



COMMAND LINE. DEBUGGING. ERROR HANDLING

September 25, 2017

AGENDA

- Node.JS REPL
- Reading from the command line
- Reading environment variables
- Console
- Logging and loggers
- Debugging Node.JS
- Error handling



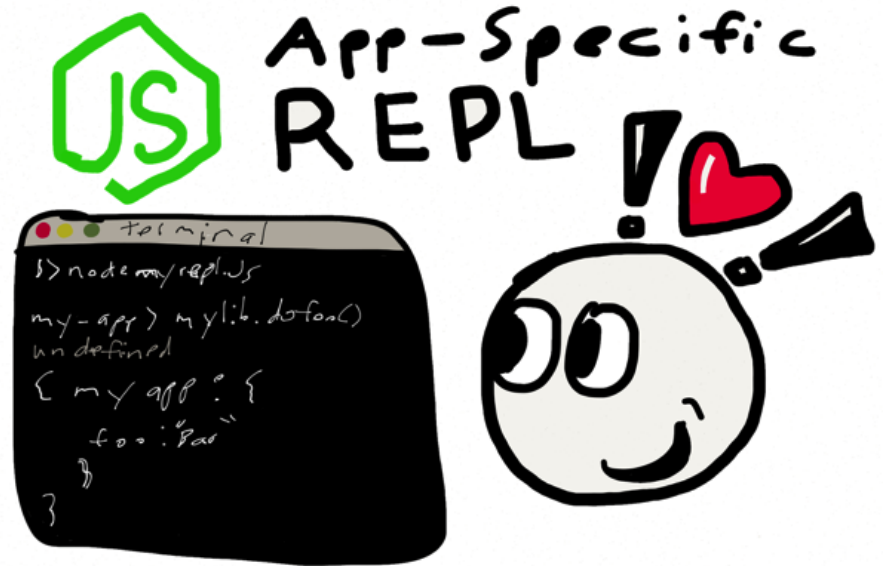
NODE REPL (READ-EVALUATE-PRINT LOOP)

- REPL stands for Read Evaluate Print Loop.
- Represents an interactive environment.
- Comes together with Node.JS.
- Is useful in experimenting with Node.JS

```
○ → node
> console.log('this is REPL')
this is REPL
undefined
> new Date()
2017-09-23T13:08:06.971Z
> encodeURIComponent('this is REPL')
'this%20is%20REPL'
> process.cwd()
'/Users/galina_kasatkina/Documents/lecture'
> require.extensions
{ '.js': [Function], '.json': [Function], '
.node': [Function] }
```

CUSTOM NODE REPL

```
1  const repl = require('repl');
2  const colors = require('colors');
3
4  let replServer = repl.start({
5    prompt: 'j> _> }>'.red,
6    useColors: true,
7    ignoreUndefined: true
8  });
9
10 replServer.context.fs = require('fs');
```



COMMAND LINE ARGUMENTS

o → node 1.custom-repl.js --abc=56 --def=75

⌋ ⌋ ●_● ⌋ process.argv

```
[ '/Users/galina_kasatkina/.nvm/versions/node/v7.4.0/bin/node',  
  '/Users/galina_kasatkina/Documents/lecture/1.custom-repl.js',  
  '--abc=56',  
  '--def=75' ]
```

⌋ ⌋ ●_● ⌋ process.argv[2]

```
'--abc=56'
```

⌋ ⌋ ●_● ⌋ process.argv[2].split('=')

```
[ '--abc', '56' ]
```

⌋ ⌋ ●_● ⌋ process.argv[2].split('=')[1]

```
'56'
```

NPM PACKAGES: COMMANDER, MINIMIST, YARGS

Modified	3 hours ago	8 days ago	9 hours ago
Total Versions	47	19	141
Version Average	every 2 months	every 3 months	every 10 days
Maintainers	5	1	3
Dependencies	0	0	13
Daily Downloads	1,255,396	1,695,028	1,505,452
Weekly Downloads	7,585,000	10,678,475	9,019,023
Monthly Downloads	31,869,721	44,017,499	36,383,819
Open Issues	203	56	129
Open Pull Requests	30	22	6
Stargazers	8,824	2,456	3,452
Subscribers	183	34	63

