Google에 탑재된 V8이 자바스크립트 속도를 향상시켜주었다.

CommonJS : 자바스크립트를 사용하려면 CommonJS의 표준을 따라서 작성해라

가이드라인이 지정되어지게됬다.

멀티스레드 :

싱글스레드 : 절차문제로인행 오래걸릴수잇다.

=>이벤트기반의 동작방식으로 비동기방식 진행

번호표를 뽑아서 번호표 순서를 기다려 각각 필요한 물품이 나오면 가져오고 기다리는 시간을

최소한 시켜준다.

싱글스레드에서 이벤트 동작방식으로 진행을 하게 되면 멀티스레드만큼의 속도를 낼 수 있는

기능이 가능하다.

싱글스레드 기반 이벤트 방식 처리속도

최근의 웹어플리케이션은 서버내에서 수학적 연산을 처리하는 경우는 거의 존재하지않고

보통 요청을 받아 디비에 전달하는 방식으로 동작

초 고성능 서버 필요없이 최적의 성능을 유지할 수 있다.

자바어플리케이션도 우수하지만 Nodejs의 속도가 평균45% 빠르게 처리되어진다.

scale-up : 성능업, 예를 들어 상황에 따라 컴퓨터의 램을 더 추가시켜 속도, 성능을 업그레

이드

Nodejs는 cpu의 속도가 나쁠 떄 컴퓨터를 하나 더 추가시켜 cpu를 더 추가

mongoDB 성능을 올리지 않고 서버를 추가시켜서 cpu이용률을 낮춰준다.

=> scale-out --- 서버를 추가

Nodejs 설치 : http://nodejs.org

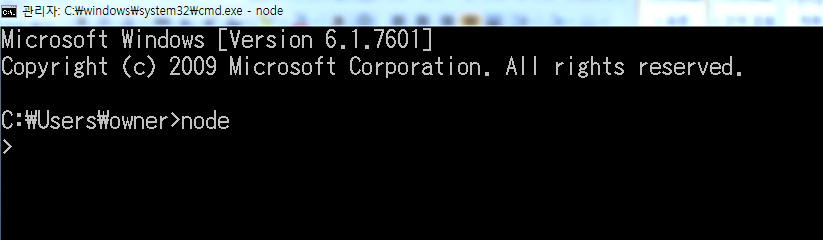
LTS : 안정화가 되어져 있는 버전

Current : 개발중인 버전이라 불안전한 상황

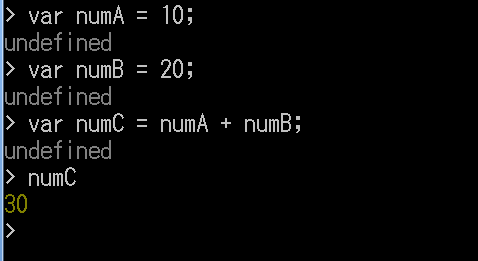
Nodejs 설치확인

cmd창 - node

클릭



nodejs가 설치되어져있으면 > 가 나온다



자바스크립트처럼 사용이가능

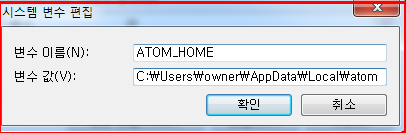
에디터 툴 WebStorm (유료) , Atom(무료)

폴더생성 D드라이브 node폴더

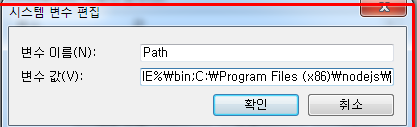
내컴퓨터 시스템고급속성 환경변수=>

ATOM\_HOME

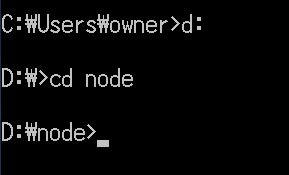
value = C:\Users\owner\AppData\Local\atom

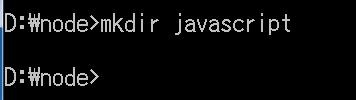


설정 :



