# Node.js Cheatsheet.

## VOL - 2

# HTTP in Node.js (continued)

```
response.statusCode;
// When using implicit headers (not calling
response.writeHead() explicitly), this property
controls the status code that will be sent to the
client when the headers get flushed.
response.headersSent;
// Boolean (read-only). True if headers were sent,
false otherwise.
response.sendDate;
// When true, the Date header will be automatically
generated and sent in the response if it is not
already present in the headers. Defaults to true.

response.on('close', function () { });
// Indicates that the underlying connection was
terminated before response.end() was called or able
to flush.
response.on('finish', function() { });
// Emitted when the response has been sent.
message.httpVersion;
// In case of server request, the HTTP version sent
by the client. In the case of client response, the
HTTP version of the connected-to server.
message.headers;
// The request/response headers object.
message.trailers;
// The request/response trailers object. Only
populated after the 'end' event.
```

@vishwajeet.js

# OS in Node.js

```
// Provides a few basic operating-system related
utility functions.
// Use require('os') to access this module.

os.tmpdir();
// Returns the operating system's default directory
for temp files.
os.hostname();
// Returns the hostname of the operating system.
os.type();
// Returns the operating system name.
os.platform();
// Returns the operating system platform.
os.arch();
// Returns the operating system CPU architecture.
os.release();
// Returns the operating system release.
os.uptime();
// Returns the system uptime in seconds.
os.totalmem();
// Returns the total amount of system memory in
bytes.
os.freemem();
// Returns the amount of free system memory in bytes.
os.cpus();
// Returns an array of objects containing information
about each CPU/core installed: model, speed (in MHz),
and times (an object containing the number of
milliseconds the CPU/core spent in: user, nice, sys,
idle, and irq).
os.networkInterfaces();
// Get a list of network interfaces.
```

HTTP in Node.js

```
message.method;
// The request method as a string. Read only.
Example: 'GET', 'DELETE'.
message.url;
// Request URL string. This contains only the URL
that is present in the actual HTTP request.
message.statusCode;
// The 3-digit HTTP response status code. E.G. 404.
message.socket;
// The net.Socket object associated with the
connection.
message.setTimeout(msecs, callback);
// Calls message.connection.setTimeout(msecs,
callback).

// URL
// This module has utilities for URL resolution and
parsing. Call require('url') to use it.

url.parse(urlStr, [parseQueryString],
[slashesDenoteHost]);  // Take a URL string, and
return an object.
url.format(urlObj);
// Take a parsed URL object, and return a formatted
URL string.
url.resolve(from, to);
// Take a base URL, and a href URL, and resolve them
as a browser would for an anchor tag.
```

# Buffer in Node.js

```
// Buffer is used to dealing with binary data.
// Buffer is similar to an array of integers but
corresponds to a raw memory allocation outside the V8
heap.

Buffer.from(size);
 // Allocates a new buffer of size octets.
Buffer.from(array);
// Allocates a new buffer using an array of octets.
Buffer.from(str, [encoding]);
// Allocates a new buffer containing the given str.
encoding defaults to 'utf8'.

Buffer.isEncoding(encoding);
// Returns true if the encoding is a valid encoding
argument, or false otherwise.
Buffer.isBuffer(obj);
// Tests if obj is a Buffer
Buffer.concat(list, [totalLength]);
// Returns a buffer which is the result of
concatenating all the buffers in the list together.
Buffer.byteLength(string, [encoding]);
// Gives the actual byte length of a string.
buf.toString([encoding], [start], [end]);
// Decodes and returns a string from buffer data
encoded with encoding (defaults to 'utf8') beginning
at start (defaults to 0) and ending at end (defaults
to buffer.length).
```

@vishwajeet.js

# Assert in Node.js

```
// This module is used for writing unit tests for
your applications, you can access it with
require('assert').

assert.fail(actual, expected, message, operator);
// Throws an exception that displays the values for
actual and expected separated by the provided
operator.
assert(value, message); assert.ok(value, [message]);
 // Tests if value is truthy, it is equivalent to
assert.equal(true, !!value, message);
assert.equal(actual, expected, [message]);
 // Tests shallow, coercive equality with the equal
comparison operator ( == ).
assert.notEqual(actual, expected, [message]);
// Tests shallow, coercive non-equality with the not
equal comparison operator ( != ).
assert.deepEqual(actual, expected, [message]);
 // Tests for deep equality.
assert.notDeepEqual(actual, expected, [message]);
// Tests for any deep inequality.
assert.strictEqual(actual, expected, [message]);
 // Tests strict equality, as determined by the
strict equality operator ( === )
assert.notStrictEqual(actual, expected, [message]);
// Tests strict non-equality, as determined by the
strict not equal operator ( !== )
assert.throws(block, [error], [message]);
// Expects block to throw an error. error can be
constructor, RegExp or validation function.
```

@vishwajeet.js

```
// This module is used for writing unit tests for
your applications, you can access it with
require('assert').

assert.doesNotThrow(block, [message]);
 // Expects block not to throw an error, see
assert.throws for details.
assert.ifError(value);
 // Tests if value is not a false value, throws if it
is a true value. Useful when testing the first
argument, error in callbacks.

// Query String
// This module provides utilities for dealing with
query strings. Call require('querystring') to use it.

querystring.stringify(obj, [sep], [eq]);          //
Serialize an object to a query string. Optionally
override the default separator ('&') and assignment
('=') characters.
querystring.parse(str, [sep], [eq], [options]);  //
Deserialize a query string to an object. Optionally
override the default separator ('&') and assignment
('=') characters.
```