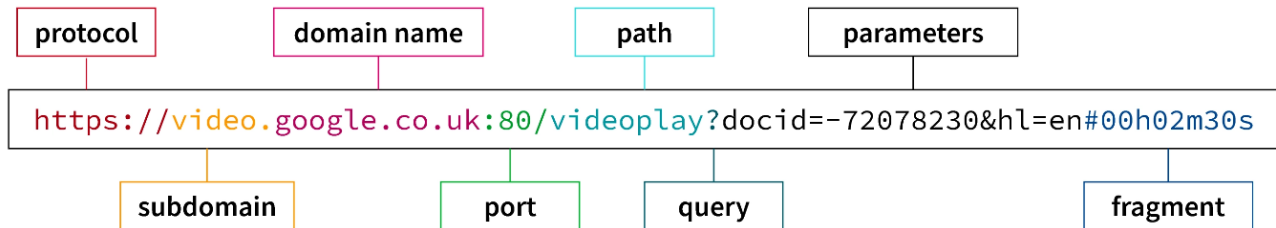# URL module

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.

Examples

Write node js script to fetch the querystring from the url and check if year is leap year or not.

```
var u=require("url");
var addr="http://localhost:8080/default.html?year=2025&month=feb";
var q=u.parse(addr,true);
console.log(q);
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

Write a nodejs script to print query string of url in file using ES6 callback. (Async File System).

Scenario 1 : Fetch query as an object. Need to convert query to string using stringify and then we can write it 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");
});
```

Scenario 2: Fetch query as a string and we can write it 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,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 external 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.
- ✓ This keyword tells Node.js that the function can be used outside the module.

Method 1

In test.js file:

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

In another file test1.js where you want to use that module:

```
var a1=require("./test.js");
console.log(a1(10,15));
```

to run -> **node test1.js**

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 test1.js:

```
var a1=require("./test.js");
console.log(a1.s(10,5));
console.log(a1.m(10,15));
```

to run -> **node test1.js**

Method 3

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 test1.js:

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

to run -> **node test1.js**

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)
```

to run -> **node test1.js**

Method 5

   In test.js file:

```
exports.add = function (x, y) {
   return x + y;
};
```

In another file:

```
const a1 = require('./test.js');
console.log("Addition is: "
   + a1.add(50, 20));
```

to run -> **node test1.js**

Examples

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.

1.js

```
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;
};
```

2.js

```
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 own module to calculate reverse of a given number. That mod ule should be used to check given number of which square of reverse and reverse of square i s same. (ADAM NUMBER)

For Example,

12 (12^2 =144)

 21 (21^2=441)

144 = reverse(441)

```
function reversenum(num)
{
   let rev=0;
   while(num>0)
   {
     rev=rev*10+(num%10);
     //num=12 (0*10+12%10)  rev=2
     //num=1 (2*10+1%10)   rev=21
     //num=441 (0*10+441%10) rev=1
     //num=44  (1*10+44%10) rev=14
     //num=4  (14*10+144%10) rev=144

     num=parseInt(num/10);
     //num 1,num=0,num=44,num==4,num=0
   }
   return rev;
}
function square(num){ return num*num; }
function checknum(num)
{
   a=square(num) //a=144
   b=square(reversenum(num)) //b=441
```

```
   if(a==reversenum(b)) { console.log("Equal") }
   else{ console.log("Not equal") }
}
module.exports.checknum = checknum
```

c2.js

```
var c1=require("./c1.js");
c1.checknum(12);
```

Chalk module

In Node.js is the third-party module that is used for styling the format of text.

