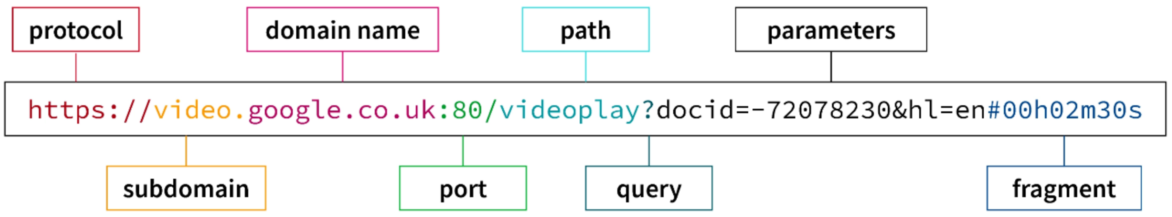
URL module

The URL module contains functions that help in parsing a URL. In other words, we can split the different parts of a URL easily with the help of utilities provided by the URL module.


protocol
domain name
path
parameters
https://video.google.co.uk:80/videoplay?docid=-72078230&hl=en#00h02m30s
subdomain
port
query
fragment

The syntax for including the url module in your application:
var url=require("url");

The url.parse() method takes the url string as a parameter and parses it. The url module returns an object with each part of the url as property of the object.

Syntax of url.parse() :

url.parse(url_string, parse_query_string, slashes_host)

Description of the parameters :

- **url_string :** <string> It is the URL string.
- **parse_query_string :** <boolean> It is a boolean value. By default, its value is false. If it is set to true, then the query string is also parsed into an object. Otherwise, the query string is returned as an unparsed string.
- **slashes_host :** <boolean> It is a boolean value. By default, its value is false. If it is set to true, then the token in between // and first / is considered host.

```
var u=require("url");
var addr="http://localhost:8080/default.html?year=2025&month=feb";
var q=u.parse(addr,true);
console.log(q);
// console.log(q.host);
// console.log(q.pathname);
// console.log(q.search);
var qdata=q.query;
console.log(qdata.year);
if(qdata.year%4==0)
{
  console.log("Its a leap year")
}
else{
  console.log("Its not a leap year")
}
==========================================================================
```

Output:

```
Url {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'localhost:8080',
  port: '8080',
  hostname: 'localhost',
  hash: null,
  search: '?year=2025&month=feb',
  query: [Object: null prototype] { year: '2025', month: 'feb' },
  pathname: '/default.html',
  path: '/default.html?year=2025&month=feb',
  href: 'http://localhost:8080/default.html?year=2025&month=feb'
}
2025
Its not a leap year
```

Example : Write a nodejs script to print query string of url in file using ES6 callback.

Fetch query as an object. Need to convert query to string using stringify and then we can write i t in file.

```
var u=require("url");
var ps=require("fs");
var adr1=" http://localhost:8080/default.html?year=2025&month=feb";

var q1=u.parse(adr1,true);
var qdata=JSON.stringify(q1.query);
ps.writeFile("fsd2.txt",qdata,(err)=>
{
   console.log("completed");
});
```

Fetch query as a string. We can directly write query string in a file.

```
var u=require("url");
var ps=require("fs");
var adr1=" http://localhost:8080/default.html?year=2025&month=feb";

var q1=u.parse(adr1,false);
var qdata=q1.query;
ps.writeFile("fsd2.txt",qdata,(err)=>
{
   console.log("completed");
});
```

Own Module

Modules are the collection of JavaScript codes in a separate logical file that can be used in ext ernal applications on the basis of their related functionality. Modules are popular as they are easy to use and reusable. To create a module in Node.js, you will need the exports keyword. T his keyword tells Node.js that the function can be used outside the module.

