**Node.js #1 Базовые концепции и установка**

**(Basic Concepts & Setup Environment)**

создаем файл с расширением .js

***const userName = "Nemo";*** //объявили константу для имени

***const sayHi = (userName) => `Hello, my name is ${userName}`;*** //создали функцию sayHi, которая в качестве аргумента принимает имя и возвращает строку приветствия

***console.log(sayHi(userName));*** // полученный результат выводим в консоль

файл запускаем командой node имя\_файла\_без\_расширения\_js

**Node.js #2 Глобальные объекты (Global Objects)**

(см. страницу официальной документации - nodejs.org)

***console.log(global);*** //вывод в консоль глобального объекта global, чтобы посмотреть, что он из себя представляет

Внутри него есть доступы к таймерам, очередям микротасок и методам асинхронных остановок

//создадим таймер, который сработает через 3 сек и выведет в консоль слово

***setTimeout(() => {***

***console.log('Hello!');***

***}, 3000);***

***consol.log(\_\_dirname);*** // с помощью данного глобального объекта мы можем получить доступ до главного исполняющего файла. В результате мы получаем путь до файла

***console.log(\_\_filename);*** // возвращает весь путь до файла включая его имя и расширение

глобальный объект process хранит очень много полезной и нужной информации по типу конфигурации, переменных среды, версий и т.д. К любому из параметров можно получить доступ и использовать его при разработке

***console.log(`Hello, ${process.argv[2]}`);*** // argv - это массив, который содержит аргументы командной строки: 0 - это node (1-й элемент), 1 - имя исполняемого JavaScript файла (2-й элемент). Если при вызове передать несколько аргументов (например число, строку и массив), то при запуске эти значения будут доступны в массиве argv, т.е. используя терминал, в выполняемый скрипт мы можем передать какие-то значения, после чего в самом скрипте мы их можем использовать. Например, передав в терминале команду: node global Nemo, получим результат: Hello, Nemo, т.к. в команде заложили вывести 3-й элемент массива argv[2]

***console.log(process.env);*** // самый часто используемый. В этот объект добавляются переменные окружения по типу develop or production и в зависимости от них запускаются разные варианты окружений. Или установить переменную Puth:3000 и слушать приложение на этом определенном порту

еще один глобальный объект - url - с его помощью мы можем работать с получаемыми сервером url-адресами, осуществляя парсинг можно вытягивать из строки полезные данные, после чего работать с ними

***const url = new URL("https://webDev.com/path/name#test");*** // url - это конструктор, поэтому при создании внутрь него мы передаем адрес, Допишем произвольный путь path/name

***console.log(url.hostname);*** // используя .hostname мы можем получить сведения о хосте

***console.log(url.href);*** // получим полную ссылку

***console.log(url.pathname);*** // используя .pathname мы легко можем его получить

***console.log(url.hash);*** // дописав хэш в конце адреса, применяя соответствующие свойства, мы также получаем к нему доступ

Таким образом, распаршивая строку url-адреса с помощью различных методов, у нас есть доступ ко всем составляющим для комфортной работы по написанию логики

**Node.js #3 Модули (импорт и экспорт) (Modules & Require)**

**01test.js**

***const userName = "Nemo";***

***const sayHi = (userName) => `Hello, my name is ${userName}`;***

// module - это глобальный объект экспорта данных в другое место - создание модуля

***module.exports = {***

***userName,***

***sayHi,***

***};*** // свойству .exports присвоили объект (а можно одну переменную) {в котором описали экспортируемые данные}, чтобы использовать в другом месте

**03modules.js**

// чтобы произвести импорт экспортируемых данных нужно воспользоваться диррективой require('./путь\_до\_файла\_модуля'), для этого создаем константу, куда будет присвоено экспортируемое значение

//импортируем переменную из модуля, при этом воспользуемся деструктуризацией и из объекта modules вытащим оба параметра:

***const { userName: user, sayHi } = require("./1test");***

***const name = "Jonny";*** // сразу внутри файла определяем локальную константу name

***console.log(sayHi(name));*** // и передаем ее в импортируемую функцию sayHi

***module.exports = name;*** // если что-то нужно экспортировать именно из этого файла, то механизм такой же: добавляем module.exports и присваиваем значение, которое хотим экспортировать из модуля

***const os = require("os");*** // импортировать можно и сторонние библиотеки. Например симпортируем модуль os, который помогает получить различные сведения об операционной системе пользователя

***console.log(os.platform(), os.release());*** // метод .platform дает информацию о платформе, метод .release дает информацию о релизе

**Node.js #4 Node.js и файловая система (Node.js & File System)**

**(чтение, создание и удаление файлов и папок)**

**04app.js**

Для этого в Node.js есть специальный модуль filesystem или fs

***const fs = require("fs");*** // подключаем модуль fs используя директиву require

// Получение доступа к файлу 4test.txt и прочтение его содержимого:

***fs.readFile("./4test.txt", "utf8", (error, data) => {***

// используем метод .readFile(аргументы: 1 - путь до файла, 2 - callback или обычная функция, которая сработает, когда мы выберем метод. Она принимает 2 аргумента: error - этообъект ошибки, который будет содержать всю информацию о возникшей проблеме, если вдруг чтение файла произвести не удалось, data - это данные, которые мы получаем при чтении). readFile - это асинхронная функция. Для выведения данных в консоль достаточно написать: console.log(data); - для получения буфера или console.log(data.toString()); - для получения строк (либо указать в аргументах utf8 и использовать 1-й вариант)

***fs.mkdir("./files", () => {});*** // создаем папку, в которую хотим поместить новый файл (путь до создаваемой папки и callback)

Создадим новый файл test2.txt, который будет содержать те же данные, т.е. по сути являться копией файла 4test.txt. Для этого ВНУТРИ метода .readFile добавим метод .writeFile, который принимает 3 аргумента: 1-путь по которому будет создан новый файл и его имя, 2-данные, которые будут записаны, 3-callback функция

***fs.writeFile("./files/test2.txt", `${data} New text!`, (error) => {***

***error ? console.log(error) : null;*** // если есть ошибка(?)=(Да) - выводим в консоль, иначе (:)

***});***

***});***

***setTimeout(() => {***//создаем таймер на 4 сек., чтобы успеть видеть процесс))

***if (fs.existsSync("./files/test2.txt")) {***//метод проверяет существует ли файл

***fs.unlink("./files/test2.txt", (error) => {***//метод удаления файла

***error ? console.log(error) : null;***

***});***

***}***

***}, 4000);***

***setTimeout(() => {***

***if (fs.existsSync("./files")) {***//метод проверяет существует ли папка

***fs.rmdir("./files", (error) => {***//метод удаления папки

***error ? console.log(error) : null;***

***});***

***}***

***}, 6000);***