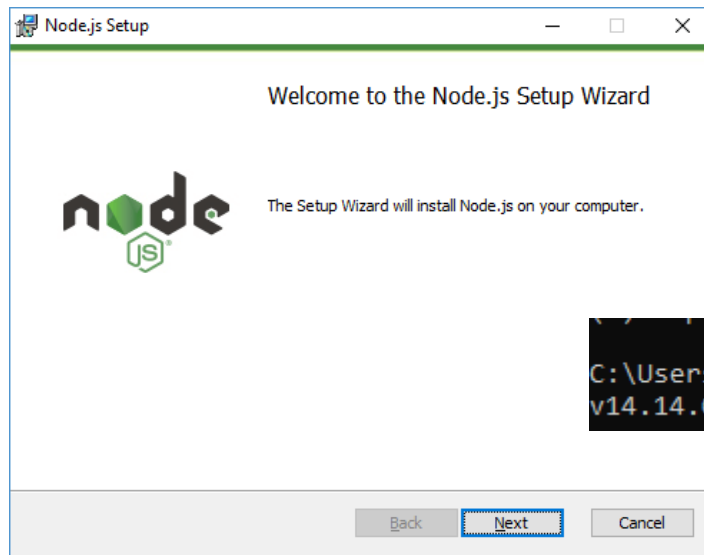
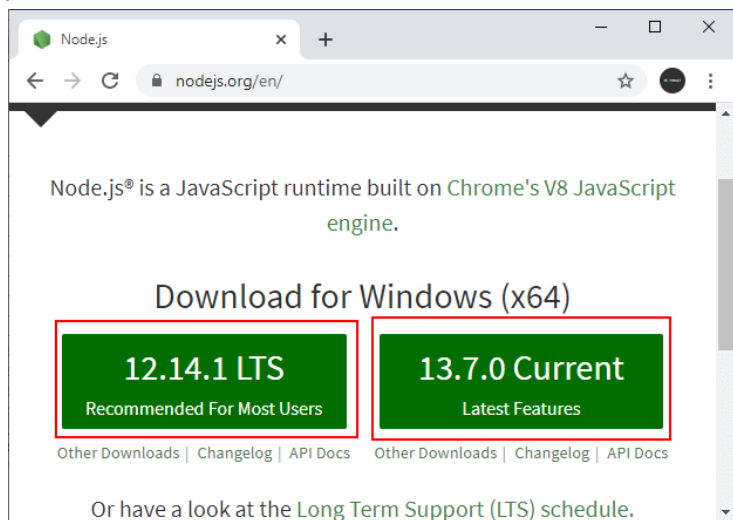


# Введение в Node JS

Модуль 1 (5 пар)

# Введение в Node JS. Установка

- Node.js представляет среду выполнения кода на JavaScript, которая построена на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код. Node.js прежде всего предназначен для создания серверных приложений на языке JavaScript. Хотя также существуют проекты по написанию десктопных приложений (Electron) и даже по созданию кода для микроконтроллеров. Но прежде всего мы говорим о Node.js, как о платформе для создания веб-приложений.
- Node.js является открытым проектом, исходники которого можно посмотреть на [github.com](https://github.com).
- Для загрузки перейдем на официальный сайт <https://nodejs.org/en/>. На главной странице мы сразу увидим две возможные опции для загрузки: самая последняя версия NodeJS и LTS-версия.
- Загрузим последнюю версию. В моем случае это версия 14.14.0. Для Windows установщик представляет файл с расширением `msi`. После запуска откроется программа установщика:



```
C:\Users\ruban>node -v
v14.14.0
```

## Инструменты разработки. REPL

- Для разработки под Node JS достаточно простейшего текстового редактора, в частности, Notepad++. Также можно использовать более изощренные редакторы типа Atom, Sublime, Visual Studio Code, либо среды разработки, которые поддерживают работу с Node.JS, например, Visual Studio или WebStorm.
- После установки NodeJS нам становится доступным такой инструмент как REPL. REPL (Read Eval Print Loop) представляет возможность запуска выражений на языке JavaScript в командной строке или терминале.
- Так, запустим командную строку (на Windows) или терминал (на OS X или Linux) и введем команду node. После ввода этой команды мы можем выполнять различные выражения на JavaScript:

```
C:\WINDOWS\system32>node
Welcome to Node.js v13.7.0
Type ".help" for more information.
> 2+6
8
>
```

