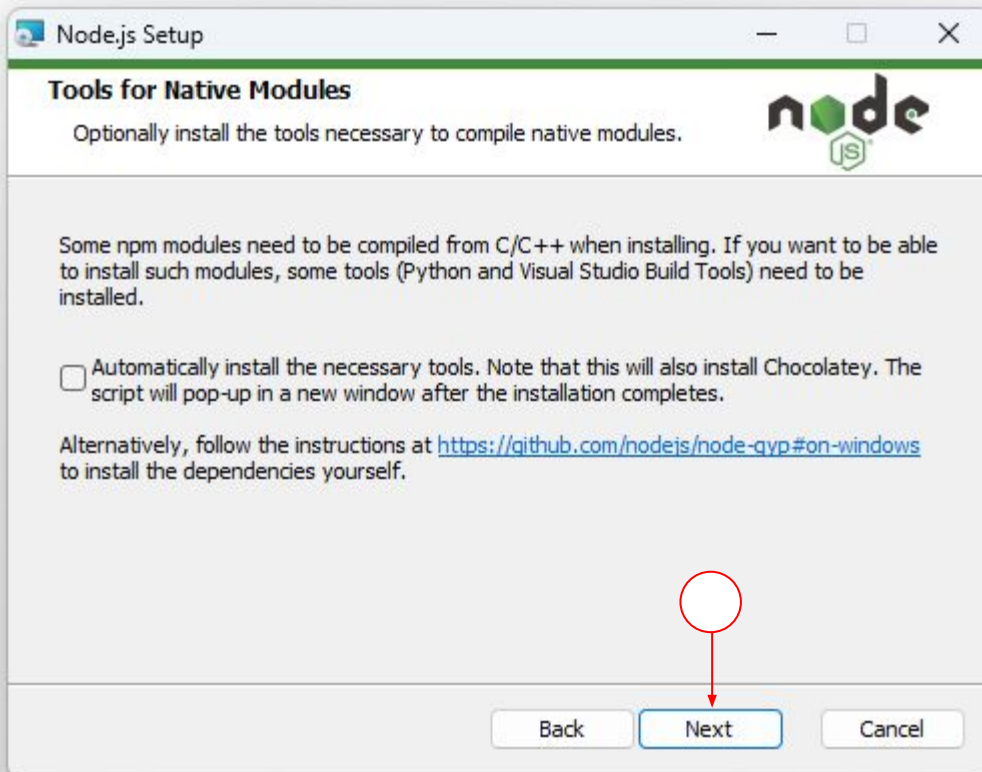
Node.js

Оставьте выбранные значения по умолчанию и нажмите на кнопку **Next**:



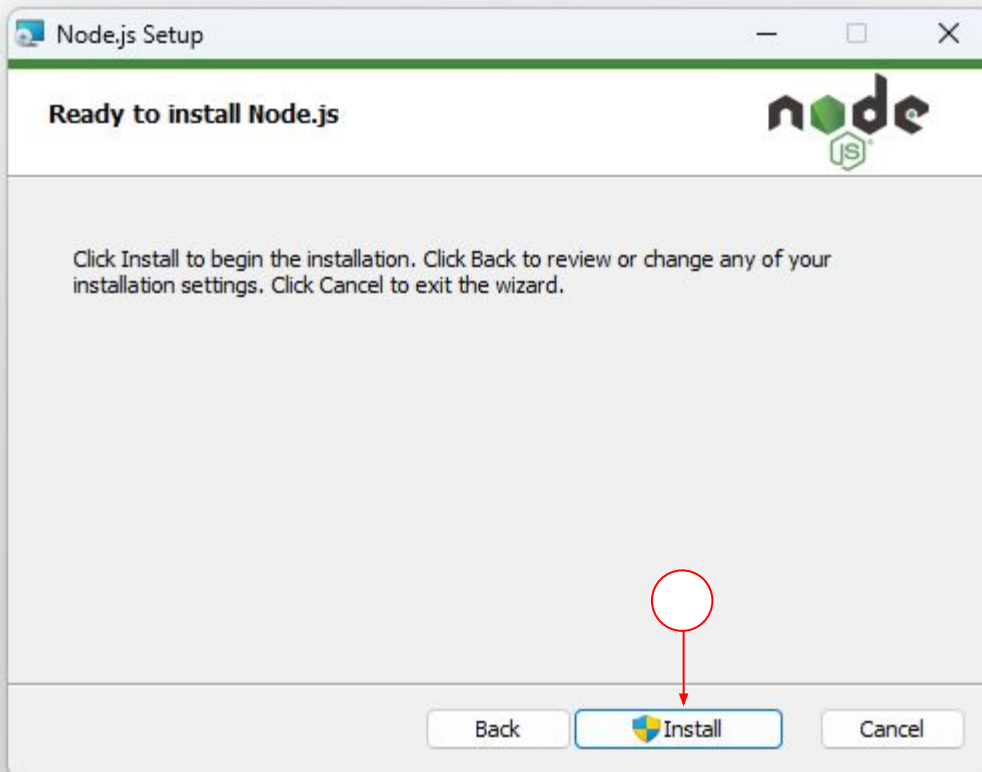
Node.js

Оставьте выбранные значения по умолчанию и нажмите на кнопку **Next**:



Node.js

Подтвердите установку нажатием на кнопку **Install**:



VS Code

VS Code самый популярный редактор кода (специальная программа, которая позволяет вам писать код) для JS.