Syntax:

```
exports.function_name = function(arg1, arg2, ....argN) {
// function body
};
```

Method 1

In test.js file:

```
const add=(a,b)=>
{
   return(a+b);
}
module.exports=add;
```

In another file where you want to use that module:

```
var d=require("./test.js");
console.log(d(10,15));
```

Method 2

In test.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
{
   return(a*b);
}
module.exports.s=sub;
module.exports.m=mul;
```

In another file:

```
var d1=require("./test.js");
console.log(d1.s(10,5));
console.log(d1.m(10,15));
```

Method 3

In test.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
{
   return(a*b);
}
module.exports.d2=sub;
module.exports.e2=mul;
```

In another file:

```
var {d2,e2}=require("./test.js");
console.log(d2(10,7));
console.log(e2(10,12));
```

Method 4

In test.js file:

```
const sub=(a,b)=>
{
   return(a-b);
}
const mul=(a,b)=>
```

```
{
   return(a*b);
}
const name="Hello"
module.exports={sub,mul,name};
```

In another file:

```
var {sub,mul,name}=require("./test.js");
console.log(sub(100,20));
console.log(mul(10,2));
console.log(name)
```

Write a node.js script to create calculator using external module having a function add(), sub(), mul(), div(). This function returns result of calculation. Write all necessary .js files.

<u>**1.js**</u>

```
exports.add = function (x, y) {
   return x + y;
};
exports.sub = function (x, y) {
   return x - y;
};
exports.mult = function (x, y) {
   return x * y;
};
exports.div = function (x, y) {
   return x / y;
};
```

<u>**2.js**</u>

```
const calculator = require('./1.js);

let x = 50, y = 20;
console.log("Addition of 50 and 20 is "
   + calculator.add(x, y));

console.log("Subtraction of 50 and 20 is "
   + calculator.sub(x, y));
```

```
console.log("Multiplication of 50 and 20 is "
   + calculator.mult(x, y));

console.log("Division of 50 and 20 is "
   + calculator.div(x, y));
```

Output:

```
Addition of 50 and 20 is 70
Subtraction of 50 and 20 is 30
Multiplication of 50 and 20 is 1000
Division of 50 and 20 is 2.5
```

Write all necessary .js files to create module having a function to check numbers from 2 to 50 are prime number or not.

1.js

```
const PrimeNo = (num) =>
{
   let temp = 0
   for(let i=2;i<num;i++)
   {
      if(num%i==0)
      {
         temp++;
      }
   }
   if(temp==0)
   {
      return true;
   }
   else{
      return false;
   }
}
module.exports=PrimeNo;
```

2.js

```
var PrimeNumber = require("./1.js")
for(i=2;i<=50;i++)
{
let x=PrimeNumber(i);
if(x==true)
{
   console.log(i+" Prime Number");
}
else{
   console.log(i+" Not a Prime Number")
}
}
```

Output:

```
2 Prime Number
3 Prime Number
4 Not a Prime Number
5 Prime Number ………. upto 50
```

Write a nodejs script to create my own module to calculate reverse of a given number. That module should be used to compute all numbers between 1 to 100 in which square of rev erse and reverse of square is same. These has call of reverse twice so call it from module.

1.js

```
function reversenum(num)
{
   let rev=0;
   while(num>0)
   {
      rev=rev*10+(num%10);
      num=parseInt(num/10);
   }
   return rev;
}
function square(num1)
{
   return num1*num1;
}
function checknum(num2)
```

```
{
   a=square(num2);
   b=square(reversenum(num2));
   if(a==reversenum(b))
   {
      console.log("eEqual")
   }
   else
   {
      console.log("Not equal")
   }
}
module.exports.cs = checksum
```

In another file:

```
var c1=require("./1.js");

c1.cs(12);
```

Output: Equal

NPM module

NPM is a package manager for Node.js packages, or modules if you like. The NPM program is installed on your computer when you install Node.js.

What is a Package?

A package in Node.js contains all the files you need for a module. Modules are JavaScript libraries you can include in your project.

Chalk module

In Node.js is the third-party module that is used for styling the format of text and allows us to create our own themes in the node.js project.

Advantages of Chalk Module:

1. It helps to customize the color of the output of the command-line output
2. It helps to improve the quality of the output by providing several color options like for warning message red color and many more

Steps to install chalk

- Create one folder
- Set proxy if required: npm config set proxy http://192.168.10.252:808

- To install chalk: npm install chalk or **npm i chalk**

- After installation of chalk module, package-lock.json and package.json will be created.