COMMAND LINE ARGUMENTS

o → node 1.custom-repl.js flag1 flag2 --abc=56 --def=75 --help --no-thanks

-- --size=large

```
{ } { } let parseArgs=require('minimist');
```

```
{ } { } parseArgs(process.argv)
```

```
{ _:
```

```
  [ '/Users/galina_kasatkina/.nvm/versions/node/v7.4.0/bin/node',
```

```
    '/Users/galina_kasatkina/Documents/lecture/1.custom-repl.js',
```

```
    'flag1',
```

```
    'flag2',
```

```
    '--size=large' ],
```

```
  abc: 56,
```

```
  def: 75,
```

```
  help: true,
```

```
  thanks: false }
```



COMMAND LINE ARGUMENTS

```
1  const parseArgs = require('minimist');
2  console.log(parseArgs(process.argv.slice(2), {
3    alias: { 'help': 'h' },
4    default: { 'help': true },
5    unknown: (arg) => {
6      if (arg !== 'help' && arg !== 'h' && arg !== 'abc') {
7        console.error('Unknown option: ', arg);
8        return false;
9      }
10    }
11  }));
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

o → node 2.read-params.js --help --unknown

Unknown option: --unknown

{ _: [], help: true, h: true }



ENVIRONMENT VARIABLES

- `process.env` returns an object containing the user environment
- `process.env` modifications will not be reflected outside the Node.js process
- Assigning a property on `process.env` will implicitly convert the value to a string
- Use `delete` to unassign a variable

○ → node

> process.env

```
{ GREP_COLOR: '1;33',  
  MANPATH: '/Users/galina_kasatkina/.nvm/versions/node/v  
ions/Wireshark.app/Contents/Resources/share/man:/Library  
TERM_PROGRAM: 'vscode',  
  NVM_CD_FLAGS: '',  
  TERM: 'xterm-256color',  
  SHELL: '/bin/bash',  
  HISTSIZE: '5000',  
  TMPDIR: '/var/folders/32/1243gjtd5t35z4hzny_d594jz80d9  
Apple_PubSub_Socket_Render: '/private/tmp/com.apple.la  
TERM_PROGRAM_VERSION: '1.16.0',
```

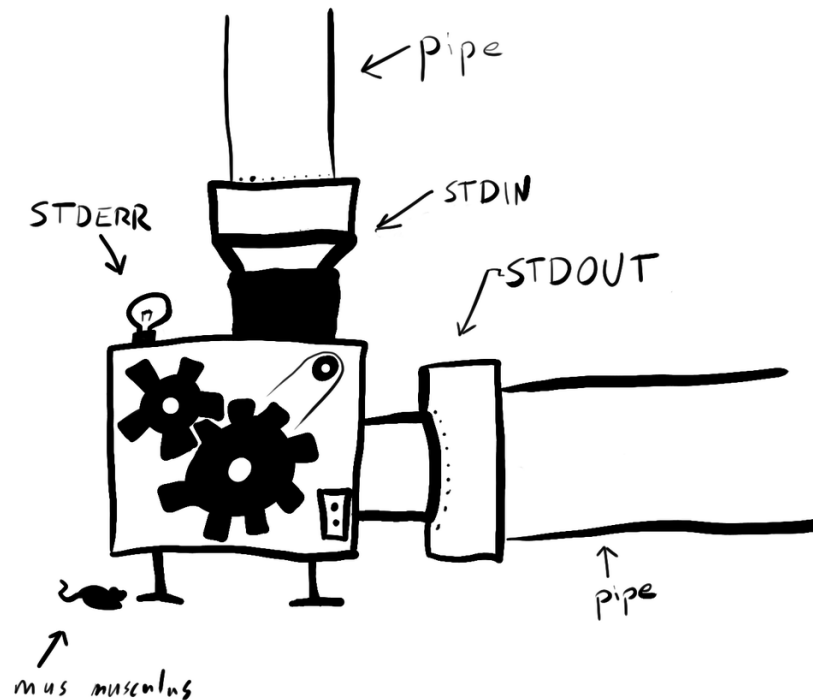
CONSOLE

- **console** is a global variable.
- Useful methods:
 - **console.log** (= **console.info**)
 - **console.error** (= **console.warn**)
 - **console.assert**
 - **console.trace**
 - **console.time/timeEnd**

