Server:
File: net_server.js

```
 const net = require('net');
 Var server = net.createServer((socket)=> {
   Socket.end('goodbye\n');
}).on('error', (err) => {
   // handle errors here
   throw err;
});
// grab a random port.
  server.listen(() => {
  address = Server.address();
  Console.log('opened server on %j', address);
});
```

Open Node.js command prompt and run the following code:

```
     node net_server.js
```

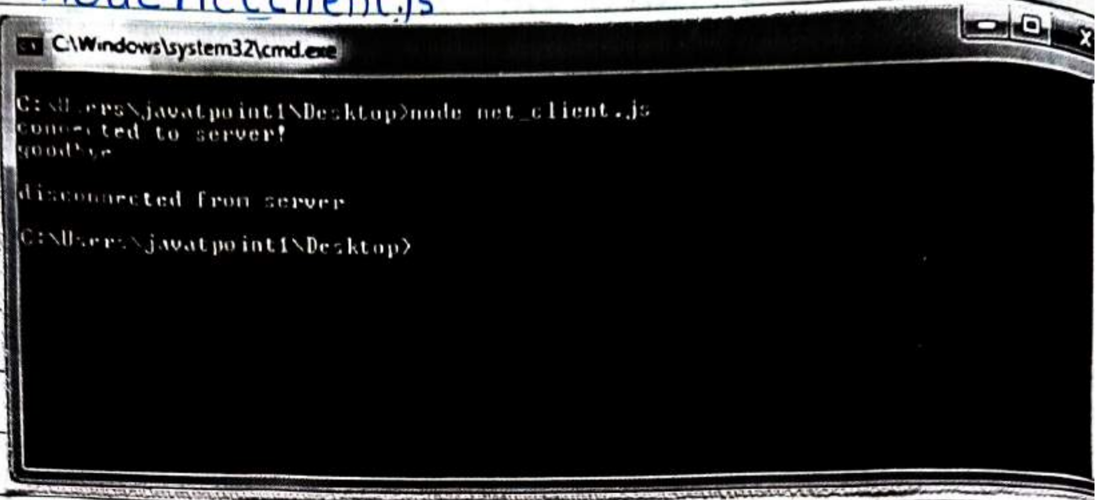
Node.js net example 1

Client:
File: net_client.js

```
Const net = require('net');
Const client = net.connect({port:50302}, () => { //use same
                                                  Port of server
   cosnole.log('connected to server!');
   Client.write('World!\r\n');
 });
  Client.on('data', (data) => {
   console.log(data.toString());
   Client.end();
});
  Client.on('end', () => {
   Console.log('disconnected from server');
});
```

Open Node.js command prompt and run the following code:

node netclient.js


C:\Windows\system32\cmd.exe

```
C:\Users\javatpoint1\Desktop>node net_client.js
connected to server!
goodbye

disconnected from server

C:\Users\javatpoint1\Desktop>
```

Node.js Crypto

The Node.js Crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's hash HMAC, cipher, decipher, sign and verify functions.

What it Hash

A hash is a fixed-length string of bits i.e. procedurally and deterministically generated from some arbitrary block of source data.

What is HMAC

HMAC stands for Hash-based Message Authentication Code. It is a process for applying a hash algorithm to both data and a secretkey that results in a single final hash.

Encryption Example using Hash and HMAC

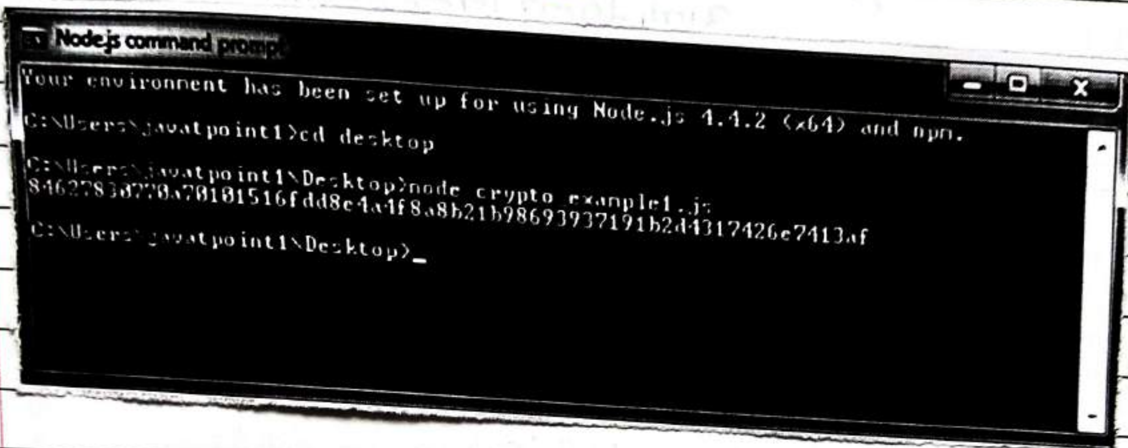
File: crypto_example1.js

```
const crypto = require('crypto');
const secret = 'abcdefg';
```

```
Const hash = Crypto.createHmac('sha256', secret)
                .update('Welcome to JavaTpoint)
                .digest('hex');
console.log(hash);
```