- Add ["type": "module",] in **package.json** file as shown below

In package.json file:

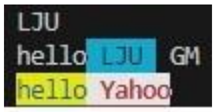
```
{
"type": "module",
  "dependencies": {
   "chalk": "^5.2.0"
  }
}
```

Note: If chalk installed successfully and package.json and package-lock.json files are not created, then try **npm init** command inside created folder. And then try to install chalk inside folder.

Example:

```
import ch from "chalk";

const log=console.log;

log("LJU");

log("hello"+ch.bgCyan(" LJU ")+" GM ")

log(ch.blue.underline.bgYellow("hello")+ch.red.bold.underline.bgWhite(" Yahoo"));
```

Output:



```
LJU
hello LJU  GM
hello Yahoo
```

Validator module

The Validator module is popular for validation. Validation is necessary to check whether the data is in format or not, so this module is easy to use and validates data quickly and easily.

Simple functions for validation like isEmail(), isEmpty() etc.

To install validator: npm install validator

Example1 : Check whether given email is valid or not

```
import validator from "validator"

let email = 'test@gmail.com'

console.log(validator.isEmail(email))           // true

email = 'test@'

console.log(validator.isEmail(email))           // false
```

Example2 : Check whether string is in lowercase or not

```
import validator from "validator"

let  name = 'hellolju'

console.log(validator.isLowercase(name))      // true

name = 'HELLOLJU'

console.log(validator.isLowercase(name))    // false
```

Example3: Check whether string is empty or not

```
import validator from "validator"

let name = ''

console.log(validator.isEmpty(name))  // true

name = 'helloLJU'

console.log(validator.isEmpty(name))  // false
```

Wrapper function

NodeJS does not run our code directly, it wraps the entire code inside a function before execution. This function is termed as Module Wrapper Function. Before a module's code is executed, NodeJS wraps it with a function wrapper that has the following structure:

```
(function (exports, require, module, __filename, __dirname) {
  //module code
});
```

The five parameters — exports, require, module, __filename, __dirname are available inside each module in Node. Though these parameters are global to the code within a module yet they are local to the module (because of the function wrapper as explained above). These parameters provide valuable information related to a module.

The variables like **__filename** and **__dirname**, that tells us the module's absolute filename and its directory path.

The meaning of the word 'anonymous' defines something that is unknown or has no identity. In JavaScript, an anonymous function is that type of function that has no name or we can say which is without any name.

Example:

```
(
  function()
    {
        console.log(__filename);
        console.log(__dirname);
     }
 )();
```
Output:

```
D:\Trynode\ex1.js //returnd path of current file
D:\Trynode     //returned path till current file (folder)
```

Event Module

If you worked with JavaScript in the browser, you know how much of the interaction of the user is handled through events: mouse clicks, keyboard button presses, reacting to mouse movements, and so on.
On the backend side, Node.js offers us the option to build a similar system using the events module.
This module, in particular, offers the EventEmitter class, which we'll use to handle our events.
You initialize that using

```
const EventEmitter = require('events');
const eventEmitter = new EventEmitter();
```

emit is used to trigger an event
on is used to add a callback function that's going to be executed when the event is triggered

Syntax:

```
eventEmitter.emit(event, [arg1], [arg2], [...])

eventEmitter.on(event, listener)

eventEmitter.addListener(event, listener)
```

eventEmmitter.on(event, listener) and **eventEmitter.addListener(event, listener)** are pretty much similar. It adds the listener at the end of the listener's array for the specified event. Multiple calls to the same event and listener will add the listener multiple times and correspondingly fire multiple times. Both functions return emitter, so calls can be chained.