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 installating a 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:

Type	Available Values
Colors	`black, red, green, yellow, blue, magenta, cyan, white, gray`
Bright Colors	`redBright, greenBright, yellowBright, blueBright, magentaBright, cyanBright, whiteBright`
Background Colors	`bgBlack, bgRed, bgGreen, bgYellow, bgBlue, bgMagenta, bgCyan, bgWhite`
Bright Backgrounds	`bgBlackBright, bgRedBright, bgGreenBright, bgYellowBright, bgBlueBright, bgMagentaBright, bgCyanBright, bgWhiteBright`
Text Styles	`bold, dim, italic, underline, inverse, strikethrough`
Custom Colors	`rgb(r, g, b), hex("#RRGGBB")`

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(), isLowercase() 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
```

cv.js

```
import ch from "chalk";
import validator from "validator"

var test = ch.red.underline.bgYellow("hello")+ch.bold.bgRed.italic.yellow("\nyahoo")
console.log(test)

console.log(validator.isLowercase(test),
ch.red.underline.bold.bgWhite(validator.isEmail(test)))
```


## Reference Folder Structure for chalk and validator module

- **Create a folder named cv.**
- **Run the command "npm init" inside that folder.**
- **Install chalk and validator inside that folder using the command:**

npm install chalk
npm install validator
or
npm install chalk validator

- **Create the files c1.js, v1.js, and cv.js inside the cv folder.**
- **Perform the required tasks in those files.**
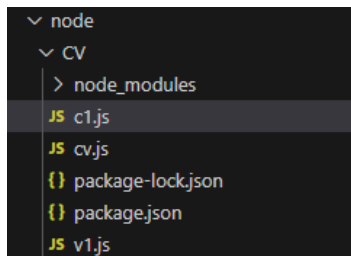
Wrapper function

✓ 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.

✓ 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.

Example:

```
(
  function()
    {
        console.log(__filename);
        console.log(__dirname);
    }
  )();
```
Output:
D:\Trynode\ex1.js //returned path of current file
D:\Trynode   //returned path till current file (folder)

Event Module

✓ In Node.js, an event emitter is like a messenger that helps different parts of a program communicate with each other. Imagine it as a broadcaster at a big event, letting everyone know what's happening.

✓ **Messenger** Role: Just like a messenger, an event emitter sends messages, or "events," to whoever is interested in hearing them. These events could be anything – a button click, a file being loaded, or a connection being established.

✓ **Listeners**: There are listeners, or the audience, who are interested in specific types of events. These listeners are set up to pay attention to certain messages. When the event emitter sends out an event, the relevant listeners respond to it.

✓ **Event** Types: Events have types, like categories. For instance, a 'click' event or a 'data received' event. When you're setting up an event emitter, you define these types, so others know what to expect.

✓ **Example**: Think of a light switch. The switch itself is like the event emitter. When you turn it on or off, it emits an event – the change in state. Now, imagine a smart home system with different devices, like lights and speakers. Each of these devices is a listener. They're interested in the 'state change' event. So, when the switch is toggled, these devices react accordingly.

✓ 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 <u>events module</u>.

You initialize that using

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

✓ **emit** is used to trigger an event

✓ **on/addListener** 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.

In code, using an event emitter in Node.js looks something like this:

```
const eventemitter = require('events');
const ee = new eventemitter();
ee.on('start', () => {
  console.log('started');
 });
 ee.emit('start');
```

In this example, when the 'start' event is emitted, the listener responds by printing 'Started' to the console.

So, in Node.js, an event emitter is a handy tool for making different parts of a program talk to each other by sending and receiving messages. It's a way to create responsive and interactive applications.

Example with arguments

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

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

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

Output: started from 1 to 100

Example:

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

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.

Examples

```
//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('myEvent1', fun1); //executes fun1 function on event myEvent "Line A"
eventEmitter.on('myEvent2', fun2); //executes fun2 function on event myEvent  "Line B"
eventEmitter.on('myEvent1', fun1); //executes fun1 function on event myEvent  "Line C"
eventEmitter.on('myEvent2', fun2); //executes fun2 function on event myEvent2  " Line D"

// Removing a listener of myEvent2
eventEmitter.removeListener('myEvent2', fun2); //remove fun2 "line B"

// Removing all the listeners of myEvent1
eventEmitter.removeAllListeners('myEvent1'); // This will remove all events named
"myEvent1" Means (Line A) and  (Line C)

// Triggering myEvent2
eventEmitter.emit('myEvent2', "LJ University"); //Executes a function fun2 (Line D)
eventEmitter.emit('myEvent1', "LJU"); // Nothing will be displayed as all events have been
removed

Output:
Message from fun2: LJ University
```

Write node js script to create two listeners for a common event. Print number of events associated with an emitter. Remove one of the listeners and call remaining listener again. Also, print number of remaining listeners.