- Или используем какую-нибудь функцию JS:

```
> console.log("Hello NodeJS");
Hello NodeJS
undefined
>
```

# REPL

- Можно определять свои функции и затем их вызывать, например, возведение числа в квадрат:

```
> function square(x){return x * x;}
undefined
>square(5)
25
>
```

- Если мы введем что-то неправильно, то REPL укажет об ошибке:

```
Администратор: Командная строка - node
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Все права защищены.

C:\WINDOWS\system32>node -v
v13.7.0

C:\WINDOWS\system32>node
Welcome to Node.js v13.7.0.
Type ".help" for more information.
> 2+6
8
> console.log("Hello NodeJS");
Hello NodeJS
undefined
> function square(x){return x*x;}
undefined
> square(5)
25
> scuare(7)
Uncaught ReferenceError: scuare is not defined  ошибка
>
```

## Выполнение файла

- Вместо того чтобы вводить весь код напрямую в консоль, удобнее вынести его во внешний файл. Например, создадим на жестком диске новый каталог, допустим, C:\node\helloapp, в который поместим новый файл app.js со следующим кодом:

```
console.log("Hello world");
```

- В командной строке перейдем с помощью команды cd к каталогу helloapp, а затем выполним команду:

```
node app.js
```

- Данная команда выполнит код из файла app.js:

```
Администратор: Командная строка
Microsoft Windows [Version 10.0.14393]
(c) Корпорация Майкрософт (Microsoft Corporation), 2016. Все права защищены.

C:\WINDOWS\system32>cd C:\node\helloapp

C:\node\helloapp>node app.js
Hello world

C:\node\helloapp>
```

# Первое приложение на Node.js

- Напишем первое простейшее приложение для NodeJS. Для создания приложений можно использовать практически все стандартные конструкции языка JavaScript. Исключением является работа с DOM, так как приложение будет запускаться на сервере, а не в браузере, поэтому DOM и такие объекты как window или document в данном случае нам будут недоступны.
- Для этого вначале создадим для приложения каталог на жестком диске. К примеру, я создал каталог C:\node\helloapp. В этом каталоге создадим файл app.js.
- Определим в файле app.js следующий код:

```
const http = require("http");
http.createServer(function(request,response){

    response.end("Hello NodeJS!");

}).listen(3000, "127.0.0.1",function(){
    console.log("Сервер начал прослушивание запросов на порту 3000");
});
```

- Вкратце разберем этот код.
- На первой строке мы получаем модуль http, который необходим для создания сервера. Это встроенный модуль, и для его загрузки необходимо применить функцию require():

```
const http = require("http");
```

# Первое приложение на Node.js

- Далее с помощью метода `createServer()` создается новый сервер для прослушивания входящих подключений и обработки запросов. В качестве параметра этот метод принимает функцию, которая имеет два параметра. Первый параметр `request` хранит всю информацию о запросе, а второй параметр `response` используется для отправки ответа. В данном случае ответ представляет простую строку "Hello NodeJS!" и отправляется с помощью метода `response.end()`.
- Но метод `http.createServer()` только создает сервер. Чтобы сервер начал прослушивать входящие подключения у него надо вызвать метод `listen`:

```
.listen(3000, "127.0.0.1",function(){
    console.log("Сервер начал прослушивание запросов на порту 3000");
});
```

- Этот метод принимает три параметра. Первый параметр указывает на локальный порт, по которому запускается сервер. Второй параметр указывает на локальный адрес. То есть в данном случае сервер будет запускаться по адресу 127.0.0.1 или localhost на порту 3000.
- Третий параметр представляет функцию, которая запускается при начале прослушивания подключений. Здесь эта функция просто выводит диагностическое сообщение на консоль.