VS Code у вас должен быть установлен в рамках предыдущих лекций. Если же нет, то перейдите по адресу code.visualstudio.com и скачайте установочный файл для вашей операционной системы.



NODE.JS



Node.js

Node.js – это среда выполнения JavaScript, предназначенная для:

1. Создания масштабируемых сетевых приложений
2. Создания инструментов (для frontend-разработки)

JavaScript – это язык программирования, с помощью которого мы можем эти самые приложения создавать.

Чтобы было понятно, как всё устроено, давайте посмотрим на две ключевых сферы использования языка: клиентская часть веб-приложений и серверная часть веб-приложения*.

Примечание*: на Node.js можно делать не только веб-приложения.



Web Application

Типичное веб-приложение устроено следующим образом:

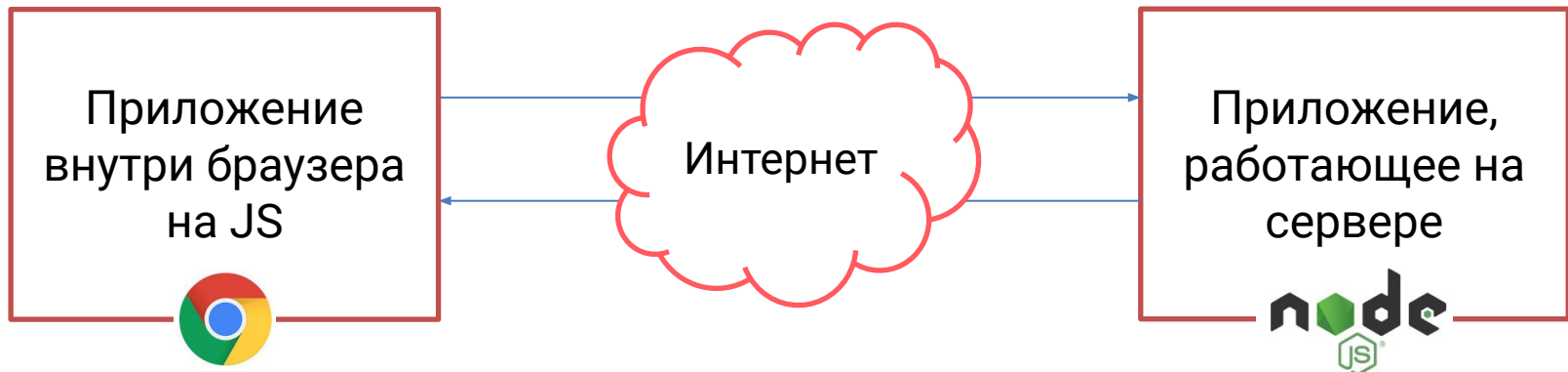


Например, если вы открываете страницу www.ya.ru, то браузер загружает и запускает клиентскую часть приложения, которая общается с серверной частью приложения.

Web Application

Как на самом деле это работает: у вас есть две программы – браузер (на вашем компьютере) и серверное приложение (на сервере). Сервер – это просто компьютер, который в отличие от вашего компьютера постоянно включен и подключен к сети Интернет.

Эти два приложения (браузер и сервер) взаимодействуют между собой, пересылая по сети определённую информацию (представьте, что вы говорите с человеком по телефону - вы тоже пересылаете друг другу информацию посредством сети).



Браузер

Но не всё так просто: браузер – это универсальное приложение (т.е. вам не нужно для каждого веб-приложения скачивать собственный браузер).

Это достигается за счёт того, что браузер внутри себя исполняет другие программы, которые написаны на языке JavaScript. Это значит, что каждый разработчик веб-приложений может написать свою программу, которая загрузится в браузер пользователя и там уже будет работать, взаимодействуя с сервером.



Браузер

При этом браузер помещает приложение в так называемую песочницу (sandbox):



Песочница – это ограничение приложения с точки зрения его возможностей.

Например, приложение, запущенное в браузере не может без вашего разрешения включить видеокамеру или микрофон на вашем ноутбуке.

В первую очередь, это сделано из соображений безопасности – чтобы клиентская часть веб-приложения не могла причинить существенный вред вам или вашему компьютеру.



Сервер

Под термином "сервер" чаще всего понимают две вещи:

1. Компьютер, который постоянно включен и подключен к сети Интернет
2. Приложение (программа), которое работает на этом компьютере и обрабатывает запросы от клиентов (например, браузера)

В случае приложения – это просто программа, задача которой очень проста: клиентские приложения обращаются к серверному приложению, формируя запрос, а сервер, обработав запрос, генерирует ответ. Аналогия: есть банк и его клиенты. Так вот клиенты отправляют запрос в банк (например, перевести деньги или оплатить счёт), а банк их обрабатывает, отвечая клиентам – успешно или не успешно были обработаны их запросы (вдруг вы случайно захотите оплатить несуществующий счёт).



Сервер

Для серверного приложения, в большинстве случаев, не существует никаких песочниц – ему предоставляется доступ ко всем возможностям ОС*: оно может создавать/удалять файлы, выполнять любые другие действия.

