1) Node.js and Express

Node.js	Express
-> Runtime environment for executing javascript on the server-side.	-> Framework built on node.js that simplifies web application development by provide routing, middleware and utilities.
-> Core modules http, fs and path to build Server Side Apps.	

2) Routing Works in Express
- -> Maintain routing table intenally.
- -> Match found, execute the corresponding callback.
- -> When request comes, Express iterates through routes and middleware in order, check if url and method match.

3) Purpose of express.json() middleware
- -> express.json() pareses incoming request with JSON payloads.
- -> Req.body with parsed JSON object for easyaccess.

4) HTTP methods in REST APIs

GET ->	Retrieve resources
POST ->	Create New Resource
PUT ->	Replace an existing resource
Patch ->	Update part of resource
Delete ->	Remove a resources
options ->	Discover http method for resources.

5)

req.params	req.query	req.body
-> url parameters	-> url query String parameters	-> Data sent in the request body
-> /user/:id	-> /search?term=abc	-> Via POST/PUT request

6) Create dynamic routes:

-> Dynamic routes use parameters.
-> Fetching user details by ID.

```
app.get('/user/:id', (req, res) => {
    res.Send(`user ID is ${req.params.id}`);
});
```

) Two routs with same path & method:

-> Express executes the first route that matches the path & method.
-> Subsequent routes can act as middleware next() is called.

8) Route mounting:

-> Express allows mounting routers to specific paths.

```
Const userRouter = require('./routes/user');
    app.use('/users', userRouter);
```

-> All routes in userRouter will be prefixed with /user.

9) Middleware and request-response cycle:

-> Middleware is a function with req, res, next.

-> Modify req or res, end the response, or call next() to pass control.

10) Properties inside req object:

-> req.params	-> req.method
-> req.query	-> req.url
-> req.body	-> req.cookies
-> req.headers	-> req.ip
-> req.originalUrl	

11) Handling 404 errors:

```
app.use ((req, res, next) => {
    res.status (404).send ('Page not found');
});
```

12) res.send ()

-> Sends a response string, buffer, object

-> Automatically sets Content-Type.

res.json ()

-> Sends a json response with Content-Type: application/json.

13) Structuring routes using MVC

Model: Database Schema and logic.

View: Templates or front-end output.

Controller: Handles request logic.

Routes connect controllers to endpoints

routes/user.js -> Controllers/userController.js ->
Models/User.js

14)

Put	Patch
-> Replace entire resource	-> Updates partial resource.

15) Full request-response lifecycle:

-> client sends HTTP request
-> Node.js receives request
-> Express parses URL, method, and headers
-> Middleware runs sequentially.
-> Route matching occurs.
-> Controller logic executes.
-> Response is constructed res.send or res.json
-> Response is sent back to client.
-> Connection close.