1. Node.js 문서
 1)Node는 매우 간결하고 가볍게 동작하고, module을 이용해서 필요한 기능을 로드해서 사용한다.
 2)필요한 module을 로딩
 3)다른 언어의 library이다.

2. Node API 문서
 1)https://nodejs.org/en/docs/
 2)https://nodejs.org/dist/latest-v6.x/docs/api/

3. Stability Index
 1)https://nodejs.org/dist/latest-v6.x/docs/api/documentation.html
 2)API에 나타나는 Stability는 API의 안정도를 의미한다.
 3)새로운 기능이 추가되면서 도입된 API는 충분한 테스트가 부족할 수도 있다.
 4)시간이 지나면서 많은 node.js 커뮤니티의 검증을 통해서 새로 추가된 기능을 안정화되는 단계로 전환된다.
 5)API 문서에는 이러한 성숙도를 Stability로 알려준다.
 6)단계가 높을 수록 안정적이다.
  -0 : Deprecated : 기능에 문제가 있어서 사용하지 않을 것을 권장
  -1 : Experimental : 시험용 기능으로 사라질 수도 있음.
  -2 : Stable : API가 안정화된 상태
  -3 : Locked : 심각한 문제가 발생하지 않는 이상, 변화없이 사용

4. Module구성
 1)module종류
  -Core module : 미리 compile된 상태로 library directory - 설치 불필요
   --위치 : node.js library directory
  -Extention module : npm으로 별도 설치
   --같은 폴더
   --node_modules 이름의 폴더
   --상위폴더의 node_modules

 2)module에는 class와 method, property, event가 정의돼 있다.

 3)e.g. Readline
  -Class : Interface
  -Method
  -Event

 4)Module loading
  -Module을 사용하려면 모듈을 로딩해야 하고, require함수를 사용한다.
  -require('모듈이름');
  -절대경로 혹은 상대 경로
   var readline = require('readline');

 5)객체 생성
  -module에 정의된 클래스는 객체를 생성하는 과정을 거쳐서 사용한다.
  -객체를 생성하는 방법은 모듈의 객체 생성 메소드를 이용한다.
   --var rl = readline.createInterface();
  -객체를 생성하고 나면 클래스에 정의된 메소드를 사용할 수 있다.
   --rl.setPrompt('>>');

 6)Event

```
   -event는 API 문서에서 정의되어있다.
   -event는 event가 발생했을 때 동작하는 listener 함수를 정의
    --rl.on([event_name],[listener 함수])
   -listener 함수의 파라미터는 API 문서의 해당 이벤트 항목에 설명이 있다.
    --rl.on('line', function(cmd){
      console.log('You just typed : ' + cmd);
    });

  7)Module 함수
   -객체 생성 없이 모듈에 직접 사용
    var readline = require('readline');
    readline.cursorTo(process.stdout, 60, 30);


5. Core Module
  1)Process : 프로세스에 대한 정보를 담고 있는 전역 객체
  2)Utility : 타입검사, 포매팅 등의 유틸리티 함수를 제공
  3)Events : 이벤트 관련 함수를 제공
  4)Buffers : 바이너리 데이터의 Octet Stream을 다루는 모듈
  5)Streams : Stream을 다루기 위한 추상 인터페이스
  6)Crypto : 암호화에 대한 함수 제공
  7)TLS/SSL : 공개키, 개인키 기반인 TLS/SSL에 대한 함수 제공
  8)File System : 파일을 다루는 함수 제공
  9)Path : 파일 경로를 다루는 함수 제공
  10)Net : 비동기 네트워크 통신 기능 제공
  11)UDP/Datagram Sockets : UDP의 데이터그램 소켓 통신 기능 제공
  12)DNS : 도메인 네임 서버를 다루는 함수 제공
  13)HTTP : HTTP 서버 및 클라이언트 기능 제공
  14)HTTPS : HTTPS 서버 및 클라이언트 기능 제공
  15)URL : URL을 다루는 함수 제공
  16)Query Strings : URL의 쿼리 문자열을 다루는 함수 제공
  17)Readline : 스트림에서 라인 단위로 읽는 기능을 제공
  18)Vm : 자바스크립트 실행 기능 제공
  19)Child Processes : 자식 프로세스 생성과 관련된 함수 제공
  20)Assert : 유닛 테스트를 위한 단언문을 제공
  21)TTY : 터미널이나 콘솔 관련 기능을 제공
  22)Zlib : zlib압축, 해제 함수 제공
  23)OS : 운영체제에 대한 정보를 구하는 함수 제공
  24)Cluster : 여러 노드 프로세스를 실행하는 클러스터 기능을 제공


6. 주요 Core Module
  1)프로세스 환경
   -os, process, cluster

  2)File과 경로, URL
   -fs, path, URL, querystring,stream

  3)Network Module
   -http, https, net, dgram, dns


7. 전역 객체
  1)소스 어디에서나 접근할 수 있는 객체
```

2)전역객체는 global이라는 이름으로 존재
3)별도의 모듈 로딩없이 사용
4)console.log()나 require(), setTimeout()등 소스어디에서나 불러사용할 수 있는 함수는 모두 global 객체가 제공하는 함수이다.
5)global.console.log()와 같이 사용해도 되지만, 보통 편의상 전역 객체는 생략이 가능

6)주요 전역 객체
-Process
-Console
-Buffer(class)
-require()
-__filename, __dirname --> console.log(__filename); console.log(__dirname);
-module
-exports
-Timeout 함수

8. Process module

1)Application 프로세스 실행 정보를 제공

2)주요 프로퍼티
-process.env : 실행 환경의 전반적인 정보
-process.version : Node.js 버전
-process.arch, process.platform : CPU 아키텍처와 플랫폼 정보
-process.stdout, stderr, stdin : 표준 입출력 스트림
-process.argv : 실행 명령 파라미터

3)주요 이벤트
-exit : 노드 어플리케이션이 종료되는 이벤트
-beforeExit : 종료 되기 전에 동작하는 이벤트
-uncaughtException : 예외 처리되지 않은 예외 발생 이벤트

4)함수
-process.exit([code]) : 어플리케이션 종료
-process.nextTick(callback[, arg][,...]) : 이벤트 루프 내 동작을 모두 실행한 이후에 콜백을 실행

5)Lab1 : processdemo.js

```
console.log(process.env);
console.log(process.arch);
console.log(process.platform);
```