Приложение,
работающее на
сервере



Примечание*: на самом деле, и для серверных приложений есть ограничения. Существуют они тоже из соображений безопасности: например, приложение не может удалять какие-то критичные для работы файлы операционной системы или мешать работе других приложений (на сервере может быть много приложений). Ограничения можно устанавливать как на уровне операционной системы, так и на уровне среды исполнения – Node.js предоставляет такие возможности начиная с 20 версии.



Node.js

Node.js – это среда выполнения JavaScript. Что это значит? JavaScript – это просто язык, например, у нас есть русский и таджикский языки. Но язык – это просто средство общения: даже если вы скажете "Я хочу заказать такси", вы не сможете его заказать, если в вашем городе нет службы такси.

Точно так же и с JS – сам по себе он даёт вам возможность взаимодействовать с внешней средой. Эта самая внешняя среда должна предоставлять вам определённые "сервисы", которые вы можете использовать. Например, в городах это: освещение, водоснабжение, общественный транспорт и т.д.

Так же и здесь – Node.js предоставляет возможность приложениям на JS работать с внешней средой.



Node.js

Изначально, Node.js создавался как среда для разработки серверных приложений. Т.е. клиентскую часть (работающую в браузере) – frontend, мы итак писали на JS, а вот серверную часть – backend приходилось писать на C++, Java, C# или каком-нибудь PHP.

Таким образом, чтобы можно было разрабатывать полноценно frontend и backend, приходилось знать два языка (+ HTML/CSS, + SQL для работы с базами данных).

Поэтому возникло желание сделать так, чтобы можно было выучить один язык и на нём иметь возможность разрабатывать и frontend, и backend.



Node.js

Изначально, Node.js создавался как среда для разработки серверных приложений. Т.е. клиентскую часть (работающую в браузере) – frontend, мы итак писали на JS, а вот серверную часть – backend приходилось писать на C++, Java, C# или каком-нибудь PHP.

Таким образом, чтобы можно было разрабатывать полноценно frontend и backend, приходилось знать два языка (+ HTML/CSS, + SQL для работы с базами данных).

Поэтому возникло желание сделать так, чтобы можно было выучить один язык и на нём иметь возможность разрабатывать и frontend, и backend.



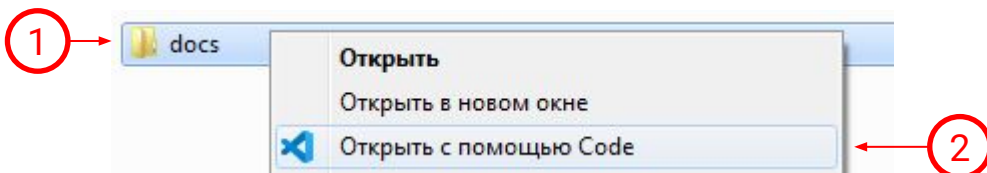
ПЕРВОЕ ПРИЛОЖЕНИЕ



Первое приложение

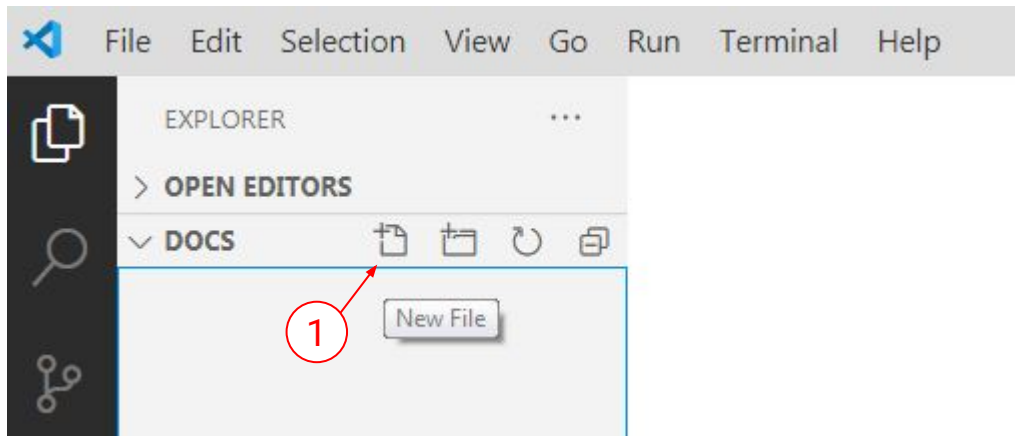
Но хватит теории, давайте попробуем сделать своё первое приложение. Мы рекомендуем вам создать специальный каталог на диске **C:** под названием **projects**, в котором и будут ваши проекты. **Важно:** делайте все ваши проекты именно в **C:\projects**, а не где-то на рабочем столе! Некоторые инструменты, которые мы будем рассматривать, не будут работать, если вы их поместите на рабочий стол или "запрячете" куда-то. Мы будем исходить из того, что все ваши проекты находятся в **C:\projects**.