Example-1:

```
const E = require('events');

const ee = new E();

ee.on('start', () => {
  console.log('started');

 });

 ee.emit('start');
```

```
const E = require('events');

const ee = new E();
ee.on('start', (start, end) => {
  console.log(`started from ${start} to ${end}`);
 });

 ee.emit('start', 1, 100);
```

Output: started from 1 to 100

```
var e=require("events");
var ee=new e();

ee.on("connection",function(){
  console.log("Connection successfully");
  ee.emit("data-received");
});
ee.on("data-received",function()
{
  console.log("data received successfully");
})
ee.emit("connection");
console.log("thanks");
```

Output:

```
Connection successfully
data received successfully
thanks
```

Removing Listener:

The **eventEmitter.removeListener()** takes two argument event and listener, and removes that listener from the listeners array that is subscribed to that event. While **eventEmitter.removeAllListeners()** removes all the listener from the array which are subscribed to the mentioned event.

Syntax:

eventEmitter.removeListener(event, listener)

eventEmitter.removeAllListeners([event])

Note:

- Removing the listener from the array will change the sequence of the listener's array, hence it must be carefully used.
- The **eventEmitter.removeListener()** will remove at most one instance of the listener which is in front of the queue.

Note: Line A, B, C, D are added to understand output scenario.

```
// Importing events
const EventEmitter = require('events');

// Initializing event emitter instances
var eventEmitter = new EventEmitter();

var fun1 = (msg) => {
   console.log("Message from fun1: " + msg);
};

var fun2 = (msg) => {
   console.log("Message from fun2: " + msg);
};
// Registering fun1, fun2
eventEmitter.on('myEvent', fun1); //executes fun1 function on event myEvent "Line A"
eventEmitter.on('myEvent', fun2); //executes fun2 function on event myEvent  "Line B"
eventEmitter.on('myEvent', fun1); //executes fun1 function on event myEvent  "Line C"
eventEmitter.on('myEvent2', fun2); //executes fun2 function on event myEvent2  " Line D"

// Removing listener fun2
eventEmitter.removeListener('myEvent', fun2); //remove fun2 "line B"

// Triggering myEvent
eventEmitter.emit('myEvent', "LJU"); //Executes two functions fun1 (Line A) and fun1 (Line C)

// Removing all the listeners to myEvent
eventEmitter.removeAllListeners('myEvent'); // This will remove all listened events. Means fun1
(Line A) and fun1 (Line C)

eventEmitter.emit('myEvent2', "LJU"); // Executes function fun2 Line D
```

Output:

```
Message from fun1: LJU //Line A
Message from fun1: LJU //Line C
Message from fun2: LJU // Line D
```

Examples

Write node js script to handle events as asked below.

1) **Check the radius is negative or not. If negative then display message “Radius” must be positive” else calculate the perimeter of circle.**
2) **Check side is negative or not. If negative then display message “Side must be positive” else calculate the perimeter of square.**

```
var e = require("events");
var ee = new e();
const radiushandler = () => {
  console.log("Radius must be positive");
}
ee.on("negside", sidehandler = () => {
  console.log("Side must be positive");
});

const findval = (r,s) => {

  if (r < 0) {
    ee.emit("negradius");
  }else{
    var rperi = 2 * 3.14 * r;
    console.log(rperi);
  }
  if (s < 0)
  {
    ee.emit("negside");
  }
  else{
    var speri = 4*s;
    console.log(speri);
  }
}
```

```
ee.on("negradius", radiushandler);
//ee.on("negside", sidehandler);
ee.on("findval", findval);
ee.emit("findval",-2,-3);
```

Write node js script to handle event of write a data in file, append data in file and then read the file and display data in console.

```
var e=require("events");
var fs=require("fs");
var ee=new e();

ee.on("connection",function()
{
   fs.writeFile("b.txt","This is data",(err)=> {console.log()});
   console.log("File Written");
   ee.emit("data-append");
   ee.emit("data-read");
});
ee.on("data-append",function()
{
   fs.appendFile("b.txt","That is data",(err)=> {console.log()});
   console.log("File Appended");
});
ee.on("data-read",function()
{
   fs.readFile("b.txt",(err,data)=>
   {
      if(err){
      console.error(err);
      }
      console.log(data.toString());
   });
});
ee.emit("connection");
```