6)Lab2 :
-In REPL

```
$ node
>process.execPath  //현재 node 실행 파일의 경로
'/usr/bin/node'
>process.cwd()     //현재 node 어플리케이션의 경로를 반환
'/home/Instructor/Eclispe'
>process.version
'v6.10.2'
>process.versions
{http_parset:'2.7.0',
  node: '6.10.2',
  v8: '5.1.281.98',
```

```
  uv: '1.9.1',
  zlib: '1.2.11',
  ares: '1.10.1-DEV',
  icu: '56.1',
  modules: '48',
  openssl: '1.0.2k' }
 >process.memoryUsage()
 {ress: 16883712,
  heapTotal: 7376896,     //V8의 메모리
  heapUsed: 5190736,
  external: 9066 }
 >process.uptime()
 364.514
 >process.exit()
 $

7)process.argv : 실행 파라미터 얻기
   process.argv.forEach(function(val, index, array){
     console.log(index + ' : ' + val);
   });
  $ node processdemo1.js  Banana Apple Lime
  ------------------------------------------------
   0 : /usr/bin/nodejs
   1 : /home/instructor/NodeHome/processdemo1
   2 : Banana
   3 : Apple
   4 : Lime

  $ node processdemo1.js  3  5
   var i = process.argv[2];   //3
   var j = process.argv[3];  //5
   var sum = parseInt(i) + parseInt(j);
   console.log(sum);           //8

8)Lab3 : processdemo2.js
   //표준 입력의 스트림은 멈춰져 있는 상태가 기본 동작이기 때문에
   //resume()을 실행하면 표준 입력에서 입력을 읽어 들일 수 있다.
   process.stdin.resume();
   process.stdin.setEncoding('utf8');

   //데이터가 새로 들어오면 이벤트 발생, 어플리케이션이 종료되면서 표준 입력이 종료될 때 end 이벤트 발생.
   process.stdin.on('data', function(chunk){
     process.stdout.write('data : ' + chunk);
   });

   process.stdin.on('end', function(){
     process.stdout.write('end');
   });
   ------------------------------------------------
   $ node processdemo2
   hi
   data : hi
   hello
   data : hello
```

```
    //Ctrl + D를 입력하면 end 이벤트 발생시킨다.
    //Ctrl + C는 프로세스를 종료시키기 때문에 end 이벤트가 발생하지 않는다.
    //Windows 에서는 process.stdin에서 Ctrl + D로 종료 이벤트가 발생하지 않는다.
    end $

 9)nextTick()
  - 노드는 직접 작성한 코드는 모두 동기로 실행하기 때문에 연산이 많은 작업을 하거나 아주 긴 반복문을 도는 중이라면 다른 이벤트가 발생하더라도 처리하지 못한다.
  - 이처럼 CPU 연산이 많이 필요한 작업을 비동기로 실행할 수 있게 하는 함수이다.
  - process.nextTick()에 등록된 콜백 함수는 바로 실행하지 않고 이벤트 큐에 등록한다.
  - 싱글 스레드가 현재 작업을 완료하고 다음 이벤트를 처리할 수 있는 때가 되면 process.nextTick()으로 등록한 콜백함수를 차례대로 호출한다.
  - 비동기로 실행하기 위해 setTimeout(function(){}, 0)와 같은 방법으로 사용하기도 한다.
  - 하지만, process.nextTick()이 더 효율적이고 더 빠르게 동작한다.
  - 그리고 이 함수로 등록한 콜백 함수는 우선순위가 높아 다른 IO의 콜백 함수보다 먼저 실행된다.

 10)Lab4 : nexttickdemo.js
    process.nextTick(function(){
      console.log('Called by nextTick() function()');
    });
    console.log('This message will display first');
    ------------------------------------------------
    $ node nexttickdemo
    This message will display first
    Called by nextTick() function()


9. Timers module
 1)함수
  -지연동작 : setTimeout
  -반복동작 : setInterval

 2)일정 시간 뒤 호출
  -setTimeout(callback, delay, arg,...)
  -clearTimeout()

 3)파라미터
  -callback : 함수 형태
  -delay : milli second
  -arg : callback 함수의 파라미터

 4)Lab1
    function sayHello(){
      console.log('Hello, World');
    }
    //3초뒤 실행
    setTimeout(sayHello, 3 * 1000);

    //타이머 취소
    var t = setTimeout(sayHello, 10);
    clearTimeout(t);

 5)반복
    setInterval(callback, delay, arg...)
```

```
    clearInterval();

  6)Lab2
    function sayGoodbye(who){
      console.log('Good bye', who);
    }
    setInterval(sayGoodbye, 1 * 1000, 'Friend');

  7)Lab3 : timeout.js
    function sayHello(){
      console.log('Hello, World');
    }

    //sayHello();
    /*setTimeout(function(){
        sayHello();
      }, 3 * 1000);*/     //3초뒤 실행

    setInterval(function(){
       sayHello();   //2초마다 sayHello 실행함.
    }, 2 * 1000);


10. Console module
  1)로그남기기
  2)실행 시간 측정
  3)로그남기기 예
    -console.log('log', 'log message');
    -console.info('info', 'info message');
    -console.warn('warn', 'warn message');
    -console.error('error', 'error message');

  4)값 출력
    var intValue = 3;
    console.log('intValue = ' + 3);

  5)객체형 출력
    var obj = {
      name : 'IU',
      job : 'Singer'
    }
    console.log('obj : ' + obj);    //obj : [object Object]
    console.log('obj : ', obj);     //obj : {name : 'IU', job: 'Singer'}

  6)Custom console
    -console type loading
      var Console = require('console').Console;
    -console object 생성
      new Console(stdout[,stderr])
    -파라미터: 출력 스트림
      --stdout : 표준 출력 스트림, info, log
      --stderr : 표준 에러 출력, warn, error

  7)파일로 로그 남기는 커스텀 콘솔
```

```
  var output = fs.createWriteStream('./stdout.log');
  var errorOutput = fs.createWriteStream('./stderr.log');
  var logger = new Console(output, errorOutput);

8)실행 시간 측정
  -콘솔 객체로 실행 시간 측정하기
  -시작 시점 설정하기
    console.time(TIMER_NAME);
  -종료 시점. 걸린 시간 계산해서 출력
    console.timeEnd(TIMER_NAME);

9)예제 코드
  //시간 측정 시간
  console.time('SUM');
  var sum = 0;
  for(var i = 1 ; i < 100000 ; i++){
    sum += i;
  }
  //시간 측정 끝
  console.timeEnd('SUM');

10)Lab1 : consoledemo.js
    var intVal = 3;
    var obj = {
       name : 'NodeJS',
       how : "Interesting"
    };

    console.log('Hello World');
    console.log('intVal : ' + intVal);
    console.log('obj : ' + obj);
    console.log('obj : ', obj);

11)Lab2 : customConsole.js
    var fs = require('fs');
    var output = fs.createWriteStream('stdout.log');
    var errorOutput = fs.createWriteStream('error.log');

    var Console = require('console').Console;
    var logger = new Console(output, errorOutput);

    logger.info('info message');
    logger.log('log message');

    logger.warn('warning');
    logger.error('error message');

12)Lab3 : consoledemo1.js
    console.time("TIMER");
    var sum = 0;
    for(var i = 1 ; i < 100000; i++){
       sum += i ;
    }
    console.log('sum : ', sum);
```

```
376       console.timeEnd("TIMER");
377
378
379  11. Util module
380    1)다른 API의 기능을 지원하는 유틸리티 기능은 util 모듈이 제공한다.
381
382    2)모듈 로딩
383      var util = require('util');
384
385    3)주요 기능
386      -문자열 포맷
387      -상속
388
389    4)문자열 포맷
390      util.format(format[, ..])
391      -format : 치환자(placeholder)를 포함한 문자열, 치환자는 두번재 이후의 파라미터로 입력한 값으로 치환되고, 타입에
         따라서 다음 문자를 사용한다.
392      -placeholder
393        --%s : String
394        --%d : Number(정수형이나 실수형)
395        --%j : JSON
396        --%% : '%' 문자 자체
397
398    5)사용예
399      var str = util.format('%d + %d = %d', 1,2 (1+2));
400      //1 + 2 = 3
401      var str1 = util.format('%s, %s, %j', 'Hello', 10, {name:'node.js'});
402      //Hello, 10, '{"name" : "node.js"}'
403
404    6)상속 : inherits
405      -두 클래스를 상속하도록 한다.
406      util.inherits(constructor, superConstructor)
407      util.inherits(ChildClassFunction, ParentClassFunction);
408
409    7)사용예
410      function Parent(){}
411      Parent.prototype.sayHello = function(){
412        console.log('Hello. from Parent Class');
413      }
414      function Child(){}
415      util.inherits(Child, Parent);
416
417      var child = new Child();
418      child.sayHello();
419
420    8)Lab1 : utildemo.js
421      var util = require('util');
422      var str1= util.format('%d + %d = %d', 1, 2, (1 + 2));
423
424      console.log(str1);
425
426      var str1 = util.format('%s, %s, %j', 'Hello', 10, {name:'node.js'});
427      //Hello, 10, '{"name" : "node.js"}'
428
```

```
9)Lab2 : inheritdemo.js
 /*
 function Parent(){
 }

 Parent.prototype.sayHello = function(){
    console.log('Hello World, from Parent Class!');
 }

 var obj = new Parent();
 obj.sayHello();

 function Child(){
 }

 var obj2 = new Child();
 obj2.sayHello();                //error 왜! 상속관계가 아니기 때문에
 */
 ---------------------------------------------------------------
 var util = require('util');

 function Parent(){
 }

 Parent.prototype.sayHello = function(){
    console.log('Hello World, from Parent Class!');
 }

 var obj = new Parent();
 obj.sayHello();

 function Child(){
 }

 util.inherits(Child, Parent);

 var obj2 = new Child();
 obj2.sayHello();                //Success


12. Events module
 1)node.js는 이벤트를 기반으로 동작한다.

 2)그래서 많은 node.js의 객체가 이벤트를 발생시킬 수 있고, 또한 발생한 이벤트에 반응해서 등록된 동작을 수행할 수 있다.

 3)node.js에서 이벤트를 발생시키고 이벤트에 반응하는 객체는 event 모듈에 정의된 EventEmitter이다.

 4)Node.js 어플리케이션의 이벤트들
  -event 예
    --클라이언트 접속 요청
    --소켓에 데이터 도착
    --파일 오픈/읽기 완료
  -event 처리
```

```
482      --비동기 처리
483      --리스너 함수
484
485   5)이벤트를 다룰 수 있는 : Readline 모듈
486     -Class:Interface
487      --rl.close();
488      --rl.pause();
489     -Events
490      --Event : 'close'
491      --Event : 'line'
492      --Event : 'pause'
493      --Event : 'resume'
494      --Event : 'SIGCONT'
495      --Event : 'SIGINT'
496
497   6)이벤트에 반응해서 동작하는 함수를 Event Listener 라고 하고 addListener나 on 함수를 이용해서 등록한다.
498
499   7)on이나 addListener로 등록한 리스너는 이벤트가 발생할 때마다 동작한다.
500
501   8)once로 등록하면 첫번째로 발생한 이벤트만 반응하고 다음부터는 반응하지 않는다.
502
503   9)타입에 정의된 이벤트 다루기
504     -이벤트 리스너 함수 등록
505      --emitter.addListener(event, listener)
506      --emitter.on(event, listener)
507      --emitter.once(event, listener)  //한번만 동작하는 리스너 등록
508     -사용예
509      process.on('exit', function(){
510        console.log('occur exit event');
511      });
512      //한번만 동작
513      process.once('exit', function(){
514        console.log('only once occur exit event');
515      });
516     -등록된 이벤트 리스너를 삭제하는 함수
517      //해당 이벤트에 등록된 개별 리스너 삭제
518      --emitter.removeListener(event, listener)
519      //해당 이벤트에 등록된 모든 리스너 제거
520      --emitter.removeAllListener([event])
521     -최대 이벤트 핸들러 갯수(기본 10개)
522      --emitter.setMaxListeners(n)
523      --emitter.getMaxListeners()
524      --EventEmitter.defaultMaxListeners
525     -실습
526      --어플리케이션 종료 이벤트
527        process.on('exit', function(code))
528
529   10)Lab1 : eventdemo.js
530      process.on('exit', function(code){
531        console.log('Exit event : ', code);  //0은 정상 전달
532      });
533      //Exit event : 0
534
535      process.once('exit', function(code){
```

```
      console.log('Exit event with once : ', code);  //0은 정상 전달
    });
    //Exit event with once : 0

11)Lab2 : eventdemo1.js
 -예외처리 되지 않는 상황 - 앱이 죽는 상황
    process.on('uncaughtException', uncaughtExceptionListener);
    //process 전역 객체의 이벤트 중 uncaughException는 예외처리 되지 않은 예외가 발생하는 이벤트이다.
    //이 이벤트 리스너를 이용하면 노드 어플리케이션에서 의도치 않은 에러 상황이 발생해도 크래쉬되지 않는다.

  sayHello();  //미리 정의되지 않았기 때문에 아래와 같은 오류 발생
  ------------------------------------------------------
  //ReferenceError : sayHello is not defined

  수정

  process.on('uncaughtException', function(code){
    console.log('uncaughtException');
  });

  sayHello();
  ------------------------------------------------------
  uncaughtException <--출력

12)Lab3 : uncaughtException.js
 process.on('uncaughtException', function(err){
   console.log('예외 : ' + err);
 });

 setTimeout(function(){
   console.log('이 코드는 실행됩니다');
 }, 500);

 //존재하지 않는 함수 실행
 nonExistentFunction();

 console.log('이 코드는 실행되지 않습니다');
 ------------------------------------------------------
 예외 : ReferenceError : nonExistentFunction is ot defined
 이 코드는 실행됩니다.

13)이벤트 발생
 -이벤트 발생시키기(emit)
   --emitter.emit(event[, arg1][, args2][,...])
   --event : event name
   --arg : 리스너 함수의 파라미터
   --emit 함수 호출 결과 : true(이벤트 처리), false(이벤트 처리 안됨)
 -예
   process.emit('exit');
   process.emit('exit', 0);  //리스너 함수의 파라미터로 0 전달

14)Lab4 : eventdemo2.js
   process.on('exit', function(code){
     console.log('Exit event : ', code);
```

```
590     });
591
592     process.once('exit', function(code){
593       console.log('Exit event with once : ', code);
594     });
595
596     process.emit('exit');
597     process.emit('exit', 0);
598     process.emit('exit', 1);
599     ------------------------------------------------
600     exit event : undefined
601     exit event with once : underfined
602     exit event : 0
603     exit event : 1
604     exit event : 0
605
606   15)커스텀 이벤트
607     -EventEmitter 객체에 커스텀 이벤트
608       var event = require('events');
609       var customEvent = new event.EventEmitter();
610
611       customEvent.on('tick', function(){
612         console.log('occur custom event');
613       });
614
615       customEvent.emit('tick');
616
617   16)커스텀 이벤트, 상속
618     -util 모듈을 이용해서 EventEmitter 상속
619       var Person = function(){}
620       //상속
621       var util = require('util');
622       var EventEmitter = require('events').EventEmitter;
623       util.inherits(Person, EventEmitter);
624
625       //객체
626       var p = new Person();
627       p.on('howAreYou', function(){
628         console.log('Fine, Thank you and you?');
629       });
630       //이벤트 발생
631       p.emit('howAreYou');
632
633
634 13. Path module
635   1)경로다루기
636     -경로에 관련된 모듈
637     -플랫폼에 따라 완전히 호환되지 않음.
638     -var path = require('path');   //$ REPL 에서 테스트할 것
639
640   2)경로를 다루는 기능을 제공한다.
641     -경로 정규화
642     -경로 생성
643     -디렉토리/파일 이름 추출
```

```
  -파일 확장자 추출

3)경로 정보
  -현재 실행 파일 경로, 폴더 경로
  -전역객체(global)
    --__filename : Node 어플리케이션 파일의 경로
    --__dirname : Node 어플리케이션 파일의 절대 경로
  -같은 폴더 내 이미지 경로
    --var path = __dirname + "/image.png';

4)경로 다듬기
  -경로 다듬기
    path.normalize();  //복잡하게 작성된 경로를 평범한 경로로 고쳐준다. 가령 '//'같은 실수를 고쳐준다.
  -경로 구성
    -'..' : 부모폴더
    -'.' : 같은 폴더
  -예제
    > path.normalize('/usr/tmp/../local///bin/');
    /usr/local/bin/

5)경로 구성 요소
  -경로 구성 얻기
    path.basename() : 파일 이름, 경로 중 마지막 요소
    path.dirname() : 파일이 포함된 폴더 경로
    path.extname() : 확장자

6)예
    > var pathStr = '/foo/bar/baz/asdf/quux.html';
    undefined
    > path.dirname(pathStr);
    '/foo/bar/baz/asdf'
    > path.basename(pathStr);
    'quux.html'
    > path.extname(pathStr);
    '.html'

7)경로 구성 객체
  -path.parse() : 경로 정보를 파싱해서 경로의 각 구성 요소로 이루어진 객체를 반환
    > var info = path.parse('/home/user/dir/file.txt');
    undefined
    > info
    {
       root : "/",
       dir : "/home/user/dir',
       base : "file.txt",
       ext : ".txt",
       name : "file"
    }
  //구성 요소 얻기
  > info.base
  'file.txt'
  > info.name
  'file'