Open Nodejs Command prompt and run the following c

```
node crypto_example1.js
```



```
Nodejs command prompt
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.

C:\Users\javatpoint1>cd desktop

C:\Users\javatpoint1\Desktop>node crypto_example1.js
846278387780a78101516fdd8e4a4f8a8b21b98693937191b2d4317426e7413af

C:\Users\javatpoint1\Desktop>_
```

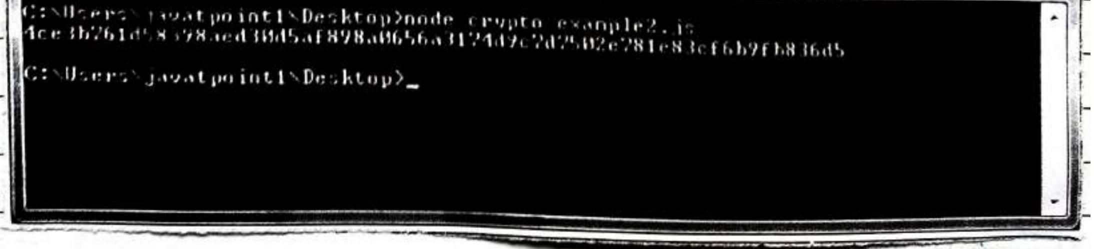
Encryption example using cipher

File: crypto_example2.js

```
Const crypto = require('crypto')
const cipher = crypto.CreateCipher('aes192', 'a password');
Var encrypted= Cipher.update ('Hello JavaTpoint', 'utf8', 'hex'
encrypted + = cipher. final ('hex');
console.log(encrypted);
```

Open Node.js Command prompt and run the following Coc

```
node crypto_example2.js
```



```
Nodejs command prompt
C:\Users\javatpoint1\Desktop>node crypto_example2.js
4ce3b761d58a98aed30d5af898a0656a3124d9c7d2502e281e83cf6b9fb836d5

C:\Users\javatpoint1\Desktop>_
```

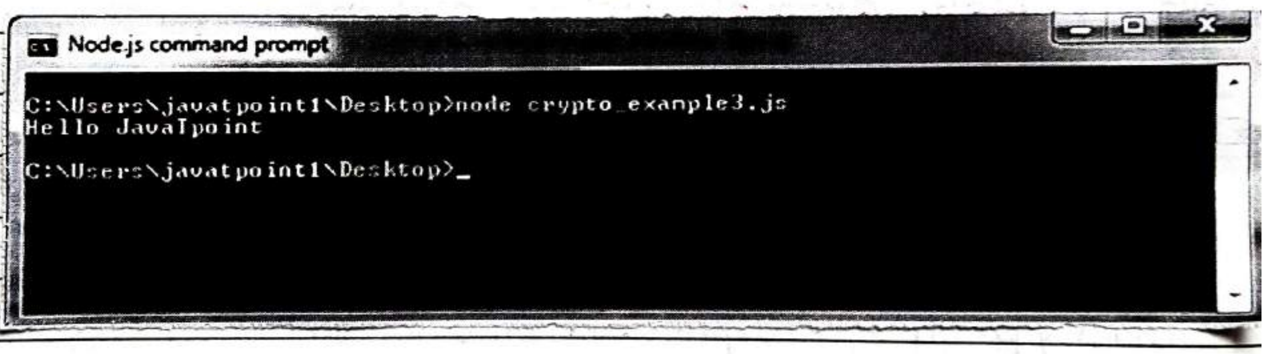
Decryption example using Decipher

File: crypto_example3.js

```
Const crypto = require('crypto');
Const decipher= crypto.createDecipher('aes 192',
                        'a password');
Var encrypted = '4ce3b761d58398aed30d5af898065
                  3174d9c7d7502e781e83c
Var decrypted = decipher.update(encrypted, 'hex',
                              'utf8');
decrypted + = decipher.final('utf8');
console.log(decrypted);
```

Open Node.js command prompt and run the following code:

```
node crypto_example3.js
```



```
C:\Users\javatpoint1\Desktop>node crypto_example3.js
Hello JavaTpoint

C:\Users\javatpoint1\Desktop>_
```

Node.js TLS/SSL

What is TLS/SSL

TLS Stands for Transport Layer Security. It is the Successor to Secure Sockets, Layer(SSL). TLS along With SSL is used for cryptographic protocols to secure communication over the web.