```
> console.time('test')
> console.timeEnd('test')
test: 7166.297ms
> console.trace()
Trace
    at repl:1:9
    at ContextifyScript.Script.runInContext (vm.js:37:29)
    at REPLServer.defaultEval (repl.js:348:29)
    at bound (domain.js:280:14)
    at REPLServer.runBound [as eval] (domain.js:293:12)
    at REPLServer.onLine (repl.js:544:10)
    at emitOne (events.js:96:13)
    at REPLServer.emit (events.js:188:7)
    at REPLServer.Interface._onLine (readline.js:247:10)
    at REPLServer.Interface._line (readline.js:591:8)
```

STANDARD STREAMS

STREAM	DEFINITION	BY DEFAULT
stdin	Standard input (fd 0)	Accepts input from terminal
stdout	Standard output (fd 1)	Outputs to terminal
stderr	Standard error output (fd 2)	Outputs to terminal



CONSOLE: OUTPUT TO STDOUT AND STDERR

JS 3.console.js x

```
1 console.log('regular info');
2 console.error('ERROR!!!!');
3
4
5
6
```

≡ debug.log x

```
1 regular info
2
3
4
5
6
```

≡ error.log x

```
1 ERROR!!!!
2
3
4
5
6
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

→ node 3.console.js 1>debug.log 2>error.log

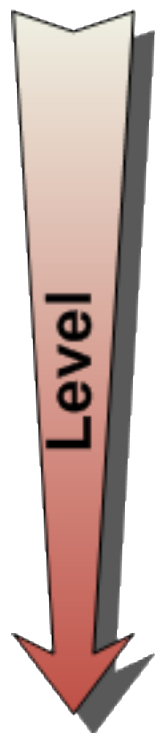
LOGGING

- **Business statistics** - logins, purchases, registrations, unsubscribes
- **Errors** - resource exhaustion, uncaught exceptions, connection failures
- **Debug information** - method calls, event triggers, connections, access to resources

LOG ALL THE THINGS



LOGGING LEVELS



DEBUG

fine-grained informational events that are most useful to debug an application

INFO

informational messages that highlight the progress of the application at coarse-grained level

WARN

potentially harmful situations

ERROR

error events that might still allow the application to continue running

FATAL

very severe error events that will presumably lead the application to abort

LOGGERS: BUNYAN, DEBUG, WINSTON

Modified	18 days ago	a day ago	11 days ago
Total Versions	106	52	42
Version Average	every 19 days	every a month	every 2 months
Maintainers	1	4	6
Dependencies	4	1	6
Daily Downloads	31,817	1,012,298	81,959
Weekly Downloads	414,789	13,332,015	1,264,180
Monthly Downloads	1,699,318	52,042,173	5,045,866
Open Issues	182	31	302
Open Pull Requests	30	6	30
Stargazers	4,407	4,881	8,378
Subscribers	108	111	182

LOGGING: DEBUG

```
1  const express = require('express'),
2      app = express(),
3      debug = require('debug')('app:server');
4
5  debug('booting app');
6  app.get('/', require('./handler'))
7      .listen(3000, function () {
8      |   debug('listening');
9      | });
10
```

```
1  const debug = require('debug')('app:handler');
2
3  module.exports = function (req, res) {
4      |   debug(req.method + ' ' + req.url);
5      |   res.end('hello\n');
6      | }
7
8
9
10
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

○ → DEBUG=app:* node 3.logging.js

app:server booting app +0ms

app:server listening +13ms

app:handler GET / +0ms

app:handler GET / +2s

LOGGING: DEBUG

- Simplest logger with minimum dependencies.
- Can create multiple loggers with different IDs.
- Loggers can be turned on and off using the environment variable DEBUG.
- No log levels but can be added with [debug-levels](#) package.
- Outputs to the standard error stream.
- Mostly used for debugging purposes.
- Repository: <https://github.com/visionmedia/debug>



ONE BIG MAMA
OF A BUG



LOGGING: WINSTON