```

```
8)경로 만들기
  -path.sep    //경로 구분자
    > path.sep
     '/'
  -pathUtil.join()   //경로 붙이기
    > pathUtil.join('home', 'outsider/nodejs');
     'home/outsider/node.js'
  -path.join([path1],[path2],[...])
    --파라미터로 전달받은 경로를 이어붙여 하나의 경로로 만든다.
    --파라미터는 원하는 만큼 추가할 수 있으며, 모두 문자열이어야 한다.
    > path.join('/foo', 'bar', 'bas/asdf','quux, '..')
    '/foo/bar/baz/asdf'
  -path.resolve();
    --전달받은 경로의 절대 경로를 반환
    > path.resolve('.');  //현재 위치의 절대 경로
    '/home/instructor/NodeHome'
    > path.resolve('../../', '/etc', 'resolve.conf');
    '/etc/resolve.conf'
  -path.relative() : 상대경로 표시
    > path.relative('../../', '.');
    'instructor/NodeHome'
  -path.dirname() : 전달받은 경로의 디렉토리명을 돌려준다.
    > path.dirname('/home/outsider');
    '/home'

9)경로만들기
  -경로 연산
    -- __dirname + pathUtil.sep + 'image.jpg';
    --현재 폴더 내 image.jpg

10)경로만들기
  -path.format()
    > var pathNew = path.format({
    ... root : '/',
    ... dir : '/home/user/dir',
    ... base : 'file.txt',
    ... ext : '.txt',
    ... name : 'file'
    ... });
    undefined
    > pathNew
    '/home/user/dir/file.txt'

11)Lab : pathdemo.js
  var pathUtil = require('path');

  var parsed = pathUtil.parse('/usr/tmp/local/image.png');
  console.log(parsed);

  console.log(parsed.base);
  console.log(parsed.exit);
  ---------------------------------------
    { root : '/',
       dir : '/usr/tmp/local',
```

```
        base : 'image.png',
        ext : '.png',
        name : 'image' }
    image.png
    undefined


14. File System module
 1)파일 시스템 모듈 : fs
  var fs = require('fs');

 2)주요 기능
  -파일 생성/읽기/쓰기/삭제
  -파일 접근성/속성
  -디렉토리 생성/읽기/삭제
  -파일 스트림
  -주의 : 모든 플렛폼에 100% 호환되지 않음.

 3)특징
  -같은 기능을 동기시과 비동기식 API 모두 제공
  -비동기식                    |          동기식
  callback 사용                | 이름규칙 : + Sync(readFileSync)
  Non-blocking 방식            | Blocking 방ㅎ식 - 성능상 주의
                               |  반환값 이용
                               | 동작이 모두 끝날때까지 다른 동작이 멈추는 방식

  -비동기식
    var data = fs.readFile('textFile.txt', 'utf8',
      function(error, data){
    });
  -동기식
    var data = fs.readFileSync('textFile.txt', 'utf8');

 4)에러 처리
  -동기식 : try ~ catch 사용
    try{
      var data = fs.readFileSync('none_exist.txt', 'utf-8');
    }catch(error){
      console.error('Readfile Error:', error);
    }
  -비동시식
    fs.readFile('none_exist.txt', 'utf-8', function(err, data){
      if(err) console.log('Readfile error', err);
      else 정상처리
    });

 5)파일 다루기
  -파일 디스크립터
  -파일 경로

 6)FileDescription 로 파일 다루기
  fs.read(fd, buffer, offset, length, position, callback);
  fs.readSync(fd, buffer, offset, length, position)
```

```
7)파일 경로로 파일 다루기
  fs.readFile(filename[, options], callback0;
  fs.readFileSync(filename[, options]);

8)파일 디스크립터
  -파일 디스크립터 얻기 : open 함수
    var fd = fs.openSync(path, flags[, mode])
    fs.open(path, flags[, mode], function(err, fs){
    });
  -flag
    r(읽기), w(쓰기), a(추가)
  -파일 닫기
    fs.close(fd, callback);
    fs.closeSync(fd);

9)파일 읽기
  -파일 내용 읽기
    fs.read(fd, buffer, offset, length, position, callback)
    fs.readFile(filename[, options], callback)
    fs.readFileSync(filename[, options])
  -파일 종류
    --문자열 읽기 : 인코딩
    --바이너리 읽기 : buffer
  -인코딩 설정 안하면 --> buffer

10)예제
  -파일 읽기 예제 - 파일 디스크립터, 동기식
    var fd = fs.openSync(file, 'r');
    var buffer = new Buffer(10);

    var byte = fs.readSync(fd, buffer, 0, buffer.length, 0);
    console.log('File Contents : ', buffer.toString('utf-8'));

    fs.closeSync(fd);  //파일 디스크립터 닫기

  -파일 읽기 예제 - 파일 디스크립터, 비동기식
    fs.open(file, 'r', function(err, fd2){
      var buffer = new Buffer(20);
      fs.read(fd2, buffer2, 0, buffer2.length, 10, function(err, byteRead, buffer){
        console.log('File Read : ', byteRead, 'bytes');
        console.log('File Content : ', buffer.toString('utf-8'));

        fs.close(fd, function(err){});
      });
    });

  -파일 읽기 - 동기식
    console.log('File Reading, with Encoding';
    var data = fs.readFileSync(file, 'utf-8');
    console.log(data);

    //바이너리 파일 읽기
    var imageData = fs.readFileSync('./image.jpg');
    console.log('Read Image File');
```

```
860     console.log(imageData);
861
862   -에러 처리 : try-catch
863
864   -파일 읽기 : 비동기, 인코딩
865     fs.readFile(file, 'UTf-8', function(err, data){
866       if(err){
867         console.error('File Read Error : ', err);
868         return;
869       }
870       console.log('Read Text File, UTF-8 Encoding');
871       console.log(data);
872     });
873
874   11)Lab1 : readfiledemo.js
875     var fs = require('fs');
876
877     fs.readFile('./helloworld.txt', 'utf-8', function(err, data){
878       if(err){
879         console.error('File Read Error : ', err);
880         return;
881       }
882       console.log('Read Text File, UTF-8 Encoding');
883       console.log(data);
884     });
885     ------------------------------------------------------
886     Read Text File, UTF-8 Encoding
887     file : Hello, world
888     Good Morning
889
890     만일 그런 파일이 없으면
891     fs.readFile('./hello.txt', 'utf-8', function(err, data){ <--수정
892     ------------------------------------------------------
893     File Read Error : { Error : ENOENT : no such file r...}
894
895     Lab : readfiledemo1.js   <-- 동기식
896     var fs = require('fs');
897
898     var data = fs.readFileSync('./helloworld.txt', 'utf-8');
899     console.log(data);
900     ------------------------------------------------------------
901     file : Hello, world
902     Good Morning
903
904     만일 에러처리까지 하려면
905     try{
906       var data = fs.readFileSync('./helloworld.txt', 'utf-8');
907       console.log(data);
908     }catch(error){
909       console.log('Error : ', error);
910     }
911
912   12) 파일 상태 확인
913     - 파일 다루기 : 파일 상태에 따라서 에러 발생
```

```
914    - 파일 다루기 전 : 파일 상태 확인
915
916   13)파일 상태 - 존재 확인
917    -fs.exists(path, callback) --> deprecated
918    -fs.existsSync(path)       --> deprecated
919    -fs.access(Sync) 사용
920    -fs.stat(Sync)
921
922   14)파일 접근 가능 확인
923    -파일 접근 가능 확인하기
924      --fs.access(path[, mode], callback)
925      --fs.accessSync(path[,mode])
926    -접근 모드
927      --fs.F_OK : 존재 확인
928      --fs.R_OK, W_OK, X_OK :읽기/쓰기/실행 여부 확인
929    -결론
930      --접근 불가능하면 에러 발생 : try-catch 사용
931
932   15)파일 접근 여부 확인 후 읽기
933    - 동기식
934      try{
935        fs.accessSync(file, fs.F_OK)
936        console.log('파일 접근 가능');
937        var data = fs.readFileSync(file, 'utf8');
938        console.log('파일 내용 : ', data);
939      }catch(exception){
940        //파일 없음
941        console.log('파일 없음 : ', exception);
942      }
943
944    -비동기식
945      fs.access(file, fs.F_OK | fs.R_OK, function(err){
946        if(err)
947          //에러처리
948        fs.readFile(file, 'utf8', function(err, data){
949          if(err)
950            //에러처리
951          console.log(data);
952        });
953      });
954
955   16)파일 상태
956    -파일 상태 얻기
957      fs.stat(path, callback);
958      fs.statSync(path);
959    -파일 상태 : fs.stats
960      --파일, 디렉토리 여부 : stats.isFile(), stats.isDirectory()
961      --파일 크기 :stats.size
962      --생성일/접근/수정일 : stats.birthtime, stats.atime, stats.mtime
963
964   17)파일 상태 확인 -동기식
965    try{
966      var stats = fs.statSync(file);
967      console.log('Create : ', stats.birthtime);
```

```
    console.log('Size : ', stats, size);
    console.log('isFile : ', stats.isFile());
    console.log('isDirectory : ', stats.isDirectory());
  }catch(err){
    console.log('파일 접근 에러', err);
  }

18)파일 상태 확인 - 비동기식
  fs.stat(file, function(err, stats){
    if(err){
      console.log('File Stats Error', err);
      return;
    }
    console.log('Create : ', stats.birthtime);
    console.log('Size : ', stats, size);
    console.log('isFile : ', stats.isFile());
    console.log('isDirectory : ', stats.isDirectory());
  });

19)Lab2 : stat.js
  var fs = require('fs');

  fs.stat('./stat.js', function(err, stats){
    if(err) throw err;
    console.log(stats);
    console.log('isFile : ' + stats.isFile());
  });
  ---------------------------------------------------------
  Stats {
    dev : 2054,
    mode : 33204,
    ...
    birthtime : ... }
  isFile : true

20)파일 상태 확인 후 읽기
  fs.stat(path, function(err, stats){
    if(stats.isFile(){
      fs.readFile(path, 'utf-8', function(err, data){
        console.log('파일 읽기 : ', data);
      });
    }
  });

21)Lab3 : fileAccessSyncDemo.js
  var fs = require('fs');
  var file = 'helloWorld.txt';
  try {
     fs.accessSync(file, fs.F_OK)
     console.log('파일 존재');
  }catch ( err ) {
     // 파일이 없을 때, 종료
     console.log('파일 존재하지 않음');
     process.exit(1);
```

```
  }
  // 파일에 내용 읽기
  try {
    var stats = fs.statSync(file)
    // console.log(stats);
    console.log('Create : ', stats['birthtime']);
    console.log('Size : ', stats['size']);
    console.log('isFile : ', stats.isFile());
    console.log('isDirectory : ', stats.isDirectory());
    console.log('isBlockDevice : ', stats.isBlockDevice());
    // 파일 읽기
    if ( stats.isFile() ) {
      var data = fs.readFileSync(file, 'utf-8');
      console.log('File Contents : ', data);
    }
  }catch ( err ) {
    console.error('File Error : ', err);
  }
  ---------------------------------------------------------------
  파일 존재
  Create : 2017-04-23T07:22:11.901Z
  Size : 33
  isFile : true
  isDirectory : false
  isBlockDevice : false
  File contents : File : Hello, World
  Good Morning

22)Lab4 : fileAccessAsyncDemo.js
  var fs = require('fs');
  var file = 'helloWorld.txt';

  fs.access(file, fs.F_OK, function(err) {
    if ( err ) {
      console.log('File Not Found');
      process.exit(1);
    }else {
      console.log('파일 존재');

      fs.stat(file, function(err, stats) {
        if ( err ) {
          console.error('File Stats Error', err);
            return;
        }

        console.log('Create : ', stats['birthtime']);
        console.log('Size : ', stats['size']);
        console.log('isFile : ', stats.isFile());
        console.log('isDirectory : ', stats.isDirectory());
        console.log('isBlockDevice : ', stats.isBlockDevice());

        if ( stats.isFile() ) {
          fs.readFile(file, function(err, data) {
            if ( err ) {
```

```
              console.error('File Read Error', err);
                return;
             }
             // encoding을 작성하지 않으면 Buffer로
             var str = data.toString('utf-8');
             console.log('File Contents : ', str);
           });
         }
       });
     }
  });
  ------------------------------------------------------------
  파일 존재
  Create : 2017-04-23T07:22:11.901Z
  Size : 33
  isFile : true
  isDirectory : false
  isBlockDevice : false
  File contents : File : Hello, World
  Good Morning

23)파일 저장
  -파일에 데이터 저장
    fs.write(fd, data, position[, encoding]], callback);
    fs.writeSync(filename, data[, options], callback);
    fs.writeFileSync(filename, data[,options])
  -파일에 데이터 저장
    --fd, filename :파일 디스크립터, 파일 경로
    --data : 문자열, 혹은 buffer
    --encoding : 문자열 저장 시 인코딩
  -같은 파일 이름 -> 덮어쓰기

24)Lab5 : writefiledemo.js

  fs.writeFie('./txtData.txt', 'Hello World', function(err){
    if(err){
      console.log('파일 저장 실패 : ', err);
      return ;
    }
    console.log('파일 저장 성공');
  });
  ------------------------------------------------------
  File Saver Success

25)파일에 추가
  -기존 파일에 내용 추가
    fs.appendFile(file, data[,options], callback);
    fs.appendFileSync(file, data[, options])
  -파일이 없으면 --> 새 파일 생성

26)파일 추가 예제
  fs.appendFile(path, 'Additional data', function(err){
    if(err) console.log('파일 내용 추가 실패 : ', err);
    console.log('파일 내용 추가 성공');
```

```
1130    });
1131
1132    27)파일 삭제
1133    fs.unlink(path, callback), fs.unlinkSync
1134    -파일이 없으면 에러
1135
1136    28)파일 삭제 예제
1137    fs.unlink('./binaryData.dat', function(err){
1138      if(err)
1139        console.error('Delete Error : ', err);
1140    });
1141
1142    29)파일 이름 변경/이동
1143    fs.rename(oldPath, newPath, callback)
1144    fs.renameSync(oldPath, newPath);
1145
1146    30)Lab6 : fileRenameDemo.js
1147    var fs = require('fs');
1148
1149    fs.rename('./helloworld.txt', './demo.txt', function(err){
1150      if(err) throw err;
1151      console.log('Success');
1152    });
1153    ------------------------------------------------------
1154    Success
1155
1156    31)파일 변경 사항 감시하기
1157    -fs.watchFile(filename, [options], listener)
1158    -filename : 감시할 파일
1159    -options :
1160      --persistent : false(프로세스 바로 종료), true(기본값)
1161      --interval : Linux에서 파일을 모니터링하는 inotify를 이용할 수 없을 때 수정여부를 확인할 간격을 밀리초 단위로
          지정
1162    -listener : 파라미터로 변경 이전의 파일과 이후 파일에 대한 fs.stats 객체를 받는다.
1163    -Windows 에서 실행가능하지 않을 수 있다. v6.x까지 지원하지 않았다.
1164
1165    32)Lab7 : watchfile.js
1166    var fs = require('fs');
1167
1168    fs.watchFile('./demo.txt',
1169      {persistent : true, interval : 0},
1170      function(curr, prev){
1171        console.log('현재 파일의 수정시간 : ' + curr.mtime);
1172        console.log('이전 파일의 수정시간 : ' + prev.mtime);
1173      }
1174    );
1175    ------------------------------------------------------
1176    $ node watchfile.js
1177    이렇게 해 놓고 새 터미널을 열고 demo.txt의 내용을 수정한다.
1178    그러면 아래와 같이 나타난다.
1179    현재 파일의 수정시간 : Sun Apr 23 ...
1180    이전 파일의 수저시간 : Sun ...
1181
1182    33)디렉토리 다루기
```

```
1183    -디렉토리 생성
1184      --같은 이름의 디렉토리가 있으면 실패
1185      fs.mkdir(path[, mode], callback), fs.mkdirSync
1186    -디렉토리 삭제
1187      --디렉토리가 비어있지 않으면 실패
1188      fs.rmdir(path, callback0, fs.rmdirSync
1189
1190   34)디렉토리 다루기 예제
1191    fs.mkdir('testdir', function(err){
1192      if(err){
1193        console.log('mkdir error : ', err);
1194        return;
1195      }
1196    });
1197
1198    try{
1199      fs.rmdirSync('test');
1200    }catch(error){
1201      console.log('디렉토리 삭제 에러');
1202    }
1203
1204   35)디렉토리 다루기
1205    -디렉토리 내 파일 목록
1206      fs.readdir(path, callback), fs.readdirSync
1207    -디렉토리가 없으면 에러
1208
1209   36)Lab8 : readDirDemo.js
1210    fs.readdir(path, function(err, files){
1211      if(err){
1212        console.error('디렉토리 읽기 에러 : ', err);
1213        return;
1214      }
1215      console.log('디렉토리 내 파일 목록(Async)\n', files);
1216    });
1217    ------------------------------------------------
1218    Directory File List(Async)
1219    [
1220    ...
1221    ]
1222
1223   37)파일 스트림
1224    -스트림 만들기
1225      fs.createReadStream(path[, options])
1226      fs.createWriteStream(path[, options])
1227
1228
1229  15. Buffer
1230   1)개요
1231    -JavaScript는 문자열을 다루는 기능을 제공하지만, binary 데이터를 다루는 기능은 없다
1232    -TCP Stream 이나 File Stream을 사용하려면 Octet Stream을 다룰 수 있어야 한다.
1233    -Octet는 8bit로 이뤄진 단위를 의미한다.
1234    -Node.js는 Octet stream을 다루는 함수를 전역 객체인 Buffer 클래스로 제공한다.
1235    -Buffer : 바이너리 데이터를 다루는 모듈
1236    -Row Data는 모두 Buffer 클래스에 저장된다.
```