Atom실행 : D드라이브 node폴더



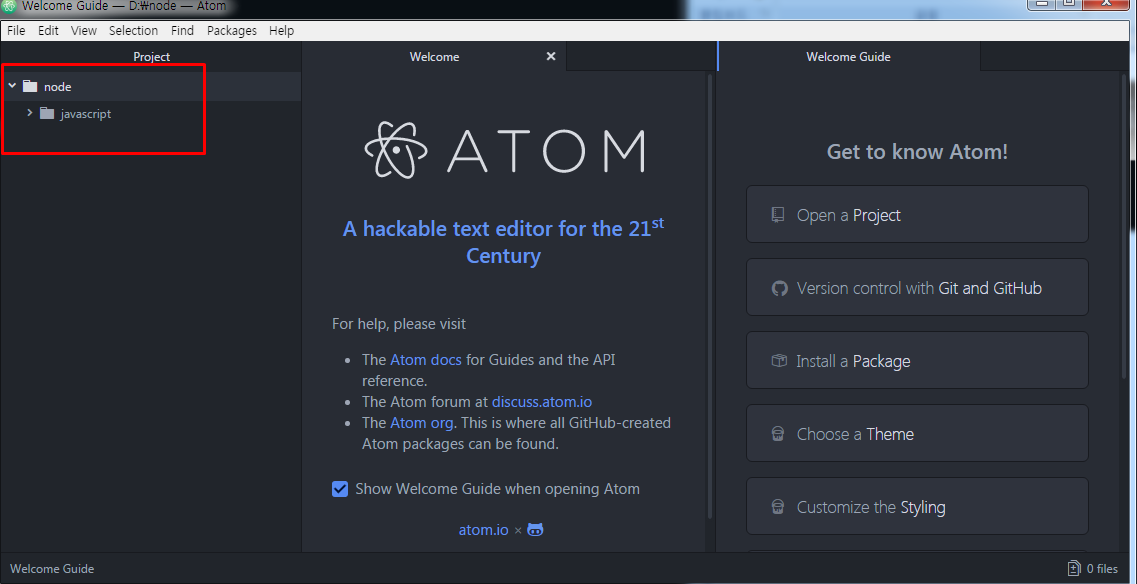


node폴더에 javascript폴더생성



D드라이브에서 아톰실행 atom .

아까 cmd에서 접속한 D드라이브 node폴더 javascript폴더에 접속되어져있다.



배열 :

값의 집합 => 관련된 값들을 하나로 묶어서 관리한다.

var scores = [100,90,100];

console.log(scores);

요소들을 가르키는 값 : 인덱스(index)

console.log(scores[0]); => 100

console.log(scores[3]); => undefined출력

undefined : null과 같다 정의되어져있지 않고 초기화되어져있지 않은값

배열요소 추가하기

scores.push(넘어주고싶은값);

배열 요소 제거하기

scores.pop(제거해주고싶은값); =>

중간의 값을 가져올순없다. 첫번쨰 값또는 제일 마지막값만 가능

배열요소 반복(for)

for(var i=0;i<socres.length;i++){ }

for-in 으로 배열요소 출력하기

for(var i in scores){ console.log(scores[i]; } scores[i]값을 출력 for(var in 인덱스)

ECMA script : 16년버전까지 나옴 ES7버전

=> commonJS의 표준을 따서 만든 스크립트

nodejs가 ES6까지만 적용이 되어진다.

nodejs에서느 for of

for(var of 값)

함수 : 일반함수 ===> 객체라고한다 배열또한 객체.

작업단위를 정하고 일부를 분리시켜 관리함

function 함수명(파라미터가능) { }

function sayHello(){

console.log("안녕하세요".):

}

함수 : 리턴함수

function 함수명(){

return 변수;

}

함수 : 매개변수가 있는 함수

1)

function getCalcNumbers(numberOne,numberTwo){

sayWelcome();

var result = numberOne + numberTwo;

return result;

}

var result = getCalcNumbers(10,60); => 70

2)

function getCalcNumbers(numberOne,numberTwo){

sayWelcome();

var result = numberOne + numberTwo;

return result;

}

var result = getCalcNumbers(10,60,70); => 70

=> 파라미터의 개수가 맞지 않더라도 함수는 정상적으로 실행되어진다.

앞에 있는 2개의 파라미터만 실행, 전송

Just In Time(JIT)

☆ ☆ ☆ ☆

함수호출시 바로 함수로가는게 아니라 자바스크립트 런타임 컴파일러로 먼저 간다.

arguments(배열)로 전달 파라미터값이 arguments.length =3 arguments.length[0] =10;

arguments.length[1]=60; arguments.length[2]=70; 이 값이 getCalcNumbers()전달

arguments값이넘어오고 arguments[2]를 파라미터에 선언해주어도 되고, 함수안에 arguments[2]를 호출해주어도 된다.