## Первое приложение на Node.js

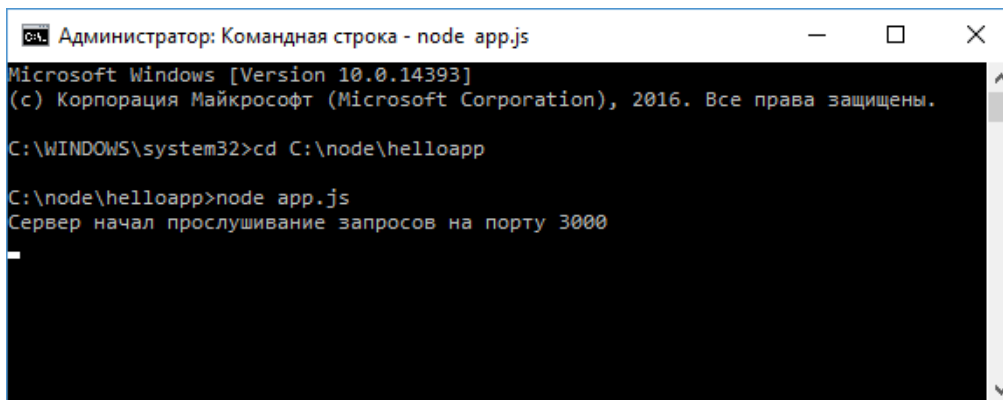
- Теперь запустим сервер. Для этого откроем терминал (в OS X или Linux) или командную строку (в Windows). С помощью команды `cd` перейдем к каталогу приложения:

```
cd C:\node\helloapp
```

- Затем вызовем следующую команду:

```
node app.js
```

- Она запускает сервер:

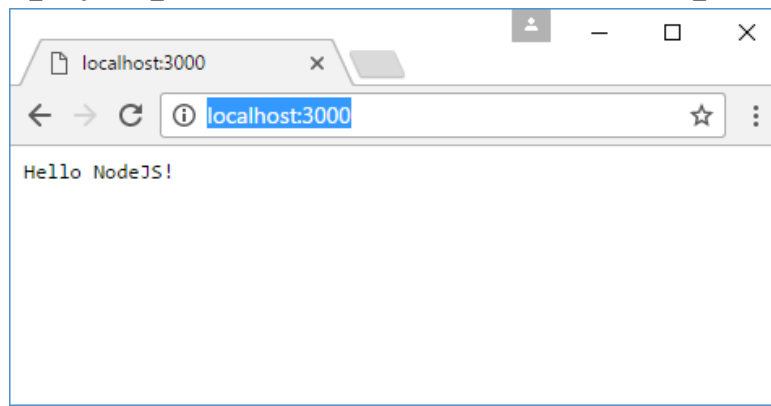


```
Администратор: Командная строка - node app.js
Microsoft Windows [Version 10.0.14393]
(c) Корпорация Майкрософт (Microsoft Corporation), 2016. Все права защищены.

C:\WINDOWS\system32>cd C:\node\helloapp

C:\node\helloapp>node app.js
Сервер начал прослушивание запросов на порту 3000
```

- Далее откроем браузер и введем в адресную строку адрес <http://localhost:3000/>:





# Модули

- Node.js использует модульную систему. То есть вся встроенная функциональность разбита на отдельные пакеты или модули. Модуль представляет блок кода, который может использоваться повторно в других модулях.
- При необходимости мы можем подключать нужные нам модули. Какие встроенные модули есть в node.js и какую функциональность они предоставляют, можно узнать из документации. (<https://nodejs.org/api/>)
- Для загрузки модулей применяется функция require(), в которую передается название модуля. К примеру, в первом приложении из предыдущей темы для получения и обработки запроса был необходим модуль http:

```
const http = require("http");
```

- После получения модуля мы сможем использовать весь определенный в нем функционал, который опять же можно посмотреть в документации.
- Подобным образом мы можем загружать и использовать другие встроенные модули. Например, используем модуль os, который предоставляет информацию об окружении и операционной системе:

```
const os = require("os");
// получим имя текущего пользователя
let userName = os.userInfo().username;

console.log(userName);
```

# Модули

- Мы не ограничены встроенными модулями и при необходимости можем создать свои. Так, в примере ранее проект состоял из файла app.js, в котором создавался сервер, обрабатывающий запросы. Добавим в тот же каталог новый файл greeting.js и определим в нем следующий код:

```
console.log("greeting module");
```

- В файле app.js подключим наш модуль:

```
const greeting = require("./greeting");
```

- В отличие от встроенных модулей для подключения своих модулей надо передать в функцию require относительный путь с именем файла (расширение файла необязательно):

```
const greeting = require("./greeting");
```

- Запустим приложение:

```
Администратор: Командная строка

C:\node\helloapp>node app.js
greeting module

C:\node\helloapp>
```

- На консоль выводится та строка, которая определена в файле greeting.js.