-Buffer는 정수의 배열인데, 각 정수는 V8 Heap Memory 밖의 Row Memory에 할당된 주소를 가리킨다.
-Socket 간에 송신되는 데이터는 기본적으로 바이너리 포맷의 버퍼로 전송된다.
-버퍼대신 문자열을 보내고 싶으면 소켓에 직접 setEncoding 을 호출하거나 소켓에 쓰는 함수 내의 인코딩을 지정하면 된다.
-기본적으로 TCP의 socket.write 메소드는 두번째 파라미터를 utf8로 설정하지만, TCP createServer 함수의 connectionListener 콜백에서 반환되는 소켓은 데이터를 문자열이 아닌 버퍼로 전송한다.

2)JavaScript 문자열과 버퍼 사이에 변환을 하려면 encoding을 지정해야 하고, 다음과 같은 인코딩 방법을 사용한다.
-별도로 지정하지 않으면 UTF-8이 사용된다.
-ascii
--7비트 ASCII 데이터로 아주 빠르다.
--7비트보다 높은 비트가 설정되어 있으면 제거한다.
--null 문자인 '\0'이나 '\u0000'을 공백 문자인 0x20으로 변환한다.
--null 문자인 0x00으로 변화하고 싶다면 utf8 인코딩을 사용해야 한다.
-utf8
-- 멀티바이트로 인코딩된 유니코드 문자이다.
-ucs2
--2바이트 little endian으로 인코딩된 유니코드 문자
-base64
--Base64 문자열 인코딩
-hex
--각 바이트를 2개의 16진수로 인코딩한다.
-binary
--각 글자의 첫 8비트만 사용해 로우 데이터를 문자열로 인코딩하는 방법이지만, 이 인코딩은 폐기됐으므로 가능하면 사용하지 않는 것이 좋다
--현재는 존재하지만 차후 노드 버전에서는 제거될 수 있다.
-사용할 수 없는 인코딩을 설정하면 다음과 같은 에러가 발생한다.
TypeError : Unknown encoding: base63

3)global 이므로 별도의 로딩 불필요

4)버퍼 얻기
-파일에서 읽기
```
var filebuffer = fs.readFileSync('image.jpg');
```
-네트워크에서 얻기
```
socket.on('data', function(data){
  //data - buffer
});
```

5)버퍼만들기
-생성후 크기 변경 불가
--new Buffer(size)
--new Buffer(array)
--new Buffer(str[, encoding])