```
1  const logger = require('winston');
2  module.exports = function (req, res) {
3    logger.info('Request: ' + req.method + ' ' + req.url);
4    if (req.path === '/cats' || req.path === '/dogs') {
5      logger.debug('IP: ' + req.ip);
6      res.end('hello\n');
7      return;
8    }
9    logger.error(req.path + ' - unknown route');
10   res.status(404).end('Not found');
11 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

o → node 5.logging-winston.js

2017-09-24T12:26:51.427Z - info: Got message: GET /cats

2017-09-24T12:26:55.722Z - info: Got message: GET /dogs

2017-09-24T12:26:58.936Z - info: Got message: GET /flies

2017-09-24T12:26:58.936Z - error: /flies - unknown route



LOGGING: WINSTON TRANSPORTS

```
let transports = [  
  new winston.transports.Console({  
    timestamp: true,  
    colorize: true,  
    level: 'info'  
  }),  
  new winston.transports.File({  
    filename: 'debug.log',  
    name: 'debug',  
    level: 'debug'  
  }),  
  new winston.transports.File({  
    filename: 'error.log',  
    name: 'error',  
    level: 'error'  
  })  
];  
  
return new winston.Logger({transports: transports});
```

≡ debug.log x

```
1 {"level":"info","message":"Request: GET /cats",  
  "timestamp":"2017-09-24T12:36:45.055Z"}  
2 {"level":"debug","message":"IP: ::1",  
  "timestamp":"2017-09-24T12:36:45.057Z"}  
3 {"level":"info","message":"Request: GET /dogs",  
  "timestamp":"2017-09-24T12:36:48.428Z"}  
4 {"level":"debug","message":"IP: ::1",  
  "timestamp":"2017-09-24T12:36:48.429Z"}  
5 {"level":"info","message":"Request: GET /flies",  
  "timestamp":"2017-09-24T12:36:51.237Z"}  
6 {"level":"error","message":"/flies - unknown route",  
  "timestamp":"2017-09-24T12:36:51.237Z"}
```

≡ error.log x

```
1 {"level":"error","message":"/flies - unknown route",  
  "timestamp":"2017-09-24T12:36:51.237Z"}
```

LOGGING: WINSTON

```
9      let transports = [  
10         new winston.transports.Console({  
11             timestamp: function () {  
12                 return Date.now();  
13             },  
14             formatter: function (options) {  
15                 return 'New format! ' + options.timestamp() + ' ' + options.level.toUpperCase() +  
16                     ' ' + (options.message ? options.message : '');  
17             }  
18         })),
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

o → node 5.logging-winston.js

New format! 1506261180965 INFO Request: GET /cats

New format! 1506261184488 INFO Request: GET /dogs

New format! 1506261187732 INFO Request: GET /flies

New format! 1506261187733 ERROR /flies - unknown route

LOGGING: WINSTON

- Logging to:
 - Console,
 - Files
 - Databases (Redis, MongoDB),
 - Online services (ElasticSearch)
- Configurable logging levels
- Configurable timestamps
- Configurable output format
- Supports both string and JSON format
- Log rotation:
 - Maximum file size
 - Maximum file count
 - Zipping old files

LOGGING: MORGAN

```
1  const express = require('express'),
2    morgan = require('morgan'),
3    app = express();
4
5  app.use(morgan('combined'));
6  app.get(/.*/, require('./handler'))
7    .listen(3000);
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