```
const EventEmitter = require('events');
var ee = new EventEmitter();
var listener1 = function listen1() {
    console.log("Listener1 executed");
};
var listener2 = function listen2() {
    console.log("Listener2 executed");
};
ee.on("conn",listener1)  // addListener/on both functions perform same task. You can use
addListener or on to execute a task.
ee.on("conn", listener2);

//Count initially all the events.
let count = ee.listenerCount("conn");
console.log("Count 1: " + count );
ee.emit("conn");

//To remove 1st listner
ee.removeListener('conn', listener1);
count = ee.listenerCount("conn");
console.log("Counting again: " + count );
ee.emit("conn");

// Above program ends here as per the question. Below is additional task of remove all
listeners. and count the listener.

//To remove all listners
ee.removeAllListeners('conn', listener1);
count = ee.listenerCount("conn");
console.log("Again Count afetr removing all listeners: " + count );
ee.emit("conn");
```

eventEmitter.listenerCount(): It returns the number of listeners listening to the specified event.

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 eventemitter = require("events");
var ee = new eventemitter();

ee.on("negradius", () => {
   console.log("Radius must be positive");
});
ee.on("negside", () => {
   console.log("Side must be positive");
});
ee.on("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.emit("findval",10,3);
```

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("data-write",function()
{
   fs.writeFile("b.txt","Hello, ",(err)=> {console.log()});
   console.log("Data Written");
   ee.emit("data-append");
   ee.emit("data-read");
});
ee.on("data-append",function()
{
   fs.appendFile("b.txt","Good Morning!",(err)=> {console.log()});
   console.log("Data Appended");
});
ee.on("data-read",function()
{
   fs.readFile("b.txt","utf-8",(err,data)=>
   {
      if(err){
      console.error(err);
      }
      console.log(data);
   });
});
ee.emit("data-write");
```

Miscellaneous

Write node.js script to create a class named person by assigning name and age in form of members. Create two objects and a method named elder which returns elder person object. Details of elder person should be printed in console as well as in file.

```
class person
{
   constructor(name,age)
   {
      this.age=age;
      this.name=name;
   }
   elder(p)
   {
      if(this.age>p.age)
      {
         return this;
      }
      else{
         return p;
      }
   }
}
var p1= new person("xyz",23);
var p2= new person("abc",34);
var p3=p1.elder(p2);
const jsonstr=JSON.stringify(p3);
var ps=require("fs");
ps.writeFileSync("d2.txt",jsonstr);
```
Output:
person { age: 34, name: 'abc' }

Write node.js script to create a class named time and assign members hour, minute and second. Create two objects of time class and add both the time objects so that it should return the value in third time object. The third time object should have hour , minute and second such that if seconds exceed 60 then minute value should be incremented and if minute exceed 60 then hour value should be incremented. The value should be printed in console as well as in file.

```
class time
{
  constructor(hour,min,sec)
  {
    this.hour=hour;
    this.min=min;
    this.sec=sec;
  }
  timer(p)
  {
    var t=new time();
    t.hour=this.hour+p.hour;
    t.min=this.min+p.min;
    t.sec=this.sec+p.sec;
    if(t.sec>60)
    {
      t.sec%=60;
      t.min++;

    }
    if(t.min>60)
    {
      t.min%=60;
      t.hour++;
    }
    return t;
  }
}
var t1= new time(1,50,50);
var t2= new time(2,30,50);
```

```
var t3=t1.timer(t2);
console.log(t3);

const jsonstr=JSON.stringify(t3);
var ps=require("fs");
ps.writeFileSync("time.txt",jsonstr);
```

Output:
time { hour: 4, min: 21, sec: 40 }