6)Lab1
```
$ node
> new Buffer(10)
<Buffer 28 02 el dd 39 7f 00 00 c0 53>
> new Buffer([1,2,3])
<Buffer 01 02 03>
> new Buffer('string', encoding='utf8')
<Buffer 73 74 72 69 6e 67>
```

```
7)버퍼 다루기 - 모듈함수
  -바이트 길이 : Buffer.byteLength(string[, encoding])
  -비교 : Buffer.compare(buf1, buf2)
  -붙이기 : Buffer.concat(list[,totalLength])
  -버퍼확인 : Buffer.isBuffer(obj)
  -인코딩 : Buffer.isEncoding(encoding)
  -다수의 버퍼를 덧붙여서 새로운 버퍼 생성 : Buffer.concat(list[, totalLength])

8)버퍼 다루기 - 객체 메소드
  -길이 : buffer.length
  -채우기 : buffer.fill(value[, offset][,end])
  -자르기 : buffer.slice([start[, end])
  -비교하기 : buffer.compare(otherBuffer)
  -복사하기 : buffer.copy(targetBuffer[, tagetStart][, sourceStart][, sourceEnd])


9)문자열과 버퍼
  -문자열 : 바이너리 데이터로 다루기
  -문자열에서 버퍼 생성
    new Buffer(str[, encoding])
  -문자열 인코딩 필요
    --ascii, utf8,...
  -잘못된 인코딩 -> 에러
  -버퍼에 문자열 쓰기
    buf.write(string[, offert][, length][, encoding])
  -변환
    buf.toString([encoding][, start][, end])

10)Lab2
  $ node
  > var buf = new Buffer(256)
  undefined
  > var len = buf.write('\00bd + \u00bc = \u00be', 0);
  undefined
  >  console.log(len + " bytes : " + buf.toString('utf8', 0, len));
  12 bytes : 1/2 + 1/4 = 3/4
  undefined

  > var str = 'node.js'
  undefined
  > var buf = new Buffer(str.length);
  undefined
  > for(var i = 0 ; i < str.length ; i++){
  ... buf[i] = str.charCodeAt(i);
  ... }
  115
  > buf.toString()
  'node.js'
  > Buffer.isBuffer(buf);   //객체가 버퍼 타입인지 검사
  true
  > buf.length      //버퍼 크기(버퍼 객체에 할당된 메모리의 크기)
  7

11)문자열과 버퍼
```

```
  -문자열에서 버퍼 생성
    var strBuffer =  new Buffer('Hello, World');
    strBuffer.toStrinig('utf-8');
    strBuffer.toStrinig('base64');   //SGVsbG8gV29ybGQ=
  -버퍼에 문자열 작성
    var buffer = new Buffer(10);
    //버퍼에 문자열 쓰기
    buffer.write('Hello World');
    buffer.toString();   //Hello Worl

12)문자열과 버퍼
  -문자열의 바이트 길이
    Buffer.byteLength(string[, encoding])

    var str1 = 'Hello World';
    str1.length //11
    Buffer.byteLength(str1);   //1

    var str2 = '그림이미지를 넣을 것';
    str2.length;   //4
    Buffer.byteLength(str2);   //8

13)버퍼 - 데이타 읽기/쓰기
  -버퍼는 문자열 외 다른 데이터의 타입에 따른 쓰기 함수를 제공한다.
  -데이터의 타입과 포맷(Big Endian, Little Endian)에 따라서 함수의 이름이 다르다.
  -noAssert 파라미터는 value와 offset의 범위 검사 기능 사용 여부를 부울 값으로 입력하고, 기본값은 false이다.
  -데이터 읽기/쓰기
    buf.readInt8(offset[, noAssert])
    buf.writeInt8(value, offset[, noAssert])
  -16비트 크기의 정수형 데이터 읽기/쓰기
    buf.readUInt16LE(offset[, noAssert])
    buf.writeUInt16LE(value, offset[, noAssert])
  -실수형 데이터 읽기/쓰기
    buf.writeFloatLE(value, offset[, noAssert])
    buf.writeFloatBE(value, offset[, noAssert])
    buf.readFloatLE(offset[, noAssert])
    buf.readFloatBE(offset[, noAssert])

14)Endian
  requre('os').endianness()

15)버퍼쓰기(value, offset)
  buffer.writeInt8(0,0);   //01
  buffer.writeUInt8(0xFF, 1);  //FF
  buffer.writeUInt16LE(0xFF, 2);  //FF 00
  buffer.writeUInt16BE(0xFF, 4);   //00 FF

16)버퍼읽기
  buffer.readInt8(0)  //1
  buffer.readUInt8(1)  //1
  buffer.readUInt16LE(2)  //255
  buffer.readUInt16BE(4)  //255

17)Lab: bufferdemo.js
```

```
    console.log('endian : ', require('os').endianness());

    var buffer = new Buffer(6);

    buffer.writeInt8(1, 0); // 01
    buffer.writeUInt8(0xFF, 1); // FF
    buffer.writeUInt16LE(0xFF, 2); // FF 00
    buffer.writeUInt16BE(0xFF, 4); // 00 FF

    console.log('HEX : ', buffer.toString('hex'));

    console.log(buffer.readInt8(0)); // 1
    console.log(buffer.readUInt8(1)); // 255
    console.log(buffer.readUInt16LE(2)); // 255
    console.log(buffer.readUInt16BE(4)); // 255
    ------------------------------------------------
    Windows 10에서
    endian : LE
    HEX : 01ffff0000ff
    1
    255
    255
    255

16. Stream module
 1)데이터의 전송 흐름
  -콘솔 입력/출력
  -파일 읽기/출력
  -서버/클라이언트 - 데이터 전송

 2)스트림 모듈
  -스트림을 다루기 위한 추상 인터페이스
  -다양한 스트림을 같은 인터페이스로 다룰 수 있다.

 3)스트림 종류
  -읽기 스트림 : Readable Stream
  -쓰기 스트림 : Writable Stream
  -읽기/쓰기 : Duplex
  -변환 : Tranform

 4)Readable Stream
  -읽기 스트림 : Readable
  -모드 : flowing, paused
  -flowing mode
    --데이터를 자동으로 읽는 모드
    --전달되는 데이터를 다루지 않으면 데이터 유실
  -paused mode
    --데이터가 도착하면 대기
    --read()함수로 데이터 읽기

 5)Readable Stream 메소드
  -읽기
    --readable.read([size])
    --readable.setEncoding(encoding)
```

```
1450        data이벤트가 Buffer대신 문자열을 사용하게 만든다.
1451        encoding 은 utf8, ascii, base64를 사용할 수 있다.
1452      -중지/재개
1453        --readable.pause()
1454          들어오는 data 이벤트를 멈춘다.
1455        --readable.resume()
1456          pause()로 멈춘 data 이벤트를 다시 받기 시작한다.
1457      -파이프
1458        --readable.pipe(destination[, options])
1459          스트림에서 읽어 들인 내용을 destination 에 지정된 쓰기 스트림에 연결한다.
1460          이 함수는 destination 스트림을 돌려주고 destination 스트림에서 end() 이벤트가 호출되어 쓸 수 없는 상태가
              되면 소스 스트림에서도 end 이벤트가 발생한다.
1461          options 에 {end : false} 를 전달하면 destination 스트림을 열린 상태로 유지한다.
1462        --readable.unpipe([destination])
1463
1464    6)Readable Event
1465      -readable : 읽기 가능한 상태
1466        --스트림이 읽을 수 있는 상태인지 알려준다
1467        --기본적으로 true 이지만 error 이벤트가 발생하거나 end 이벤트가 발생하면 false로 바뀐다.
1468      -data
1469        --읽을 수 있는 데이터 도착할 때 발생
1470        --기본적으로 Buffer 클래스를 이용하지만 setEncoding()이 사용되었다면 문자열을 사용한다.
1471        --콜백함수는 function(data)이다.
1472      -end
1473        --더 이상 읽을 데이터가 없는 상태
1474        --즉, 스트림이 EOF나 FIN을 받았을 때 발생
1475        *FIN(end,
              http://blog.joins.com/media/folderListSlide.asp?uid=swift&folder=5&list_id=11624597)
1476        --이 이벤트가 발생하면 더 이상 data 이벤트가 발생하지 않음을 의미하지만, 스트림이 쓰기도 가능하다면 쓰기는 여전히
              가능하다.
1477        --콜백함수는 function(){} 이다.
1478      -close
1479        --사용하는 파일 디스크립터가 닫혔을 때 발생한다.
1480        --모든 스트림이 이 이벤트를 사용하지는 않는다.
1481        --예를 들어, HTTP 요청은 종료 시점을 알 수 없으므로 close를 발생시키지 않는다.
1482      -error
1483        --데이터를 받는 동안 에러가 있을 때 발생한다.
1484        --콜백함수는 function(exception){} 이다.
1485
1486    7)flowing mode
1487      -data 이벤트 구현
1488      -pipe 연결
1489      -resume() 호출
1490
1491    8)Stream에서 읽기
1492      -파일 스트림에서 읽기 : flowing mode
1493        var is = fs.createReadSteam(file);
1494        is.on('readable', function(){
1495          console.log('==READABLE EVENT');
1496        });
1497
1498        is.on('data', function(chunk){
1499          console.log('==DATA EVENT');
1500          consiole.log(chunk.toString());
```

```
    });

    is.on('end', function(){
      console.log('==END EVENT');
    });
  -파일 스트림에서 읽기 : paused mode
    var is = fs.createReadStream(file);

    //data 이벤트가 없으면 pause mode
    is.on('readable', function(){
      console.log('==READABLE EVENT');

      //10byte 씩 읽기
      while(chunk = is.read(10){
        console.log('chunk ; ', chunk.toString());
      }
    });

9)Writable Stream
  -Writable Stream : 데이터 출력
  -예
    --http 클라이언트의 요청
    --http 서버의 응답
    --파일 쓰기 스트림
    --TCP 소켓

10)Writable Stream
  -메소드
  -데이터 쓰기, 인코딩
    --writable.setDefaultEncoding(encoding)
    --writable.write(chunk[, encoding][, callback])
      string 문자열을 encoding 으로 인코딩해 스트림에 쓴다.
      문자열이 커널 버퍼로 flush 되면 true를 리턴하고 커널버퍼가 가득 찼으면 false 를 리턴한다
      커널 버퍼가 다시 비워졌을 때 drain 이벤트가 발생한다.
      옵션 파라미터인 fd는 파일 디스크립터를 의미한다.
      문자열 대신 버퍼를 쓰려면 stream.write(buffer)를 사용한다.
  -Stream 닫기
    --writable.end([chunk][, encoding][, callback])
      EOF나 FIN을 스트림을 종료한다.
      큐에 추가된 데이터가 있으면 종료하기 전에 모두 내보낸다.
      end() 는 종료하면서 데이터를 쓰기 위한 end(string, encoding)과 end(buffer)도 사용할 수 있다.
  -버퍼
    --writable.cork()
    --writable.uncork()

11)Writable Stream Event
  -drain
    --출력 스트림에 남은 데이터를 모두 보낸 이벤트
    --write()메소드가 false를 돌려준 후 스트림이 다시 쓸 수 있는 상태가 되었음을 알리기 위한 이벤트이다.
    --콜백함수는 function(){} 이다.
  -error
    --스트림에서 에러가 발생하면 발생하는 이벤트이다.
    --콜백함수는 function(exception){} 이다.
  -finish : 모든 데이터를 쓴 이벤트
```

```
  -pipe
    --읽기 스트림과 연결(pipe)된 이벤트
    --Readable Stream의 pipe 함수로 스트림이 전달 되었을 때 발생한다.
  -unpipe : 연결(pipe) 해제 이벤트
  -close
    --사용하는 파일 디스크립터가 닫히면 발생한다.
    --콜백함수는 function() {} 이다.

12)파일 기반의 출력 스트림에 쓰기
  var fs = require('fs');
  var os = fs.createWriteStream('output.txt');
  os.on('finish', function(){
    console.log('==FINISH EVENT');
  });

  os.write('1234\n');
  os.write('5678\n');

  os.end('9\n'); //finish event

13)표준 입출력 스트림
  -process.stdin : 콘솔 입력
  -process.stdout : 콘솔 출력

14)스트림 연결
  -스트림 연결과 해제(Readable)
    --readable.pipe(destination[, options])
    --readable.unpipe(destination])
  -연결 이벤트(Writable)
    -pipe
    -unpipe
  -스트림 연결
    --입력 스트림 : stdin
    --출력 스트림 : 파일

15)스트림 연결 예제
  var fs = require('fs');
  var is = process.stdin;
  var os = fs.createWriteStream('output.txt');

  os.on('pipe', function(src){
    console.log('pipe event');
  });
  //exit입력이 오면 파이프 연결 해제
  is.on('data', functiuon(data){
    if(data.toString().trim() == 'exit'){
      is.unpipe(os);
    }
  });

  is.pipe(os);

16)Lab : stream.js
  var fs = require('fs');
```

```
  var os = fs.createWriteStream('./output.txt');
  os.on('finish', function() {
    console.log('finish!');
  });

  os.write('1234');
  os.write('4567');
  os.end('89');
  ------------------------------------------
  output.txt에 123456789 들어감

  var fs = require('fs');
  var os = fs.createWriteStream('./output1.txt');
  os.on('finish', function() {
    console.log('finish!');
  });

  // 키보드에서 입력한 내용
  var is = process.stdin;
  // 아웃풋 스트림(파일)로 연결
  is.pipe(os);
  ------------------------------------------
  output1.txt에는 키보드에 입력한 내용이 파일로 전송

17. URL module
 1)네트워킹
  -네트워킹의 시작
    --서버주소
    --서버에서 요청 위치
    --서버에서 리소스의 위치
  -URL : Uniform Resource Locator
    --http://nodejs.org/api/
    --http://nodejs.org/api/http.html
    --http://nodejs.org/api/http.html#http_event_connect

 2)URL 구성요소
  -protocol
  -host
  -post
  -path
  -query
  -fragment
  -http:www.google.com/search?q=iphone&format=json
   scheme     host                    query
  -http://images.apple.com/mac/home/images/tap_hreo_macpro_2x.jpg
   scheme     host                    path

 3)URL 모듈
  -var url =  require('url');
  -url.parse(urlStr[, parseQueryStrinig][, slashesDenoteHost])
    --urlStr : URL 문자열
    --parseQueryString : 쿼리 문자열 파싱, 기본값 false
    --slashesDenoteHost : //로 시작하는 주소의 경우, 호스트 인식 여부, 기본값 false
```

```
$ node
> var url = require('url');
undefined
> url.parse('http://domain/tags/search?q=node.js&page=2&year=2011');
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'domain',
  port: null,
  hostname: 'domain',
  hash: null,
  search: '?q=node.js&page=2&year=2011',
  query: 'q=node.js&page=2&year=2011',
  pathname: '/tags/search',
  path: '/tags/search?q=node.js&page=2&year=2011',
  href: 'http://domain/tags/search?q=node.js&page=2&year=2011' }
>

> url.parse('http://domain/tags/search?q=node.js&page=2&year=2011', true);
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'domain',
  port: null,
  hostname: 'domain',
  hash: null,
  search: '?q=node.js&page=2&year=2011',
  query: { q: 'node.js', page: '2', year: '2011' },
  pathname: '/tags/search',
  path: '/tags/search?q=node.js&page=2&year=2011',
  href: 'http://domain/tags/search?q=node.js&page=2&year=2011' }
>
> url.parse('//domain/search', false);
Url {
  protocol: null,
  slashes: null,
  auth: null,
  host: null,
  port: null,
  hostname: null,
  hash: null,
  search: null,
  query: null,
  pathname: '//domain/search',
  path: '//domain/search',
  href: '//domain/search' }
>
> url.parse('//domain/search', false, true);
Url {
  protocol: null,
  slashes: true,
  auth: null,
```

```
    host: 'domain',
    port: null,
    hostname: 'domain',
    hash: null,
    search: null,
    query: null,
    pathname: '/search',
    path: '/search',
    href: '//domain/search' }
  >

4)URL 분석하기
  -url 분석하기
    var urlStr = 'http://examples.burningbird.net:8124/?file=main';
    var parsed = url.parse(urlStr);
    console.log(parsed);
  -결과
    Url {
      protocol: 'http:',
      slashes: true,
      auth: null,
      host: 'examples.burningbird.net:8124',
      port: '8124',
      hostname: 'examples.burningbird.net',
      hash: null,
      search: '?file=main',
      query: 'file=main',
      pathname: '/',
      path: '/?file=main',
      href: 'http://examples.burningbird.net:8124/?file=main' }

5)URL과 쿼리 문자열
  -쿼리 문자열
    이름=값&이름-값 형태로 정보 전달
      http://examples.burningbird.net:8124/?file=main
  -URL 모듈로 쿼리 문자열 파싱
    url.parse('http:/...', true);

6)URL 분석하기
  -URL 분석하기
    var urlStr = 'http://examples.burningbird.net:8124/?file=main';
    var parsed = url.parse(urlStr, true);
    var query = parsed.query;
    console.log(query);
  -결과
    { file: 'main' }

7)Lab1 : url.js
  var url = require('url');

  var urlStr = 'http://examples.burningbird.net:8124/?file=main';
  var parsed = url.parse(urlStr);
  console.log(parsed);
```

```
  console.log('protocol : ', parsed.protocol);
  console.log('host : ', parsed.host);
  console.log('query : ', parsed.query);
  ---------------------------------------------------------------------
  Url {
     protocol: 'http:',
     slashes: true,
     auth: null,
     host: 'examples.burningbird.net:8124',
     port: '8124',
     hostname: 'examples.burningbird.net',
     hash: null,
     search: '?file=main',
     query: 'file=main',
     pathname: '/',
     path: '/?file=main',
     href: 'http://examples.burningbird.net:8124/?file=main' }
    protocol :  http:
    host :  examples.burningbird.net:8124
    query :  file=main

8)Lab2 : url1.js
  var url = require('url');

  var urlStr = 'http://examples.burningbird.net:8124/?file=main';
  var parsed = url.parse(urlStr, true);
  console.log('query : ', parsed.query);
  ---------------------------------------------------------------------
    var url = require('url');

    var urlStr = 'http://examples.burningbird.net:8124/?file=main';
    var parsed = url.parse(urlStr, true);
    console.log('query : ', parsed.query);

9)URL 만들기
  -URL 만들기
    url.format(urlObj);
  -URL 변환
    url.resolve(from, to)
  -URL 만들기 : format
    --protocol
    --host
    --pathname : 경로
    --search : 쿼리 스트링
    --auth : 인증정보

  $ node
  >var url = require('url');
  undefined
  > var obj = url.parse('http://domain/tags/search?q=node.js');
  undefined
  > url.format(obj);
  'http://domain/tags/search?q=node.js'
  >
```