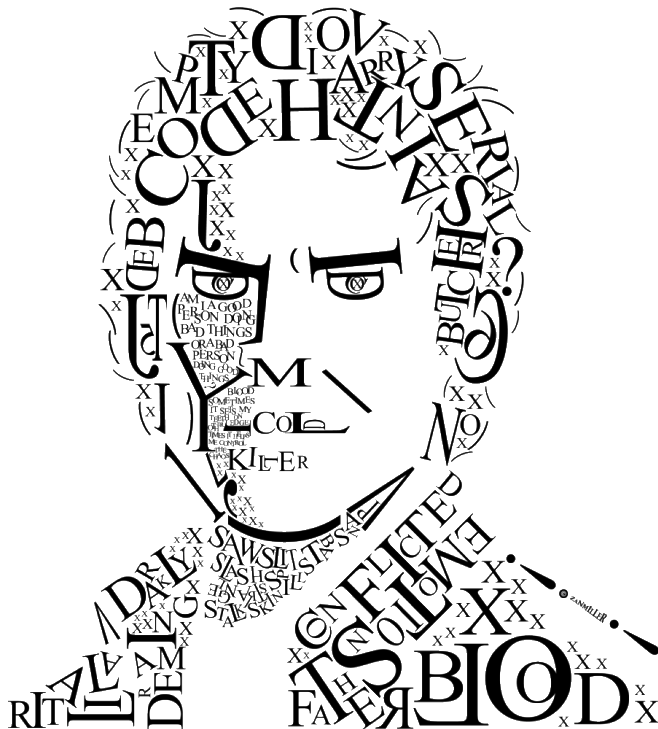
TERMINAL

2: node

- node 4.logging-morgan.js

```
:::1 - - [24/Sep/2017:14:13:40 +0000] "GET /cats HTTP/1.1" 200 - "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36"

:::1 - - [24/Sep/2017:14:13:43 +0000] "GET /dogs HTTP/1.1" 200 - "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.91 Safari/537.36"
```



LOGGING: MORGAN

```
1  const express = require('express'),
2      morgan = require('morgan'),
3      app = express();
4
5  app.use(morgan(':date[iso] :url'));
6  app.get(/.*/, require('./handler'))
7      .listen(3000);
```