# Модули

- Теперь изменим файл `greeting.js`:

```
let currentDate = new Date();
module.exports.date = currentDate;

module.exports.getMessage = function(name){
  let hour = currentDate.getHours();
  if(hour > 16)
    return "Добрый вечер, " + name;
  else if(hour > 10)
    return "Добрый день, " + name;
  else
    return "Доброе утро, " + name;
}
```

- Здесь определена переменная `currentDate`. Однако из вне она недоступна. Она доступна только в пределах данного модуля. Чтобы определенные переменные или функции модуля были доступны, необходимо определить их в объекте `module.exports`. Объект `module.exports` - это то, что возвращает функция `require()` при получении модуля.
- Вообще объект `module` представляет ссылку на текущий модуль, а его свойство `exports` определяет все свойства и методы модуля, которые могут быть экспортированы и использованы в других модулях. Подробнее определение загрузки модуля и все его функции можно посмотреть на странице <https://github.com/nodejs/node/blob/master/lib/module.js>.
- В частности, здесь определяется свойство `date` и метод `getMessage`, который принимает некоторый параметр.

# Модули

- Далее изменим файл app.js:

```
const os = require("os");
const greeting = require("./greeting");

// получим имя текущего пользователя
let userName = os.userInfo().username;

console.log(`Дата запроса: ${greeting.date}`);
console.log(greeting.getMessage(userName));
```

- Все экспортированные методы и свойства модуля доступны по имени: `greeting.date` и `greeting.getMessage()`.
- Перезапустим приложение:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1139]
(c) Корпорация Майкрософт (Microsoft Corporation), 2019. Усі права захищено.

E:\Work\Step\NodeJS\Week1\Meeting1>node text_greet.js
Доброе утро, ruban
Дата запроса: Sun Oct 25 2020 00:45:56 GMT+0300 (за східноєвропейським літнім часом)

E:\Work\Step\NodeJS\Week1\Meeting1>
```

# Определение конструкторов и объектов в модуле

- Кроме определения простейших функций или свойств в модуле могут определяться сложные объекты или функции конструкторов, которые затем используются для создания объектов. Так, добавим в папку проекта новый файл `user.js`:

```
function User(name, age){
  this.name = name;
  this.age = age;
  this.displayInfo = function(){

    console.log(`Имя: ${this.name}  Возраст: ${this.age}`);
  }
}
User.prototype.sayHi = function() {
  console.log(`Привет, меня зовут ${this.name}`);
};
module.exports = User;
```

- Здесь определена стандартная функция конструктора `User`, которая принимает два параметра. При этом весь модуль теперь указывает на эту функцию конструктора:

```
module.exports = User;
```

- Подключим и используем этот модуль в файле `app.js`:

```
const User = require("./user.js");
let user1= new User("Vadym", 35);
user1.sayHi();
```

## Работа с модулями

- Рассмотрим некоторые аспекты работы с модулями в Node.js. Прежде всего надо отметить, что подключаемые модули кэшируются. В частности, в файле <https://github.com/nodejs/node/blob/master/lib/internal/modules/cjs/loader.js> есть такие строки:

```
var filename = Module._resolveFilename(request, parent, isMain);

var cachedModule = Module._cache[filename];
if (cachedModule) {
  updateChildren(parent, cachedModule, true);
  return cachedModule.exports;
}
```

- Это, с одной стороны, увеличивает производительность, а с другой, может создать некоторые проблемы, если мы не будем учитывать этот аспект. Например, возьмем проект из прошлой темы, где в главный файл приложения `app.js` подключается модуль `greeting.js`. Изменим файл `greeting.js` следующим образом:

```
module.exports.name = "Alice";
```

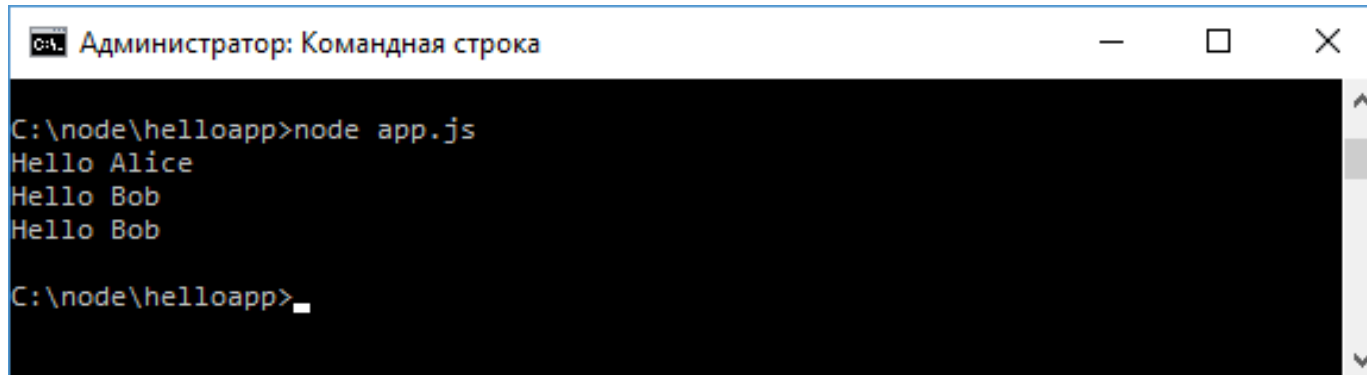
- В файле определена только одна строка, которая устанавливает свойство `name`.

## Работа с модулями

- Изменим код файла app.js:

```
var greeting1 = require("./greeting.js");  
console.log(`Hello ${greeting1.name}`); //Hello Alice  
  
var greeting2 = require("./greeting.js");  
greeting2.name= "Bob";  
  
console.log(`Hello ${greeting2.name}`); //Hello Bob  
// greeting1.name тоже изменилось  
console.log(`Hello ${greeting1.name}`); //Hello Bob
```

- Несмотря на то, что здесь два раза получаем модуль с помощью функции require, но обе переменных - greeting1 и greeting2 будут указывать на один и тот же объект.



```
Администратор: Командная строка  
C:\node\helloapp>node app.js  
Hello Alice  
Hello Bob  
Hello Bob  
C:\node\helloapp>
```

# Структура модулей

Нередко модули приложения образуют какие-то отдельные наборы или области. Такие наборы модулей лучше помещать в отдельные каталоги. Например, создадим в каталоге приложения подкаталог `welcome` и создадим в нем три новых файла:

- `index.js`
- `morning.js`
- `evening.js`
- В итоге общая структура проекта пусть будет выглядеть следующим образом:
- `welcome`

- `index.js`
- `morning.js`
- `evening.js`

- `app.js`
- `greeting.js`
- В файл `morning.js` поместим следующую строку:

```
module.exports = "Доброе утро";
```

- Аналогично изменим файл **`evening.js`**:

```
module.exports = "Добрый вечер";
```

- Эти два файла определяют сообщения приветствия в зависимости от времени суток.



# Структура модулей

- И определим в файле index.js следующий код:

```
const morning = require("./morning");
const evening = require("./evening");