3)

파라미터가 선언되어져 있어도 값이 없으면 undefined처리된다

**숫자 +undefined => NaN**

함수 : 중첩함수 자기블럭에 잇는 함수가 우선이다.

함수안에 함수가 포함되어있는 형태

function get(num1,num2){

var v = calc(num1,num2);

function calc(num1,num2){

return num1+num2;

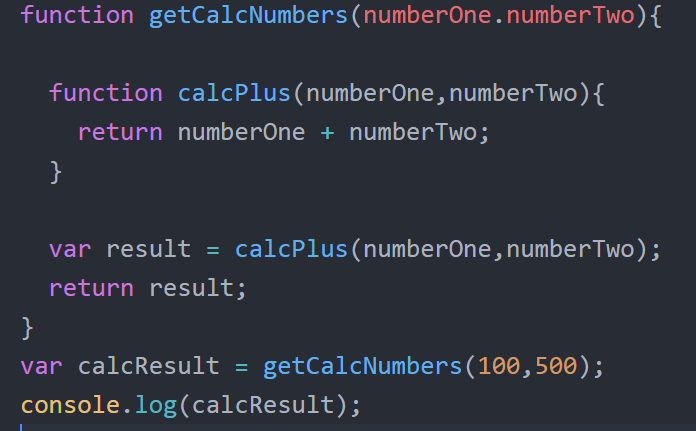
}

return result;

}

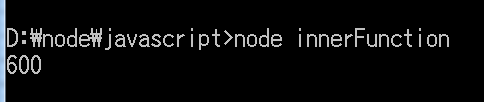
var result = get(10,60); ===> 70

**함수 실습**



node/javascript 폴더접속

node innerFunction 엔터



콜백 함수 : **함수가 언제실행되어지는지 모를때**

=>처리 후 작업

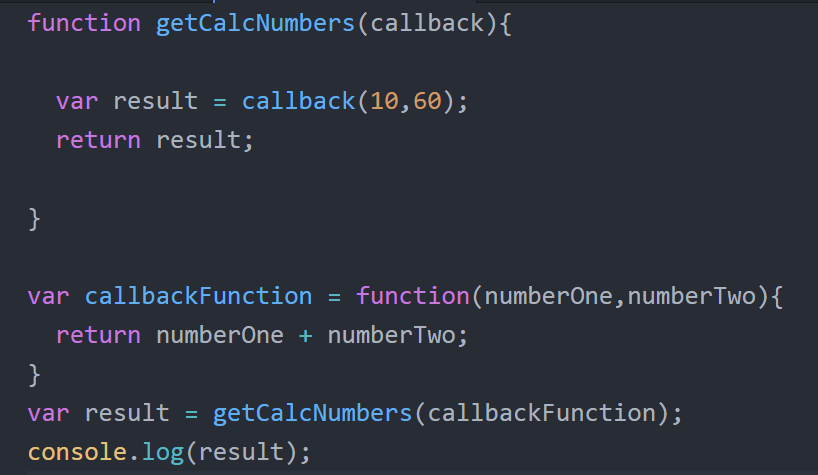
함수에서 특정 작업이 완료되었을때 추가로 실행해야 하는 작업을 기술한 함수

함수가 종료되는 시점이 불분명할때 콜백을 사용한다.

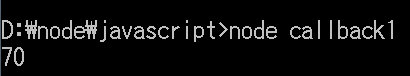
**=> 예) ajax Network 작업**

함수가 파라미터로 전달된다.

