**Node.js**

**Установка сервера**

Выкачиваем инсталляху с основного сайта, ставим. Вместе с нодой обновляется и npm. Таким же образом сервер обновляется - просто ставим поверх новый.

Проверяем текущую версию ноды и npm через cmd:

node -v

npm -v

**Модули Node**

**CommonJS** -это стандартная библиотека для всех модулей Javacript

Node.js написан на CommonJS

Стандарты CommonJS:

* Каждый javascript файл - свой собственный модуль
* Переменные модуля предоставляют доступ к текущему расположению модуля в файле
* module.exports - переменная, которая определяет экспорт из текущего модуля:

module.exports = function () {

return {

perimeter: function(x,y) { return (2 \* (x + y)); },

area: function(x,y) { return (x \* y); }

};

}

Например, сохранили его под названием require.js. Использование:

var rect = require('./rectangle') // если лежит в той же директории

**require** - это функция, которая используется для импорта модуля

Альтернативный способ экспорта файла. Например, сохраним такой код в файл rect1.js

exports.perimeter = function(x,y) {

return (2 \* (x + y));

},

exports.area = function(x,y) {

return (x \* y);

}

// exports - алиас для module.exports

Теперь создаем второй файл, назовем его rectangle.js, там пишем код, который будет использовать эту функцию, и включаем саму функцию:

var rect = require('./rect1.js');

function solveRect(l, b) {

console.log("Area is " + rect.area(l, b));

console.log("Perimeter is " + rect.perimeter(l, b));

}

}

solveRect(5, 12);

Откываем терминал, переходим в нужную папку, где лежит файл, пишем

node rectangle // можно без расширения .js

**Типы файлов в модулях Node**

**Файловые модули (File-based)**

Это модули, которые находятся где-то в файлах и к ним необходимо указывать путь (примеры, рассмотренные выше). Позволяет создавать свои собстенные модули.

Подключаются через require('./module\_name'), если лежат в папке Node или можно указать полный путь

**Модули ядра (Core moules)**

Эти модули - часть ядра Node js. Спроектированы таким образом, что бы дать возможность подключать к Node модули от стронних разработчиков. Модули ядра включают Файловую систему (fs), Путь (Path) и другие

**Внешние модули**

Любые подключаемые сторонние модули, которые работают через Node:

* Модули сторонних разработчиков
* Исталляция через NPM
* node\_modules папка в Node

Подключаются через require('./module\_name'), Node начинает поиск с папки /node\_modules, постепенно поднимаясь выше до тех пор, пока модуль не будет найден

**Внутреннее устройство модуля**

Каждый модуль обычно имеет управляющий файл - index.js, так же он имеет package.json, в котором указываются все зависимости и другая информация.

**Версионность модулей**

Каждый модуль имеет свою версию, которая выражается в 3х цифрах, например 3.6.24.

Первая цифра - главная версия, которая не обязательно будет иметь обратную совместимость. Вторая цифра - минор версия, где были исправленны серьезные баги или добавлена новая функциональность

Третья цифра - багфиксы

Когда инсталлим через npm модули, можно указать:

npm install express@4.0.0 // будет заинсталлена конкретная версия 4.0.0

npm install express@"~4.0.0" // будет заинсталлена версия с новыми патчами

npm install express@"^4.0.0" // будет заинсталлена новая минорная версия

**Callbacks and error handling**

В Node есть так называемые Event Loop, который выполняет задачи асинхронно. Пришел запрос - попал в Event Loop, получил промис, что будет вызван callback, когда запрос выполнится, пока запрос выполняется - может прийти еще один запрос, и тоже получит промис.

Перепишем rectangle в стиле Node:

module.exports = function (x, y, callback) {

try {

if (x < 0 || y < 0) {

throw new Error("Rectangle params < 0: l=" + x + ", and b=" + y);

}

else

callback(null, {

perimeter: function (x, y) { return (2 \* (x + y)); },

area: function (x, y) { return (x \* y); }

});

}

catch (error) {

callback(error, null);

}

};

Модуль вызывает callback в любом случае. Если попалась ошибка, первым параметром в коллбек передается ошибка, а вторым, где должен быть return с полезной частью функции - null. Если же ошибки нет, вызывается callback с первым аргументом null - вместо ошибки.

Использование модуля:

// использование функции из модуля

var rect = require('./rectangle-2');

rect(l, b, function(err, rectangle){

if(err) { // если первым параметром придет ошибка - выводим в консоль

console.log(err);

} else {

console.log("Area is " + rectangle.area());

console.log("Perimeter is " + rectangle.perimeter());

}

});

Для ввода параметров через командеую строку, можно заинсталлить модуль yargs:

npm install yargs --save

Модуль сохраняется в текущую директорию, создавая подпапку node\_modules, в которой размещается сам со всеми зависимостями.

В js файле, в котором планируется его использовать, необходимо его включить:

//подключения модуля для ввода параметров через командную строку

var argv = require('yargs')

.usage('Usage: node $0 --l [num] --b [num]')

.demand(['l','b'])

.argv;

var rect = require('./rectangle'); // использование функции из модуля

function solveRect(l,b){

rect(l, b, function(err, rectangle){

if(err) { // если первым параметром придет ошибка - отобразить в консоли

console.log(err);

} else {

console.log("area: " + rectangle.area());

console.log("perimeter " + rectangle.perimeter());

}

});

}

solveRect(argv.l, argv.b);

После инсталла становится возможным вводить параметры:

node rect2 --l=2 --b=4