Создайте в каталоге ваших проектов каталог **docs** и откройте его в VS Code (клик правой кнопкой мыши на каталоге):

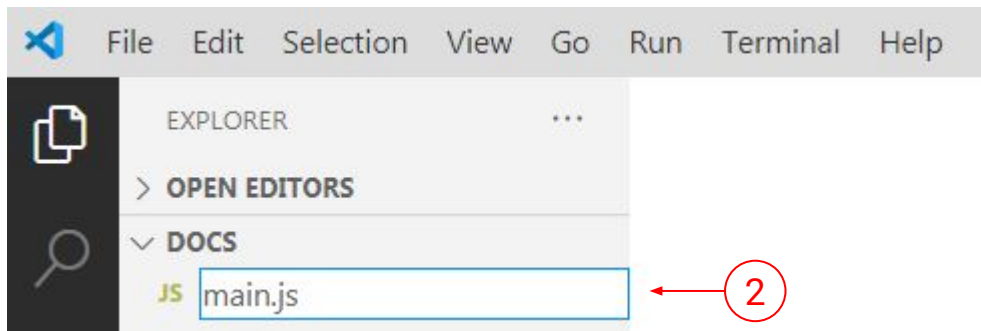


JS

В боковой панельке нажмите на **+** для создания нового файла:

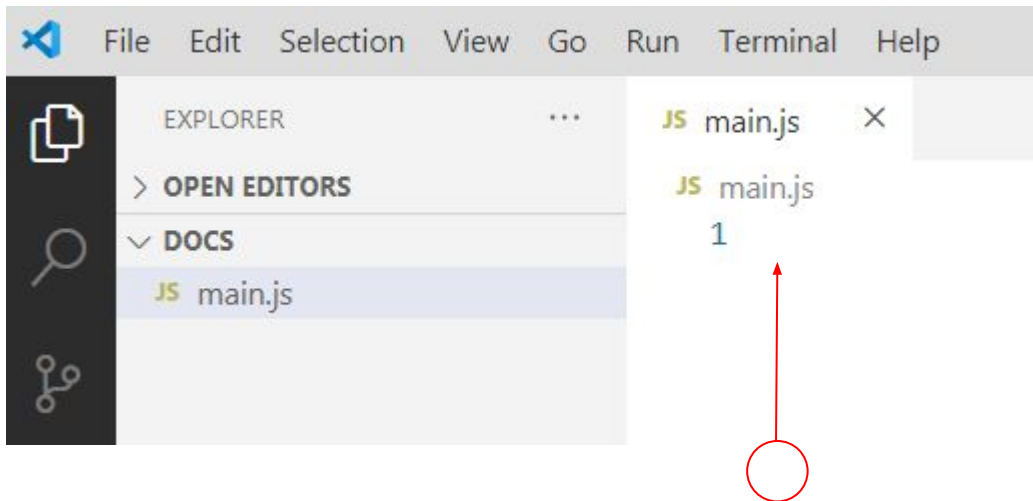


Введите **main.js** и нажмите на **Enter**:



JS

После этого файл откроется для редактирования в основной части редактора:

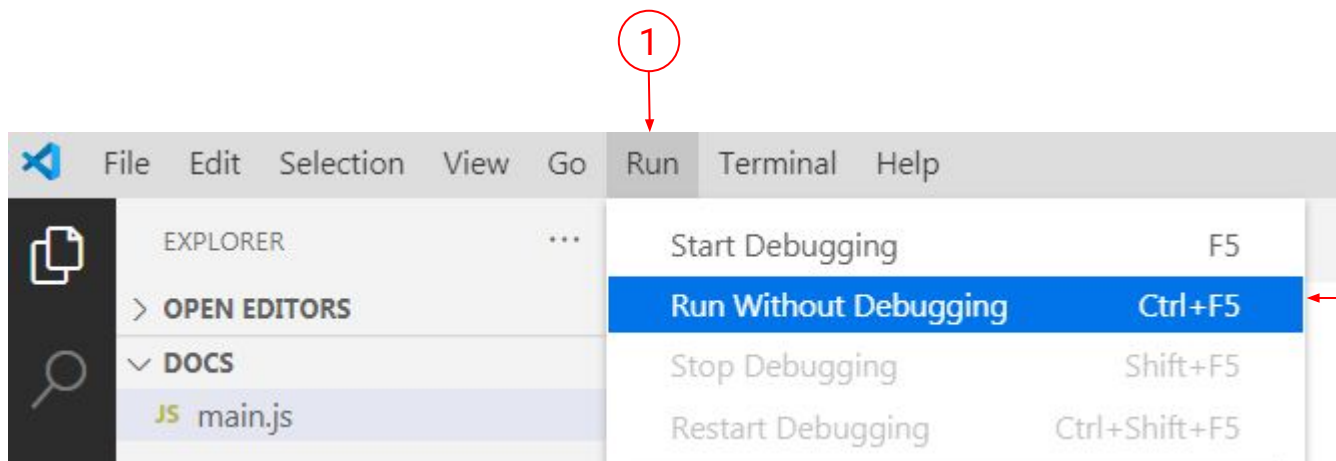


Там мы и будем писать код.

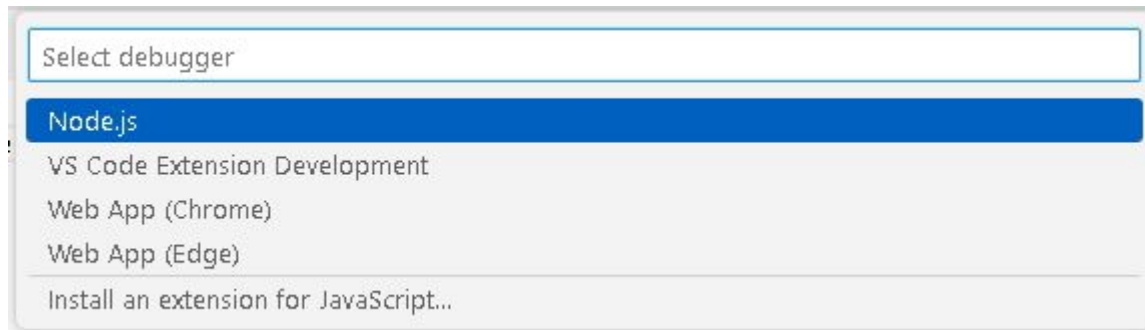


Запуск

Для начала попробуем запустить приложение средствами VS Code:



Ctrl + F5 позволит запускать
с клавиатуры,
чтобы каждый раз
не "лазить" в меню



Запуск

Если вы всё сделали правильно, то в нижней части редактора увидите следующую панельку:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
C:\Program Files\nodejs\node.exe c:\projects\docs\main.js  
Debugger listening on ws://127.0.0.1:58987/eb98f6b9-0e5e-46df-bd84-b3ea163aa47c  
For help, see: https://nodejs.org/en/docs/inspector  
Debugger attached.  
Waiting for the debugger to disconnect...  
Process exited with code 0
```

Что произошло? Исполняемый файл Node.js (**node.exe**) прочитал наш файл **main.js** (а именно все инструкции, которые были описаны в нём) и выполнил.

Поскольку инструкций в нашем файле не было, то и Node.js ничего не сделал интересного.



Запуск

Давайте попробуем что-нибудь добавить из уже знакомого нам:



```
JS main.js ×  
JS main.js  
1 con  
  [e] console      var console: Console  
  [e] const  
  [e] continue
```

Важно: несмотря на то, что некоторые объекты в Node.js, такие же, как в браузере, не стоит думать, что в Node.js доступно всё то же, что и в браузере (здесь не будет никаких **window**, **document** и прочего).



Запуск

JS main.js X

JS main.js

```
1 console.log('hello world');
```

```
2
```

```
3
```

Запустим и увидим в консоли:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Program Files\nodejs\node.exe c:\projects\docs\main.js

Debugger listening on ws://127.0.0.1:59243/89153894-28aa-4f9d-a0d8-a556ebc71f74

For help, see: <https://nodejs.org/en/docs/inspector>

Debugger attached.

--- Truncated to last 15 messages, set outputCapture to 'all' to see more ---

disconnect...

Process exited with code 0

hello world



Запуск

Node.js выполняет наш файл сверху вниз, поэтому если мы разместим два вызова `console.log`, то исполняться они будут последовательно:

JS main.js ×

JS main.js

```
1 console.log('hello world');
2
3 console.log('hello node.js');
4 |
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Program Files\nodejs\node.exe c:\projects\docs\main.js

Debugger listening on ws://127.0.0.1:59452/f42c6b24-67c8-407f-8e54-7158f56f19fb

For help, see: <https://nodejs.org/en/docs/inspector>

Debugger attached.

--- Truncated to last 15 messages, set outputCapture to 'all' to see more ---
disconnect...

Process exited with code 0

hello world

hello node.js



Запуск

Важно: у Node.js свой Event Loop (не такой, как в браузере, поскольку здесь нет инструкций отрисовки элементов на странице, да и самих элементов нет).

Про Event Loop Node.js мы будем говорить в следующих лекциях.