PROBLEMS

OUTPUT

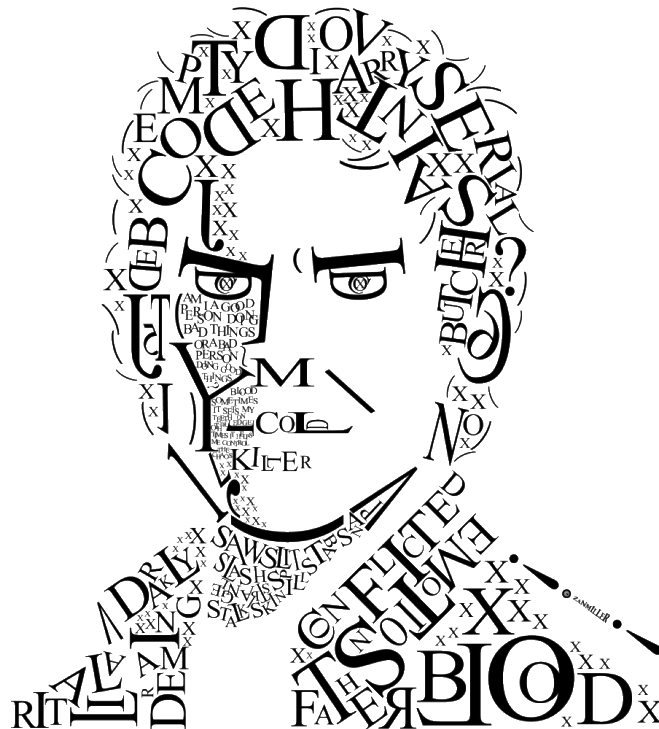
DEBUG CONSOLE

TERMINAL

- node 4.logging-morgan.js

2017-09-24T14:25:46.199Z /cats

2017-09-24T14:25:48.809Z /cats22

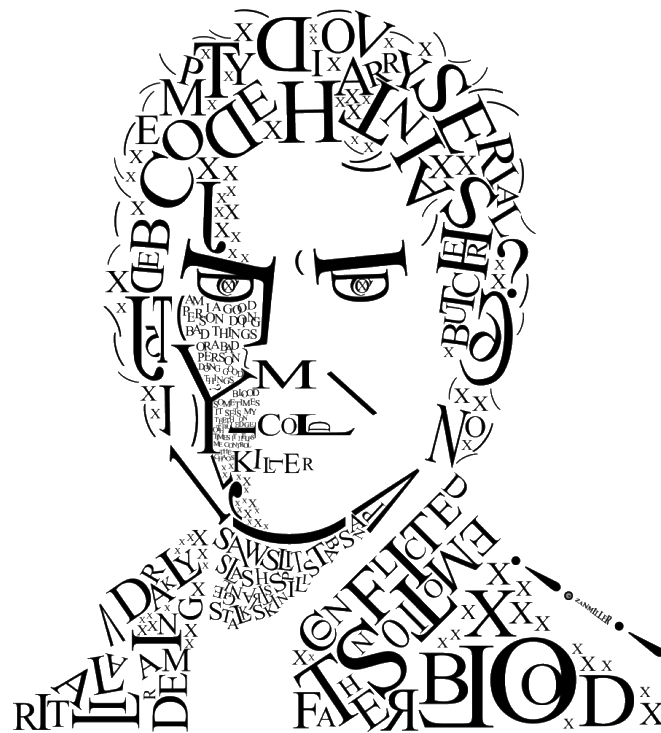


LOGGING: MORGAN TOKENS

- :date[format]
 - CLF for the common log format
 - ISO (ISO 8601)
 - web (default, RFC 1123)
- :http-version
- :method
- :referrer
- :remote-addr
- :remote-user
- :req[header]
- :res[header]
- :response-time[digits]
- :status
- :url
- :user-agent

LOGGING: MORGAN

- Used as middleware with HTTP servers.
- Creates access logs.
- Configurable format using predefined tokens.
- Can log to any writable stream.
- Log rotation using [rotating-file-stream](#).



DEBUGGING: NODE-INSPECTOR

Devices

☒ Discover USB devices

Port forwarding...


☒ Discover network targets

Configure...

[Open dedicated DevTools for Node](#)

Remote Target #LOCALHOST

Target (v8.5.0)

 4.logging-morgan.js file:///Users/galina_kasatkina/Documents/lecture/4.logging-morgan.js
[inspect](#)

DEBUGGING: VSCODE

```
{  
  "type": "node",  
  "request": "launch",  
  "name": "Launch Program",  
  "program": "${file}"  
}
```

```
{  
  "type": "node",  
  "request": "launch",  
  "name": "nodemon",  
  "runtimeExecutable": "nodemon",  
  "program": "${file}",  
  "restart": true,  
  "console": "integratedTerminal",  
  "args": ["--abc"],  
  "env": {"NODE_DEBUG": "development"},  
  "stopOnEntry": true,  
  "skipFiles": ["node_modules/**/*.js"]  
},
```

DEBUGGING: VSCODE

DEBUG Launch Program [Settings] [Run and Debug]

JS 4.logging-morgan.js [Run] [Run and Debug] [Attach] [Detach] [Restart]

VARIABLES

- Local
 - this: global
 - req: IncomingMessage {_readableState: Readab...
 - res: ServerResponse {domain: null, _events: ...}
- Closure
- Global

WATCH

- console: Console {log: , info: , warn: , ...}

CALL STACK PAUSED ON BREAKPOINT

module.exports	handler.js	4:5
handle	layer.js	95:5
next	route.js	137:13
dispatch	route.js	112:3

BREAKPOINTS

- ☐ All Exceptions
- ☒ Uncaught Exceptions
- ☒ handler.js 4

```
1 const debug = require('debug')('app:handler');
2
3 module.exports = function (req, res) {
4   debug(req.method + ' ' + req.url);
5   res.end('hello\n');
6 }
7
8
9
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

○ → []

ERROR HANDLING: TYPES OF ERRORS

OPERATIONAL ERRORS

- failed to connect to server
- failed to resolve hostname
- invalid user input
- request timeout
- server returned a 500 response
- socket hang-up
- system is out of memory

SHOULD BE HANDLED

PROGRAMMING ERRORS

- tried to read property of "undefined"
- called an asynchronous function without a callback
- passed a "string" where an object was expected
- passed an object where an IP address string was expected

SHOULD NOT BE HANDLED

ERROR HANDLING: OPTIONS

- **Throw an error and generate exception**

Only used in synchronous scenarios, mostly when parsing data.

- **Invoke an error-first callback**

Typical for standard Node modules and most of Node applications.

- **Reject a promise**

Any callback-based function can be promisified.

- **Generate an error event**

Used when working with EventEmitters and in larger applications.

ERROR HANDLING: TRY....CATCH

```
1  try {
2    |    JSON.parse('Not a JSON!');
3  } catch(e) {
4    |    console.log('parsing error');
5  }
6
7  try {
8    |    setTimeout(() => {
9    |      |    JSON.parse('Not a JSON!');
10   |    }, 1000);
11 } catch(e) {
12 |    console.log('callback from error');
13 }
```

o → node 6.error-handling.js

parsing error

undefined:1

Not a JSON!

^

SyntaxError: Unexpected token N in JSON at position 0
at JSON.parse (<anonymous>)
at Timeout.setTimeout [as _onTimeout] (/Users/galina_kasatkina/Documents/lecture/6.error-handling.js:9:14)

at ontimeout (timers.js:365:14)

at tryOnTimeout (timers.js:237:5)

at Timer.listOnTimeout (timers.js:207:5)

ERROR HANDLING: ERROR-FIRST CALLBACK