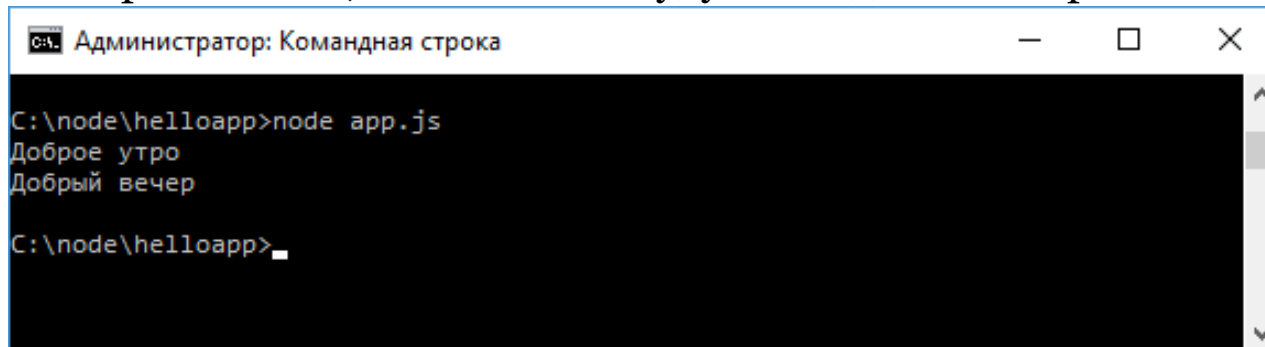
module.exports = {
  getMorningMessage : function(){ console.log(morning);},
  getEveningMessage : function(){ console.log(evening);}
}
```

- В модуле определен объект, который имеет две функции для вывода приветствий.
- Теперь используем этот модуль в файле app.js:

```
const welcome = require("./welcome");

welcome.getMorningMessage();
welcome.getEveningMessage();
```

- Несмотря на то, что нет такого файла как welcome.js, но если в проекте есть каталог, который содержит файл с именем index.js, то мы можем обращаться к модулю по имени каталога, как в данном случае.
- Запустим приложение, и на консоль будут выведены оба приветствия:



```
Администратор: Командная строка

C:\node\helloapp>node app.js
Доброе утро
Добрый вечер

C:\node\helloapp>
```

# Объект global и глобальные переменные

- Node.js предоставляет специальный объект global, который предоставляет доступ к глобальным, то есть доступным из каждого модуля приложения, переменным и функциям. Примерным аналогом данного объекта в javascript для браузера является объект window. Все доступные глобальные объекты можно посмотреть в документации.
- Для примера создадим следующий модуль greeting.js:

```
let currentDate = new Date();

global.date = currentDate;

module.exports.getMessage = function(){
  let hour = currentDate.getHours();
  if(hour >16)
    return "Добрый вечер, " + global.name;
  else if(hour >10)
    return "Добрый день, " + name;
  else
    return "Доброе утро, " + name;
}
```

- Здесь, во-первых, происходит установка глобальной переменной date: global.date = currentDate;
- Во-вторых, в модуле получаем глобальную переменную name, которая будет установлена из вне. При этом обратиться к глобальной переменной name мы можем через объект global: global.name, либо просто через имя name, так как переменная глобальная.

# Объект global и глобальные переменные

- Определим следующий файл приложения app.js:

```
const greeting = require("./greeting");

global.name = "Serhii";

global.console.log(date);
console.log(greeting.getMessage());
```

- Здесь устанавливаем глобальную переменную name, которую мы получаем в модуле greeting.js. И также выводим на консоль глобальную переменную date. Причем все глобальные функции и объекты, например, console, также доступны внутри global, поэтому мы можем написать и global.console.log(), и просто console.log().
- Однако по возможности все таки рекомендуется избегать определения и использования глобальных переменных, и преимущественно ориентироваться на создание переменных, инкапсулированных в рамках отдельных модулей.

# Передача параметров приложению

- При запуске приложения из терминала/командной строки мы можем передавать ему параметры. Для получения параметров в коде приложения применяется массив `process.argv`. Это аналогично тому, как в языках C/C++/C#/Java в функцию `main` передается набор аргументов в виде строкового массива.
- Первый элемент этого массива всегда указывает на путь к файлу `node.exe`, который вызывает приложение. Второй элемент массива всегда указывает на путь к файлу приложения, который выполняется.
- К примеру, определим следующий файл `app.js`:

```
let nodePath = process.argv[0];
let appPath = process.argv[1];
let name = process.argv[2];
let age = process.argv[3];

console.log("nodePath: " + nodePath);
console.log("appPath: " + appPath);
console.log();
console.log("name: " + name);
console.log("age: " + age);
```

- В данном случае мы ожидаем, что приложению будут переданы два параметра: `name` и `age`.
- Теперь запустим приложение с помощью следующей команды:

```
node app.js Tom 23
```

- В данном случае "Tom" и "23" - это те значения, которые помещаются соответственно в `process.argv[2]` и `process.argv[3]`:

**Спасибо за внимание.**