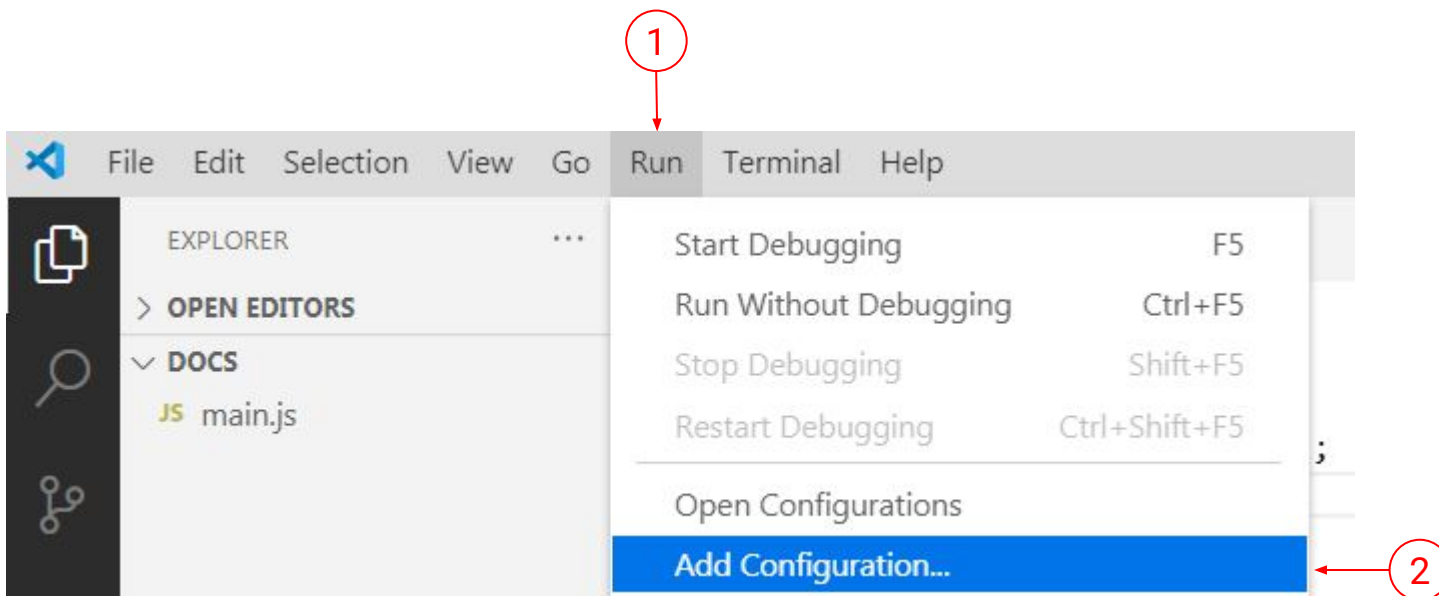


КОНФИГУРАЦИЯ ЗАПУСКА

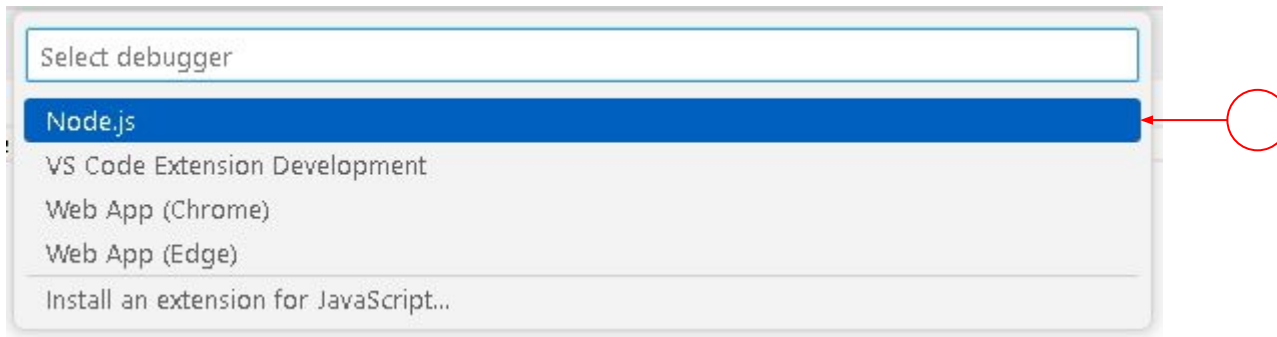


Конфигурация запуска

Каждый раз выбирать Node.js из выпадающего списка не особо удобно, поэтому мы можем настроить конфигурацию запуска: сообщить VS Code, что мы хотим, чтобы по умолчанию запускался Node.js:



Конфигурация запуска

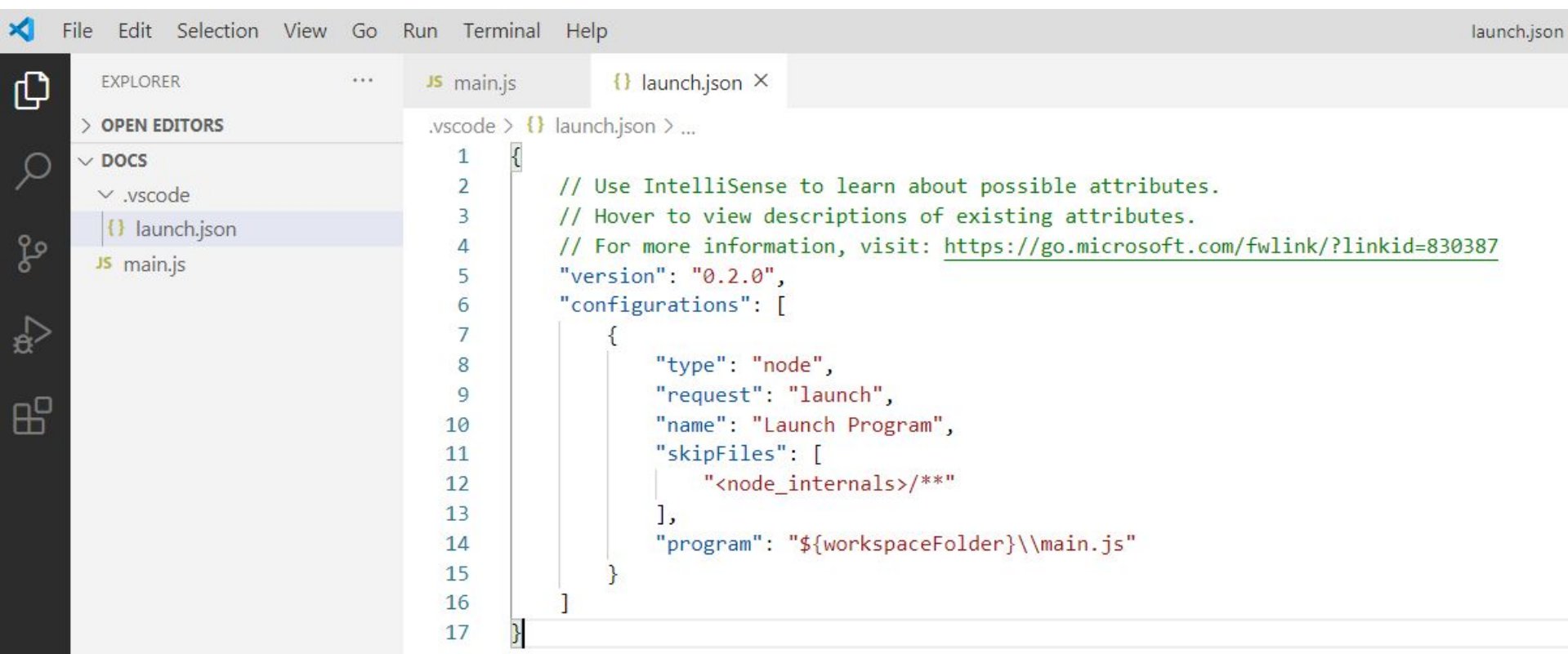


Это приведёт к тому, что у вас автоматически создастся каталог **.vscode** с файлом **launch.json** в нём. А дальнейшие нажатия **Ctrl + F5** будут приводить к автоматическому запуску Node.js.