```
1  const fs = require('fs');
2  fs.readFile('nonexistent', (err, data) => {
3      if (err) {
4          console.log(err);
5          return;
6      }
7      //do sth
8  });
```


ERROR HANDLING: PROMISE REJECTION

```
1  const promisify = require("util").promisify;
2  const fs = require('fs');
3  const readFile = promisify(fs.readFile);
4
5  readFile('nonexistent')
6    .then((data) => {
7      JSON.parse(data);
8    }).catch((err) => {
9      console.log(err);
10    });
```



ERROR HANDLING: ERROR EVENTS

```
1  const http = require('http');
2  const server = http.createServer((req, res) => {
3    res.end('Hello!')
4  });
5
6  server.on('error', (err) => {
7    console.error('ERROR!!!');
8    console.error(err);
9  });
10
```

```
ERROR!!!
{ Error: listen EACCES 0.0.0.0:80
  at Object._errnoException (util.js:1026:11)
  at _exceptionWithHostPort (util.js:1049:20)
  at Server.setupListenHandle [as _listen2] (net.js:1
326:19)
  at listenInCluster (net.js:1391:12)
  at Server.listen (net.js:1474:7)
  at Object.<anonymous> (/Users/galina_kasatkina/Docu
ments/lecture/7.error-handling-events.1.js:11:8)
  at Module._compile (module.js:624:30)
  at Object.Module._extensions..js (module.js:635:10)
  at Module.load (module.js:545:32)
  at tryModuleLoad (module.js:508:12)
  code: 'EACCES',
  errno: 'EACCES',
  syscall: 'listen',
  address: '0.0.0.0',
  port: 80 }
```

ERROR HANDLING: ERROR CODES

- EACCES - permission denied
- EADDRINUSE - address already in use
- ECONNREFUSED - connection refused
- ECONNRESET - connection reset by peer
- EEXIST - file exists
- EISDIR - is a directory
- ENOTDIR - not a directory
- ENOENT - no such file or directory



ERROR HANDLING: UNCAUGHT EXCEPTIONS

- Can be caught using `process.on('uncaughtException')`.
- What you SHOULD NOT do in the handler:
 - Attempt to restore the program's normal operation
- What you SHOULD do:
 - Log errors,
 - Free all resources,
 - Exit the process with an appropriate error code.



ERROR HANDLING: CUSTOM ERROR TYPES

- All errors must be inherited from Error.
- Required fields:
 - name
 - message
 - stack
- You can extend custom errors with any additional fields.
[Here](#) is the list of recommended field names.

ERROR HANDLING: CUSTOM ERROR TYPES

```
1  const dictionary = { hello: "Hello", world: "World" };
2  const logger = require('winston');
3
4  class DictError extends Error {
5      constructor(word) {
6          super(word);
7          logger.warn('Missing translation for: ' + word);
8      }
9  }
```

ERROR HANDLING: CUSTOM ERROR TYPES

```
try {  
    | translateText(['%%%%', 'heellloooo']);  
} catch (e) {  
    | if (e instanceof ArgumentError) {  
    |     | console.log("Invalid input: a dictionary key may only contain Latin characters!");  
    | } else {  
    |     | console.log("Caught a missing word: " + e.message);  
    | }  
}
```

USEFUL LINKS

- Error handling in Node: <https://www.joyent.com/node-js/production/design/errors>
- REPL:
<https://www.safaribooksonline.com/library/view/learning-node-2nd/9781491943113/ch04.html>
- Debugging in VSCode: <https://code.visualstudio.com/docs/nodejs/nodejs-debugging>

A stylized world map with a green-to-blue gradient background. The map shows the outlines of continents and countries in a light blue color. The text is overlaid on the map.

NODE.JS GLOBAL

**COMMAND LINE.
DEBUGGING.
ERROR HANDLING
BY
GALINA KASATKINA**