10)Lab3 : url2.js
```
var urlObj = {
  protocol : 'http',
  host : 'idols.com',
  pathname : 'schedule/radio',
  search : 'time=9pm&day=monday'
}

var urlStr =  url.format(urlObj);
console.log(urlStr);

------------------------------------------------------
http://idols.com/schedule/radio?time=9pm&day=monday
```

11)URL 인코딩
-URL에 허용되는 문자
  --알파벳, 숫자, 하이픈, 언더스코어, 점, 틸드
-URL 인코딩하기
  https://www.google.com/search?q=아이폰
  https://www.google.com/search?q=%EC%95%..
-써드 파티 모듈
  urlencode

12)URL 모듈은 Query String 모듈과 함께 사용되는 경우가 많다.
-Query String 모듈은 수신된 쿼리 문자열을 파싱하거나 쿼리 문자열로 사용하기 위한 문자열을 준비하기 위한 기능들을 제공하는 간단한 유틸리티 모듈이다.
-Query 문자열에서 키/값 쌍을 추출해내려면 querystring.parse 메소드를 사용한다.
  var vals = querystring.parse('file=main&file=secondary&type=html');

```
$ node
> var qs = require('querystring');
undefined
> qs.parse('q=nodejs&year=2011');
{ q: 'nodejs', year: '2011' }
> qs.parse('q=nodejs&year=2011', ';');
{ q: 'nodejs&year=2011' }
> qs.parse('q=nodejs&year=2011', ';', ':');
{ 'q=nodejs&year=2011': '' }
>
```

13)Lab4 : querystring.js
```
var querystring = require('querystring');
var vals = querystring.parse('file=main&file=secondary&type=html');
console.log(vals);
------------------------------------------------------------------
{ file: [ 'main', 'secondary' ], type: 'html' }
```

14)query string 내에서 file 이 두번 나오므로, file의 두 값은 배열로 그룹화되고 각 값에 개별적으로 접근할 수 있다.
-console.log(vals.file[0]); //main 을 반환

15)Lab5 :
```
$ node
> var qs = require('querystring');
```

```
  undefined
  > qs.stringify({q:'node.js', year:2011});
  'q=node.js&year=2011'
  > qs.stringify({q:'node.js', year:2011}, ';');
  'q=node.js;year=2011'
  > qs.stringify({q:'node.js', year:2011}, ';', ':');
  'q:node.js;year:2011'
  >
  > qs.stringify({q:'nodejs', some:'한글'});
  'q=nodejs&some=%ED%95%9C%EA%B8%80'
  > qs.stringify({q:'nodejs', some:'%%'});
  'q=nodejs&some=%25%25'
  >

 16)Lab6 : querystring1.js
  var querystring = require('querystring');
  var vals = querystring.parse('file=main&file=secondary&type=html');
  var qryString = querystring.stringify(vals);
  console.log(qryString);
  ------------------------------------------------------------
  file=main&file=secondary&type=html


18. Net module
 1)비동기 네트워크를 다루는 모듈

 2)비동기 네트워크 서버와 클라이언트에 관련한 함수를 제공

 3)Lab1 : net.js
  var net = require('net');

  var server = net.createServer(function(socket){
     console.log('connected');
     console.log('From ' + socket.remoteAddress + ' ' + socket.remotePort);
     socket.on('close', function(){
        console.log('client closed connection.');
     });
     socket.write('Hello\r\n');
  });

  server.listen(8124, function(){
     console.log('listening on port 8124');
  });
  ------------------------------------------------------------
  --Windows 에서는 telnet client 설치할 것

  $ node net
  listening on port 8124
  connected
  From ::1 50774
  client closed connection.

  $ telnet localhost 8124
  Hello
```

```
1932
1933  4)Lab2
1934    <netserver.js>
1935    var net = require('net');
1936
1937    var server = net.createServer(function(socket){
1938          console.log('connected');
1939          socket.on('data', function(data){
1940                console.log(data + ' from ' + socket.remoteAddress + ' ' + socket.remotePort);
1941                socket.write('Repeating: ' + data);
1942          });
1943          socket.on('close', function(){
1944                console.log('client closed connection.');
1945          });
1946          socket.on('error', function(){});
1947    });
1948
1949    server.listen(8124, function(){
1950          console.log('listening on port 8124');
1951    });
1952    ------------------------------------------------------------
1953    $ node netserver
1954    listening on port 8124
1955    connected
1956    Who needs a browser to communicate? from ::ffff:127.0.0.1 50818
1957    hello
1958     from ::ffff:127.0.0.1 50818
1959    world
1960     from ::ffff:127.0.0.1 50818
1961    client closed connection.
1962
1963
1964    <netclient.js>
1965    var net = require('net');
1966
1967    var client = new net.Socket();
1968    client.setEncoding('utf8');
1969
1970    client.connect('8124', 'localhost', function(){
1971       console.log('connected to server');
1972       client.write('Who needs a browser to communicate?');
1973    });
1974
1975    process.stdin.resume();
1976
1977    process.stdin.on('data', function(data){
1978       client.write(data);
1979    });
1980
1981    client.on('data', function(data){
1982       console.log(data);
1983    });
1984
1985    client.on('close', function(){
```

```
    console.log('connection is closed');
});
-----------------------------------------------------
$ node netclient
connected to server
Repeating: Who needs a browser to communicate?
hello
Repeating: hello

world
Repeating: world

^C
```

5)생성된 TCP 서버는 net.Server 클래스의 객체이고, 새로운 연결을 받기 위해 net.Socket의 객체이기도 하다.
- TCP 서버는 server.listen(port, [host], [listeningListener])를 실행하면 특정 호스트와 포트로부터 연결을 받기 시작한다.
-host를 생략하면 IPv4에 맞는 모든 주소로부터 연결을 받고, port 에 0을 설정하면 임의의 포트를 선택한다.
-server.listen() 함수는 비동기 함수이므로 port로 전달한 포트에 서버가 바인딩되면 listening 이벤트가 발생한다.
-listeningListener 파라미터에 콜백 함수를 지정하면 listening 이벤트 리스너에 추가한다.
-server.address()는 서버에 호스트와 포트에 대한 정보가 담겨있으며 {"port":8124, "family": 2, "address" : "0.0.0.0"}과 같은 형식이다.

6)server 메소드
-server.pause(msecs)
--msecs 밀리초만큼 서버가 새로운 요청을 받지 않는다.
--DoS 공격처럼 서버에 부하가 심할 때 유용하다.
-server.close()
--더 이상 서버가 새로운 요청을 받지 않는다.
--비동기로 실행되므로, 완료되면 close 이벤트가 발생한다.

7)server 프라퍼티
-server.maxConnections : 서버가 최대로 받아들일 수 있는 연결 수 지정
-server.connections : 현재 서버의 동시 연결 수를 알 수 있다.

8)server 이벤트
-listening
--server.listen() 가 호출됐을 때 발생하는 이벤트
--콜백 함수는 function(){} 이다.
-connection
--새로운 연결이 생겼을 때 발생하는 이벤트
--콜백 함수는 function(socket){}이고, 파라미터인 socket은 연결된 소켓으로 net.Socket의 객체이다.
-close
--서버가 닫혔을 때 발생하고, 콜백함수는 function(){}이다.
-error
--서버에서 에러가 생겼을 때 발생하는 이벤트
--error 이벤트가 발생한 뒤 이어서 close 이벤트가 발생한다.

9)서버에서 connection 이벤트가 발생하면 콜백 함수로 소켓이 전달된다.
-실제 서버의 로직은 대부분 이 소켓을 이용해 작성하며, net.Socket의 객체이다.

10)net.Socket은 TCP나 유닉스 소켓의 추상 객체로, 이중 통신 방식의 스트림 인퍼페이스를 구현했다.

11)net.Socket의 메소드

-socket.setEncoding()
--소켓으로 받는 데이터의 인코딩을 지정
--ascii, utf8, base64 가능
-socket.write(data[, encoding][, callback])
--소켓에 데이터 보내기
--encoding은 data가 문자일 때 설정
--기본값은 utf-8
--문자열이 커널 버퍼로 flush 되면 true를 리턴하고 커널버퍼가 가득 찼으면 false 를 리턴한다
--커널 버퍼가 다시 비워졌을 때 drain 이벤트가 발생한다.
--callback에 지정된 콜백 함수는 데이터가 모두 쓰였을 때 호출
-socket.end([data][, encoding])
--소켓을 종료하는 메소드
--이 함수는 FIN 패킷을 보내 소켓을 닫기 때문에 서버 쪽에서는 여전히 데이터를 보낼 수 있다.
--end()에 data와 encoding 파라미터를 전달하면 socket.write() 실행 후 end()를 실행한 것과 같다.
-socket.pause()
--소켓에서 데이터를 읽는 것을 멈추기 때문에 더 이상 data이벤트가 발생하지 않는다.
--다시 데이터를 받으려면 socket.resume()을 실행한다.