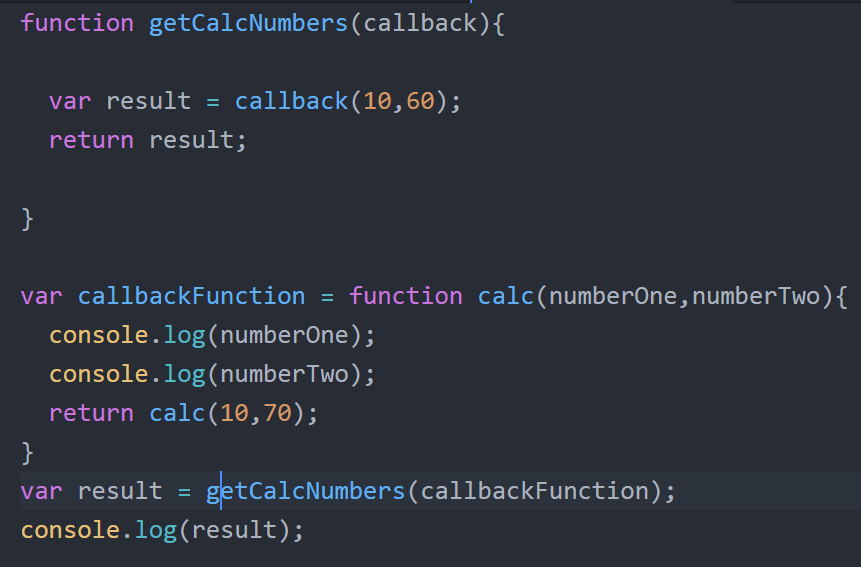
**실습 callback1.js생성 익명함수**



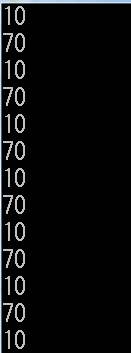
**실행**



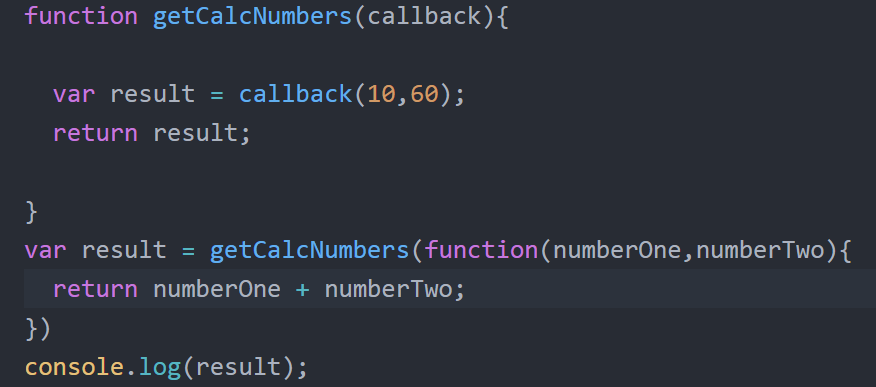
**callback2.js생성 익명함수에 함수명 추가**



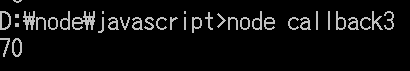
**무한루프에 빠진다.**

****

**callback3.js 생성**

****

**실행**

****

**함수 : 함수를 리턴하는 함수**

**중첩함수와 유사하지만 함수를 리턴하는 점이 다름**

( function (number){

return ++number;

} ) (10);

**변수의 영역**

function func(){

var a = 10;

if(a==10){

var c= a++;

var a=50;

}

var c = ++c; ===>12 출력

)

**자바스크립트에서의 변수의 영역**

javascript function기반의 영역을 사용함

function 내부에서 선언된 변수는 function의 모든 영역에서 사용가능하다

=> 호이스팅

function foo(){

var count =0;

var bar; => 똑같은 변수명이면 덮어씌워버린다.

if(true){

var bar = 10;

}

++bar; => 11

}

객체 : 리터럴 => JSON 형식과 같다(text)화

var object ={} 객체생성

object.name = "test test"; object객체에 name 필드추가

object.job = "Developer"; obejct객체에 job필드추가

console.log(object.name);

console.log(object.job);

console.dir(object);

1. object.name => name의 값을 알떄

2. object["name"] => object["test"]

=> var id="test";

=> object[id] ===> test출력

객체생성

1. object = {}

2. object ={ name : "----" , job : ".-----"}

Node JS

NPM : Node.js 모듈의 생태계

java진영의 maven과 같은 존재 노드용 외부모듈 <http://modulecounts.com>

NPM : <https://www.npmjs.com/>

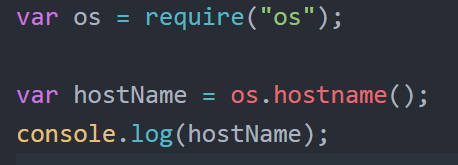
노드용 모둘관리 시스템

Modules

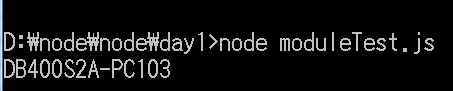
var 변수명 = require("모듈의이름");

모듈의 이름과 변수명은 해야한다.

require는 import의 의미이다.



**실습 :**



ES6에서 var변수를 const상수명으로 사용이 될수 있다.

const지역변수 개념을 사용할 수 있다.

let 는 var를 대신하는 변수지시자