TLS uses public-key cryptography to encrypt messages. It encrypts communication generally on the TCP layer.

What is public-key cryptography

In public-key cryptography, each client and each server has two keys: public key and private key. Public key is Shared with everyone and private key is Secured. To encrypt a message, a computer requires its private key and the recipient?s public key. On the other hand, to decrypt the message, the recipient requires its own You have to use require('tls') to access this module.

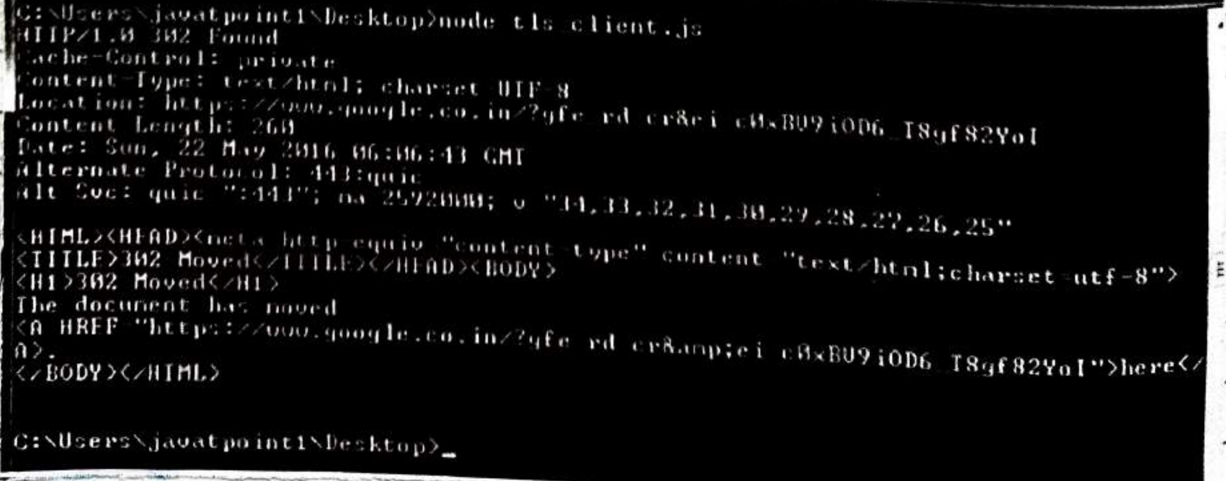
```
Var tls = require('tls');
```

Nodejs TLS Client example

File: tls_client.js

```
 tls = require ('tls');
 function Connected(stream){
     if (stream){
       //Socket Connected
       Stream.Write("GET/HTTP/1.0\n\rHost: encrypted.
                     google.com: 443\n\r\n\r");
    } else{
       console.log("Connection failed");
    }
}
 //needed to keep socket variable in scope
 Var dummy = this;
 //try to connect to the server
 dummy.socket = tls.connect(443, 'encrypted.google.com
                           function(){
   // Callback called only after successful socket connection.
   dummy.connected = true;
   if(dummy.socket.authorized){
    //authorization successful
```

```
        dummy.socket.setEncoding('utf-8');
        connected (dummy.socket);
    } else {
        //authorization failed
        console.log(dummy.socket.authorizationError);
        connected (null)
      }
  });
  dummy.socket.addListener('data', function(data) {
     //received data
     console.log(data);
});
  dummy.socket.addListener('error', function(error) {
    if(!dummy.connected) {
      //socket was not connected, notify callback
      connected (null);
    }
    console.log("FAIL");
    console.log(error);
});
  dummy.socket.addListener('close', function() {
   // do something
});
```



```
C:\Users\javatpoint1\Desktop>node tls_client.js
HTTP/1.0 302 Found
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Location: https://www.google.co.in/?gfe_rd=cr&ei=c0xBU9iOD6_I8gf82YoI
Content-Length: 260
Date: Sun, 22 May 2016 06:06:43 GMT
Alternate-Protocol: 443:quic
Alt-Svc: quic=":443"; ma=2592000; v="34,33,32,31,30,29,28,27,26,25"

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="https://www.google.co.in/?gfe_rd=cr&amp;ei=c0xBU9iOD6_I8gf82YoI">here</
A>.
</BODY></HTML>


C:\Users\javatpoint1\Desktop>_
```