12)net.Socket의 프라퍼티
-socket.remoteAddress : 접속한 클라이언트의 원격 IP를 돌려준다.
-socket.bufferSize
--소켓에 쓰기 위해 현재 버퍼에 있는 캐릭터의 크기를 알 수 있다.
--버퍼에 존재하는 문자열은 실제 데이터를 보낼 때 인코딩되기 때문에 이 프라퍼티가 알려주는 크기는 인코딩되기 전의 문자 크기이다.

13)net.Socket의 이벤트
-connect
--소켓 연결이 이뤄졌을 때 발생
--콜백함수는 function(){}.
-data
--소켓에서 데이터를 받았을 때 발생
--콜백함수는 function(data){}
--data는 Buffer나 문자열이 된다.
-end
--소켓으로 FIN 패킷을 받았을 때 발생
--콜백 함수는 function(){}
-drain
--쓰기 버퍼가 비워졌을 때 발생
--콜백 함수는 function(){}
-error
--에러가 발생했을 때 발생
--콜백함수는 function(exception){}
--error 이벤트가 발생한 뒤 이어서 close 이벤트가 발생한다.
-close
--소켓이 완전히 닫혔을 때 발생
--콜백함수는 function(had_error){}
--had_error는 소켓을 닫는 중 에러가 발생했는지를 나타내는 boolean 값

19. OS module
1)많이 사용하지는 않지만 서버의 기본적인 하드웨어 자원들의 정보를 확인할 때 주로 사용

2)모듈 로딩
var os = require('os');

```
  3)메소드
    -tmpdir() : 서버의 temp 디렉토리 반환
    -endianness() : 엔디언 타입 반환, BE or LE
    -hostname() : 서버의 호스트 이름
    -homedir() : 홈디렉토리 정보
    -type() : 서버의 OS 타입
    -platform() : 서버의 플랫폼
    -arch() : 서버의 CPU 아키텍처
    -release() : 운영체제 OS 버전
    -uptime() : 운영체제 시작된 시간
    -loadavg() : load average에 담긴 정보
    -totalmem() : 시스템 메모리
    -freemem() : 사용가능 메모리
    -cpus() : cpu 정보
    -networkInterfaces() : 네트워크 정보

20. DNS module
  1)비동기 DNS 요청 기능을 가진 C 라이브러리인 c-ares 를 사용하여 DNS 해석을 제공

  2)DNS 모듈은 다른 모듈들에세 사용되고, 도메인이나 IP 주소를 찾아내는 것이 필요한 어플리케이션들에게 유용

  3)메소드
    -dns.lookup(hostname[, options], callback)
      --지정된 도메인의 IP 주소를 찾아주는 메소드
      var dns = require('dns');
      var options = {
        family: 4,
        hints: dns.ADDRCONFIG | dns.V4MAPPED,
      };
      dns.lookup('google.com', options, function(err, address, family){
        console.log('address: %j family: IPv%s', address, family);
      });

      // When options.all is true, the result will be an Array.
      options.all = true;
      dns.lookup('google.com', options, function(err, addresses){
        console.log('addresses: %j', addresses);
      });
      ---------------------------------
      addresses: [{"address":"172.217.26.46","family":4}]
      address: "172.217.26.46" family: IPv4


    -dns.reverse(ip, callback)
      --지정된 IP 주소에 대해 도메인명의 배열을 반환
      dns.reverse('172.217.26.46', function(err, domains){
        domains.forEach(function(domain){
          console.log(domain);
        });
      });
      ------------------------------------------------
        nrt12s17-in-f46.1e100.net
        nrt12s17-in-f14.1e100.net
```

```
  -dns.resolve(hostname[, rrtype], callback)
    --A, MX, NS 등과 같이 지정된 유형에 따라 레코드 유형의 배열을 반환
    --rrtype
      ---'A' - IPV4 addresses, default
      ---'AAAA' - IPV6 addresses
      ---'MX' - mail exchange records
      ---'TXT' - text records
      ---'SRV' - SRV records
      ---'PTR' - PTR records
      ---'NS' - name server records
      ---'CNAME' - canonical name records
      ---'SOA' - start of authority record
      ---'NAPTR' - name authority pointer record

    dns.resolve('google.com', 'MX', function(err, addresses){
       addresses.forEach(function(address){
          console.log(address);
       });
    });
    ---------------------------------------------------------
    { exchange: 'alt2.aspmx.l.google.com', priority: 30 }
    { exchange: 'aspmx.l.google.com', priority: 10 }
    { exchange: 'alt4.aspmx.l.google.com', priority: 50 }
    { exchange: 'alt1.aspmx.l.google.com', priority: 20 }
    { exchange: 'alt3.aspmx.l.google.com', priority: 40 }


21. Readline module
  1)줄 단위로 스트림을 읽을 수 있게 해 주는 모듈

  2)하지만, 이 모듈을  포함시키고 나면 인터페이스와 stdin 스트림을 닫기 전까지 Node 프로그램이 종료되지 않는다는 것에 주의해야

  3) 1번 입력 받는 예제
    -createInterface를 통해 input과 output을 생성
    -question()에 callback 함수를 생성
      var readline = require('readline');

      var r = readline.createInterface({
        input:process.stdin,
        output:process.stdout
      });

      r.question("What is your name?", function(answer) {
        console.log("Hi! ", answer);
        r.close() // 반드시 close()를 해줘야
      });

  4)반복적으로 입력받는 예제
    -interface 생성
    -생성받은 interface에 prompt 세팅
    -line은 한줄을 입력받는 내용
      var readline = require('readline');
      var r = readline.createInterface({
```

```
    input:process.stdin,
    output:process.stdout
  });
  r.setPrompt('> ');
  r.prompt();
  r.on('line', function(line){
    if (line == 'exit') {
      r.close();
    }
    console.log(line);
    r.prompt()
  });
  r.on('close', function() {
    process.exit();
  });
```

22. Cluster module

1)개요

-여러 시스템을 하나로 묶어서 사용하는 기술

-Node.js는 Single Thread에서 동작하기 때문에 멀티프로세스의 이점을 얻지 못한다.

-IO에 대한 처리는 이벤트 루프를 통해 좋은 성능을 보여주지만, CPU 계산량이 많은 부분에서는 취약한 부분이 있다.

-이 부분을 해결하기 위한 모듈이다.

-node.js는 기본적으로 하나의 프로세스가 32bit에서는 512MB의 메모리, 64Bit에서는 1.5GB 메모리를 사용하도록 제한되어 있다.

-V8엔진의 제한을 그대로 반영한 것인데, 물론 설정으로 더 늘릴 수는 있지만 그렇게 하기 보다는 worker 를 늘리는 것을 권장하고 있다.

-여러개의 워커들이 병렬로 동작하며 효율을 극대화하는 것을 바람직한 방향으로 권하고 있는 것이다.

-node.js 에서 worker 를 생성하는 방법은, child_process와 cluster 정도로 요약할 수 있다.

-cluster 는 node.js v0.8 부터 소개되었는데, 큰 부하를 노드 프로세스들의 클러스터를 통해 다루려는 목적으로 시작되었다.

-추가적으로 이 프로세스들은 서버의 포트들을 공유할 수 있기 때문에 web application 에 매우 적합하다.

-cluster의 경우 node.js v0.12 이전 버전에서 worker 들에게 균일하게 load balancing 이 안되는 문제가 있다는 점을 주의해야 한다.

-다행히 0.12버전부터는 Round-Robin Load Balancing 이 적용되어 해당 이슈가 해결된 상태이다.

-프로세스들을 단순하게 병렬로 실행하는 것은 child_process.fork() 로 가능하고, 여기에 로드밸런싱과 포트 공유 등이 필요하다면 클러스터로 접근하는 것이 좋다.

-두 방식 모두 IPC(Inter-Process Communication)로 process 간에 통신이 가능하기 때문에 로드 밸런싱 등의 추가적인 기능이 필요한 경우 클러스터를 활용하고 워커를 직접적으로 컨트롤해야하는 경우 child_process 를 주로 활용하게 된다.

-참고로 새로운 child process 는 모두 V8의 인스턴스이기때문에 30ms의 시작시간과 10MB 가량의 메모리를 소모한다는 것을 기억해야 한다.