Конфигурация запуска

Вот так это будет выглядеть (вам это создавать не нужно - оно произойдёт автоматически):



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left shows the file structure with 'launch.json' selected under the '.vscode' folder. The main editor area displays the 'launch.json' file, which contains a JSON configuration for launching a Node.js program. The configuration includes a 'version' of '0.2.0' and a 'configurations' array with one configuration object. This object specifies the type as 'node', the request as 'launch', the name as 'Launch Program', and the program path as '\${workspaceFolder}\\main.js'. There are also comments about using IntelliSense and a link to Microsoft's documentation.

```
.vscode > {} launch.json > ...
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "type": "node",
9              "request": "launch",
10             "name": "Launch Program",
11             "skipFiles": [
12                 "<node_internals>/**"
13             ],
14             "program": "${workspaceFolder}\\main.js"
15         }
16     ]
17 }
```

Переключитесь снова на вкладку **main.js** (нужно кликнуть на ней) и продолжим.



DEBUGGER



Debugger

Для того, чтобы увидеть, как Node.js исполняет наш файл (и исполняет ли вообще), есть специальный инструмент, который называется Debugger (отладчик).

Отладчик – это специальный инструмент, который позволяет перевести Node.js в режим пошагового выполнения. При этом мы можем смотреть, что и как выполняется.




Debugger

Открываем файл `main.js` и на боковой панели кликаем левой кнопкой мыши (либо клавиша **F9**) - поставится точка остановки (breakpoint):

JS main.js ×

JS main.js



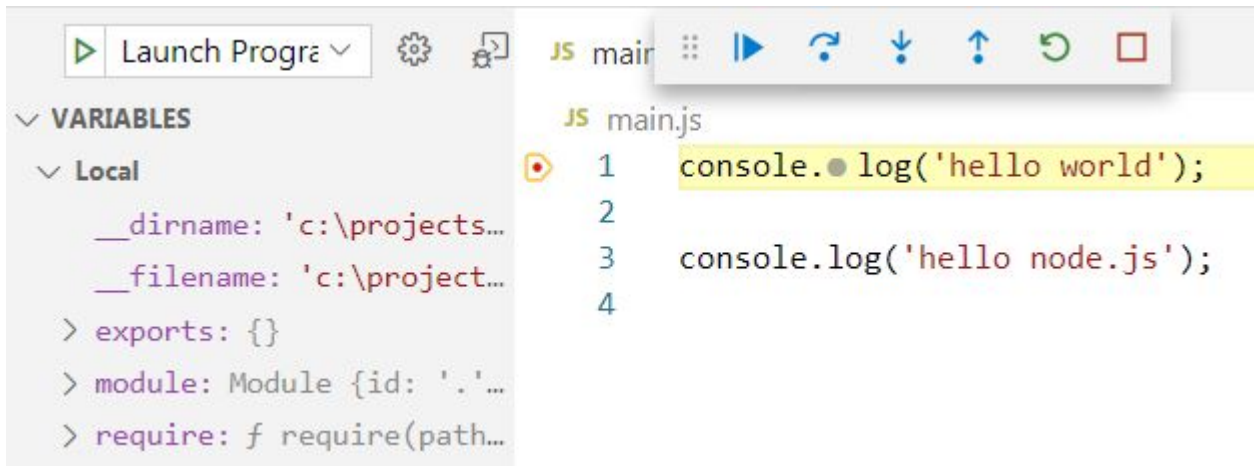
```
1 console.log('hello world');  
2  
3 console.log('hello node.js');  
4 |
```

Точка остановки – это строка, на которой остановится выполнение. Для того, чтобы её активировать, нужно запустить приложение в режим отладки (**F5**).

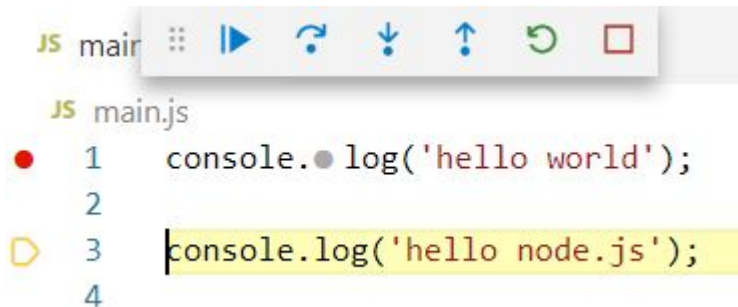


Debugger

Мы увидим строку, подсвеченную жёлтым – это значит, что эту строку Node.js ещё не исполнил (поэтому и в консоли **hello world**) ещё не будет:

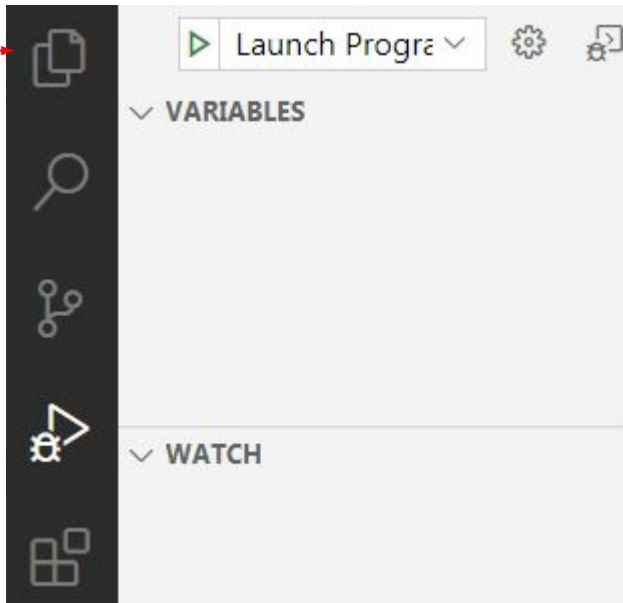


Чтобы перейти на следующую строку, нужно нажать на кнопку  или **F10**:



Debugger

Чтобы обратно переключиться в режим отображения файлов, просто выберите в боковой панели файловый менеджер.



Таким образом, Debugger позволяет нам прошагать всю программу, и понять, как она выполняется на самом деле.



ИТОГИ



Итоги

В этой лекции мы обсудили достаточно много важных моментов:

1. Установку инструментов
2. Работу в VS Code

В следующих лекциях мы будем опираться на то, что вы уже изучили в этой лекции (и не будем детально описывать процессы создания файлов, запуска приложений, переход в режим отладчика и т.д.).



ДОМАШНЕЕ ЗАДАНИЕ



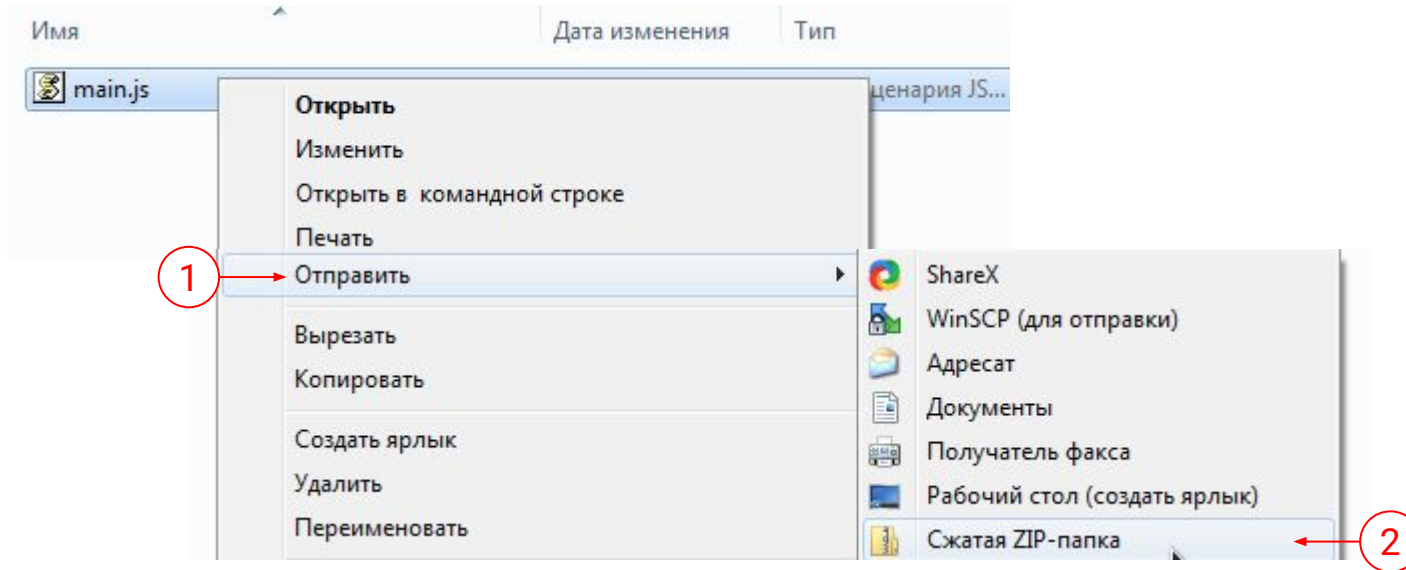
ДЗ: Hello Node.js!

Создайте проект аналогично тому, как мы это делали на лекции (включая создание конфигурации запуска). В `main.js` разместите код: `console.log('Hello, Node.js!');`



Как сдавать ДЗ

Вам нужно запаковать в zip-архив содержимое каталога с проектом (не сам каталог, а его содержимое за исключением каталога `node_modules*`) – выделяете его и выбираете Отправить –> Сжатая ZIP-папка:



Полученный архив загружаете в личном кабинете пользователя.

Важно: учитывается только последняя отправленная попытка.

Примечание*: у вас пока его нет, но скоро появится.



Спасибо за внимание

alif skills

2023г.