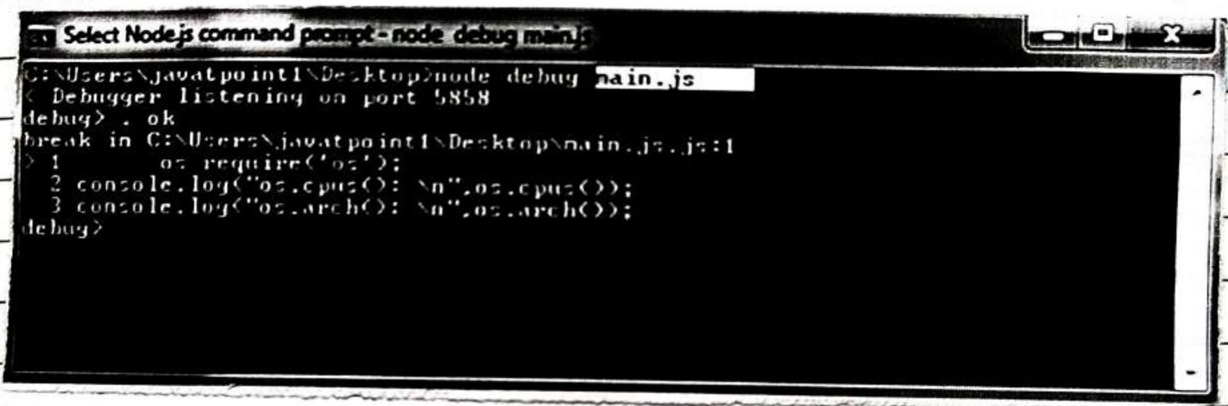
Node.js Debugger

Node.js provides a simple TCP based protocol and built-in debugging client. For debgging your JavaScript file, you can see use debug argument followed by the js file name you want to debug.

```
node debug [script.js | -e "script" | <host>:<port>]
```

Example

```
node debug main.js
```



Select Node.js command prompt - node debug main.js

```
C:\Users\javatpoint1\Desktop>node debug main.js
< Debugger listening on port 5858
debug> . ok
break in C:\Users\javatpoint1\Desktop\main.js.js:1
> 1       os require('os');
  2 console.log("os.cpus(): \n",os.cpus());
  3 console.log("os.arch(): \n",os.arch());
debug>
```

Node.js Process

Node.js provides the facility to get process information such as process id, architecture, platform, version, release uptime, upu usage etc. It can also be used to kill process set uid, set groups, unmask etc.
The process a global object, an instance of Event-Emitter, can be accessed from anywhere.

Node.js process properties

A list of commonly used Node.js process properties are given below.

Property	Description
arch	returns process architecture: 'arm', 'ia32', or 'x64'
args	returns commands line arguments as an array
env	returns user environment.
pid	returns process id of the process.
platform	returns platform of the process: 'drawin', 'freebsd', 'linux', 'sunos' or 'win32'
release	returns the metadata for the current node release
Version	returns the node version
versions	returns the node version and its dependencies

Node.js Process Properties Example

File: process_example1.js

```
Console.log(`Process Architecture: ${process.arch}`);
console.log(`Process PID: ${process.pid}`);
Console.log(`Process Platform: ${process.platform}`);
Console.log(`Process Version: ${process.version}`);
```

Node.js Process Functions

A list of commonly used Node.js process functions are given below.

Function	Description
cwd()	returns path of current working directory
hrtime()	returns the current high-resolution real time in a [seconds, nanoseconds] array
memory-usage()	returns an object having information of memory usage.
process.kill(pid [,signal])	is used to kill the given pid.
uptime	returns the Node.js process uptime in seconds.

Node.js Process Functions Example
File: process_example3.js

```
Console.log(`Current directory:${process.cwd()}`);
Console.log(`Uptime: ${process.uptime()}`);
```

Node.js Child Process

The Node.js child process module provides the ability to spawn child processes in a similar manner to popen (3).

There are three major way to create child process:

Node.js child process.exec() method

The child_process.exec() method runs a command in a console and buffers the output.

```
Child_process.exec(command[, options], callback)
```

Parameters:

1) Command: It specifies the command to run, with space-separated arguments.
2) Options: It may contain one or more of the following options:
- cwd: It specifies the current working directory of the child process.
- env: It specifies environment key-value pairs.
- encoding: String (Default: 'utf8')
- shell: It speifies 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 (Default: 200* 1024)
- killsSignal: String (Default 'SIGTERM')
- uid Number: sets the user identity of the process.
- gid Number: sets the group identity of the process.

Callback: The callback function specifies three arguments error, stdout and stderr which is called with the following output when process terminates.

Node.js child process.exec() example
File: child_process_example1.js