2)개별 시스템에서 Cluster

-Multi Processor

-Multi Core

3)Cluster

-Node.js 어플리케이션 : 1개의 Single Thread

-Multi Core 시스템의 장점을 살리기 - Cluster

-Node.js Cluster

--Cluster 사용시 포트 공유 - 서버 작성 편리

--Core(Processor)의 갯수 만큼 사용

-clustering : Master와 Worker process

```
  -Master
    --Main Process
    --worker 생성
  -Worker
    --보조 프로세스
    --마스터가 생성

4)cluster module
  var cluster = require('cluster');
  -cluster 모듈을 가져왔다면, 실제 클러스터를 생성하기 전에 스케쥴링 방식을 설정해줄 수 있는데, 아래와 같이 지정해 줄 수 있다.
    //워커 스케쥴을 OS에 맡긴다.
    cluster.schedulingPolicy = cluster.SCHED_NONE;
    //워커 스케쥴을 Round Robin 방식으로 한다.
    cluster.schedulingPolicy = cluster.SCHED_RR;
  -원래는 기본적으로 스케쥴을 OS에 맡기는 방식이었는데, 이 경우 특정 워커에 작업이 몰리는 경우가 많아서 차라리 순차적으로 하나씩 작업을 배분하는 Round Robin 방식이 node.js v0.12에서 추가되었다.
  -동일한 JavaScript 파일을 실행하면서 처음 실행되면 기본적으로 마스터가 된다.
  -마스터에서는 cluster.fork() 메서드를 통해서 워커들을 생성하면 생성된 워커들도 마찬가지로 동일한 JavaScript 파일을 실행하게 되는데, 이때 이미 마스터가 있다면 새롭게 실행되는 프로세스는 워커가 된다.
  -마스터와 워커가 수행해야 할 각 작업은 isMaster, isWorker 메서드를 활용해서 마스터일 때와 워커일 때를 구분해서 규정해주면 된다.
  -마스터인 경우 되도록 워커들을 생성/관리하는 로직만 포함하고 그 외의 로직은 적게 가져가는 것이 좋다.
  -cluster 생성
    cluster.fork()  //워커 생성은 fork를 수행한 만큼 생성된다.
  -구분하기
    cluster.isMaster
      --내부적으로 process.env.NODE_WORKER_ID의 값이 undefined 이면 마스터 프로세스로 판단한다.
    cluster.isWorker

5)cluster 생성과 동작
  -클러스터링을 사용하는 대략적인 구조
  -마스터-워커 생성
    if(cluster.isMaster){
      //마스터 코드
      cluster.fork();
    }else{
      //워커 코드
    }

6)Lab : cluster.js
    var cluster = require('cluster');
    var http = require('http');
    var numCPUs = require('os').cpus().length;

    if (cluster.isMaster) {
        // 클러스터 워커 프로세스 포크
        for (var i = 0; i < numCPUs; i++) {
            cluster.fork();
        }

        cluster.on('exit', function(worker, code, signal) {
            console.log('worker ' + worker.process.pid + ' died');
        });
```

```
    } else {
        http.createServer(function(req, res) {
            var str = "";
            for (var i = 0; i < 10; i++) {
                str += i;
            }
            res.writeHead(200);
            res.end("hello world " + process.pid + " : " + str);
        }).listen(8000);
        console.log(process.pid);
    }
    ------------------------------------------------------------
    In Browser, http://localhost:8000
    브라우저에 hello world 출력됨.
    7500
    1448
    6372
    4248
    6400
    7832
    8344
    6940
    현재 본인은 코어가 8개임, 운영체제 프로세스 목록에 가면 위의 프로세스번호(pid)에 해당하는 node.exe 프로세스가
    보임

7)cluster event
  -cluster event
    --fork : 워커 생성 이벤트
    --online : 워커 생성 후 동작하는 이벤트
    --listening : 워커에 작성한 서버의 listen 이벤트
    --disconnect : 워커 연결 종료
    --exit : 워커 프로세스 종료
  -워커의 이벤트
    --fomessage : 메시지 이벤트
    --disconnect : 워커 연결 종료

8)Worker
  -worker 접근
    cluster.worker
  -worker 식별자
    worker.id

9)worker 종료
  worker.kill([signal='SIGTERM'])

10)cluster를 사용하는 대략적인 구조
  if(cluster.isMaster0{
    cluster.fork();
    culster.on('online', function(worker){
      //워커 생성 후 실행
      console.log('Worker #' + worker.id + ' is Online.');
    });
    cluster.on('exit', function(worker.code, signal){
      //워커 종료 이벤트
```

```
2347        console.log('Worker #' + worker.id + ' exit');
2348        console.log('Worker #' + worker.id + '\'s exit code : ' + code);
2349        console.log('Worker #' + worker.id + '\'s signla ' + signal);
2350      });
2351    }else{
2352      var worker = cluster.worker;
2353      //워커 종료
2354      worker.kill();
2355    }
2356
2357  11)서버에 클러스터 적용
2358    if(cluster.isMaster){
2359      cluster.fork();
2360    }else{
2361      http.createServer(function(req, res){
2362        //서버 코드
2363      }).listen(8000);
2364    }
2365    -clustering 기능 지원 프로세스 모듈
2366      --pm2
2367
2368  12)데이터 전달
2369    -마스터와 워커간 통신을 사용하면, 워커의 재시작이 필요한 경우 워커들에게 종료할 예정이라고 메시지를 보내고 워커가 종
       료 준비를 마쳤을 때 마스터에게 다시 메시지를 보내서 안전하게 워커를 죽인 뒤 재생성을 수행하는 등의 작업을 해줄 수 있
       다.
2370    -마스터가 워커에게 데이터 전달
2371      worker.send(data);
2372    -워커의 데이터 이벤트
2373      worker.on('data', function(data){
2374      });
2375    -워커가 미스터에게 데이터 전달
2376    -
2377      process.send(data);
2378    -마스터에서의 데이터 이벤트
2379      var worker = cluster.fork();
2380      worker.on('message', function(data){
2381      });
2382
2383  13)데이터 전달 예
2384    if(cluster.isMaster)[
2385      var worker = cluster.fork();
2386      worker.on('message', function(message){
2387        console.log('Master received from ' + worker.process.pid + ', message = ', message);
2388      });
2389      cluster.on('online', function(worker){
2390        worker.send({message : 'Hello Worker'});
2391      });
2392    }else{
2393      var worker = cluster.worker;
2394
2395      worker.on('message', function(message){
2396        console.log('Worker received from Master, message = ', message);
2397      });
2398      process.send({message : 'Fine Thank You!'});
```

```
2399    }
2400
2401   14)마스터와 워커 분리
2402    -별도의 파일로 분리하기
2403      cluster.setupMaster([settings])
2404        --exec : 워커 파일
2405        --args : 실행 파라미터
2406    -마스터 - fork
2407        cluster.setupMaster({
2408          exec : 'worker.js'
2409        });
2410        cluster.fork();
2411
2412
2413  23. Child Process
2414   1)자식 프로세스 생성
2415    -보통 4가지 방법 사용
2416      var child_process = require('child_process');
2417   2)첫번째 생성 방법: spawn() 사용
2418    -child_process.spawn(command[, args][, options])
2419      --options 파라미터의 기본값은 {cwd:undefined, env:process.env, setsid:false}
2420      --cwd : 생성된 프로세스가 실행되는 디렉토리를 지정
2421      --env : 새 프로세스가 접근할 수 있는 환경 변수 지정
2422      --setsid가 true이면 서브프로세스를 새로운 세션으로 생성한다.
2423    -생성된 자식 프로세스는 child.stdin, child.stout, child.stderr 세가지 스트림 사용
2424   3)Lab : childprocess.js
2425    var spawn = require('child_process').spawn, pwd = spawn('pwd'); //not Windows, but Linux
2426    pwd.stdout.on('data', function(data){
2427      console.log('stdout : ' + data);
2428    });
2429    pwd.stderr.on('data', function(data){
2430      console.log('stderr : ' + data);
2431    });
2432    pwd.on('exit', function(code){
2433      console.log('child process exited with code ' + code);
2434    });
2435
2436   4)만일 오류가 발생하지 않으면 명령줄에 출력된 내용은 자식 프로세스의 stdout으로 전송, 프로세스의 data이벤트가 발생
2437   5)만일 오류가 발생하면 stderr로 전송되고 콘솔에 오류를 표시한다.
2438    var spawn = require('child_process').spawn, pwd = spawn('pwd', ['-g']);
2439
2440    stderr : pwd : invalid option -- 'g'
2441    Try 'pwd --help' for more information.
2442    child process exited with code 1
2443
2444   6)Lab : childprocess1.js
2445    var spawn = require('child_process').spawn;
2446    var ls = spawn('ls', ['-l', './']);  //not Windows, but Linux
2447    ls.stdout.on('data', function(data){
2448      console.log('stdout : ' + data);
2449    });
2450    ls.stderr.on('data', function(data){
2451      console.log('stderr : ' + data);
2452    });
```

```
  ls.on('error', function(err){
    console.log('Failed to start child process.');
  });
  ls.on('exit', function(code){
    console.log('child process exited with code ' + code);
  });
  ----------------------------------------------------------------
  stdout : total 8
  drwxrwxr-x 3 instructor  instructor  4096  4월 24 16:54  0424
  -rw-rw-r-- 1 instructor  instructor   414  4월 26 08:15  spawn.js

7)위의 코드에서 자식 프로세스의 stdout 및 stderr로 출력하는 것을 보았는데, 그렇다면 stdin으로 데이터를 보내는 것은 어떻게 하나?
  i.g) fild . -ls | grep test

8)Lab : childprocess2.js
  var spawn = require('child_process').spawn;
  var ls = spawn('find', ['.', '-ls']);
  var grep = spawn('grep', ['test']);

  grep.stdout.setEncoding('utf8');

  //찾은 결과를 grep으로 전달
  find.stdout.on('data', function(data){
    grep.stdin.write(data);
  });

  //grep을 실행해서 결과를 출력
  grep.stdout.on('data', function(data){
    console.log(data);
  });

  //양쪽에 대한 오류 처리
  find.stderr.on('data', function(data){
    console.log('find stderr : ' + data);
  });
  grep.stderr.on('data', function(data){
    console.log('grep stderr : ' + data);
  });

  //양쪽에 대한 종료 처리
  find.on('exit', function(code){
    ifOcode !== 0)
      console.log('find process exited with code ' + code);
    //grep 프로세스도 종료시킴
    grep.stdin.end();
  });
  grep.on('exit', function(code){
    if(code !== 0)
      console.log('grep process exited with code ' + code);
  });

9)Node 0.6에서는 자식 프로세스가 종료되고 모든 STDIO 파이프가 닫힐 때까지 exit 이벤트가 발생되지 않느다.
  -Node 0.8부터 자식 프로세스가 종료되자마다 해당 이벤트가 발생한다.
```

-이럴경우 어클리케이션이 crash되는데, grep 자식 프로세스가 데이터를 처리하려고 하는 시점에 자식 프로세스의 STDIO 파이프가 닫혀버리기 때문이다.
-해결방법은 find 프로세스의 exit 이벤트 대신 close 이벤트를 수신 대기해야 한다.
-Node 0.8부터는 close 이벤트는 자식 프로세스가 종료되고 모든 STDIO 파이프가 닫히면 발생한다.

```
//양쪽에 대한 종료 처리
find.on('close', function(code){
  ifOcode !== 0)
    console.log('find process exited with code ' + code);
  //grep 프로세스도 종료시킴
  grep.stdin.end();
});
```

10)child_process.exec(command[, options][, callback])
-command 를 실행하고 그 결과를 반환하는 메소드
-콜백함수는 function(error, stdout, stderr){}
-child_process.execFile 메소드와의 유일한 차이점은 execFile은 명령을 실행하는 대신, 파일에서 어플리케이션을 실행하는 것이다.
-options
--cwd <String> Current working directory of the child process
--env <Object> Environment key-value pairs
--encoding <String> (Default: 'utf8')
--shell <String> Shell to execute the command with (Default: '/bin/sh' on UNIX, 'cmd.exe' on Windows, The shell should understand the -c switch on UNIX or /s /c on Windows. On Windows, command line parsing should be compatible with cmd.exe.)
--timeout <Number> (Default: 0)
--maxBuffer <Number> largest amount of data (in bytes) allowed on stdout or stderr - if exceeded child process is killed (Default: 200*1024)
--killSignal <String> | <Integer> (Default: 'SIGTERM')
--uid <Number> Sets the user identity of the process. (See setuid(2).)
--gid <Number> Sets the group identity of the process. (See setgid(2).)
-만일 timeout 이 0보다 크면 timeout 밀리초 후에 프로세스를 종료하는데, 종료할 때 killSignal을 이용

11)Lab : exec.js

```
var exec = require('child_process').exec;

exec('cat *.js bad_file | wc -l', function(error, stdout, stderr){
  console.log('stdout : ' + stdout);
  console.log('stderr : ' + stderr);
  if(error !== null) console.log('exec error : ' + error);
});
```

```
stdout : 20

stderr : cat: bad_file: No such file or directory
```

12)다음은 execFile 의 예제이다
-실행 파일의 내용이 만일 다음과 같다면,

```
<app.js>
#! /usr/bin/nodejs
console.log(global);
```

-다음 어플리케이션은 버퍼에 있는 결과를 출력한다.

```
<execFile.js>
var execFile = require('child_process').execFile, child;
```

```
2555
2556      child = execFile('./app.js', function(stdin, stdout, error){
2557        if(error){
2558          console.log("Error : ", error);
2559        }
2560        console.log('stdout : ', stdout);
2561      });
2562
2563    13)Windows OS에서 자식 프로세스 어클리케이션 실행하기
2564      -cmd.exe의 첫번째 인수로 전달하는 /c 플래그는 명령을 수행한 후 종료하라는 것
2565      -예제
2566        var spawn = require('child_process').spawn;
2567        var cmd = spawn('cmd', ['/c', 'dir\n']);  //not Linux, buf Windows
2568        cmd.stdout.on('data', function(data){
2569          console.log('stdout : ' + data);
2570        });
2571        cmd.stderr.on('data', function(data){
2572          console.log('stderr : ' + data);
2573        });
2574        cmd.on('error', function(err){
2575          console.log('Failed to start child process.');
2576        });
2577        cmd.on('exit', function(code){
2578          console.log('child process exited with code ' + code);
2579        });
2580
2581
2582  24. 기본 모듈을 이용한 간단한 채팅 프로그램
2583    <tcp-chat.js>
2584      var net = require('net')
2585      var sockets = [];
2586
2587      var server = net.createServer(function(socket){
2588        sockets.push(socket);
2589        socket.on('data', function(data){
2590          for(var i = 0 ; i < sockets.length ; i++){
2591            if(sockets[i] !== socket){
2592              sockets[i].write('[' + socket.remoteAddress + '] : ' + data, 'utf-8');
2593            }
2594          }
2595        });
2596        socket.on('end', function(){
2597          var i = sockets.indexOf(socket);
2598          sockets.splice(i, 1);
2599        });
2600      });
2601
2602      server.listen(8000, function(){
2603        console.log('TCP Chatting Starting...');
2604      });
```