Date: _____ M T W T F S

Node Js

- Nodejs is a Js runtime built on Chrome's V8 Js engine
- Designed to build scalable network applications.
- Free & open source server environment.
- Allow us to run js on server.
- Can run on multiple OS.

Why use NodeJs

- You can use Js in entire stack.
- Nodejs is fast.
- Comes with a lot of useful built in modules.

Data

M T W T F S

REPL

The REPL feature of Node is very useful in experimenting with Node.js codes & to debug js codes.

Read <= REPL => Loop
Evaluate Print

Read => Reads user's input, parses the input into Js data structure & stores in memory.

Eval => Takes & eval & data structure

Print => Print the result.

Loop => Loops the above command until & user enter Ctrl+C twice.

Type out>index.js ⇒ to create a new file using terminal

We can do in REPL:

1. Js expression
2. Use variables
3. Multiple codes/loops.
4. Use (_) to get last result.
5. We can use editor mode.

Core Modules

Core modules to be same as Js libraries.

Nodejs has a set of built-in modules which can use without any further installation.

File System

To get a module:

Syntax:

const var_name = require("module_name")

Example:

const fs = require("fs")

To create a new file sync.

~~fs.writeSync~~

fs.writeFileSync("read.txt", "Abc")

file_name data

If file is not available, then create a file, if file available, then overwrite the data.

To append data sync.

fs.appendFileSync("file name","data")

Node.js has one additional data type "buffer"

To read data from a file.

fs.readFileSync("file name")

It returns data in buffer type.
.toString() is used to convert into string.

To rename file sync

fs.renameSync("old file name","new file name")

Date:

To append data sync.

fs.appendFileSync("file_name","data")

Node.js has one additional data type "buffer"

To read data from a file.

fs.readFileSync("file name")

It returns data in buffer type.
.toString() is used to convert into string.

To rename file Sync

fs.renameSync("old file name","new file name")

Date _____ M T W T F S

To delete a file

```
fs.unlinkSync("file-name")
```

To read data as a string

```
fs.readFileSync("file-name","utf-8")
```

To create a folder

```
fs.mkdirSync("Folder name")
```

To delete a folder

```
fs.rmdirSync("folder path")
```

Date:

Asynchronous File System

To create a file

fs.writeFile("file path", data, callback => (err) => { ... });

It is neccessary to call a callback function in asynchronous file system

To append data:

fs.appendFile("path", "data", callback)

OS Module

To access => require("os")

To get architecture of your machine.

os.arch()

To get free memory

os.freemem()

* output is in bytes

To get total memory

os.totalmem()

To get hostname

os.hostname()

For platform

os.platform()

To get type

os.type()

os.arch()

To get free memory

os.freemem()

* output is in bytes

To get total memory

os.totalmem()

To get hostname

os.pt os.hostname()

For platform

os.platform()

To get type

os.type()

Date _____ M T W T F S

os.arch()

To get free memory

os.freemem()

* output is in bytes

To get total memory

os.totalmem()

To get hostname

os.gethostname()

For platform

os.platform()

To get type

os.type()

PATH Modules

To access => require("path")

to get dir name
path.dirname("location")

To get extension name
path.extname("location")

To get file name
path.basename("location")

To get root, dir, base, ext. & name
path.parse("location")

Date:

PATH Modules

To access => require("path")

to get dir name
path.dirname("location")

To get extension name
path.extname("location")

To get file name
path.basename("location")

To get root, dir, base, ext, & name
path.parse("location")

Date: M T W T F S

PATH Modules

To access => require("path")

to get dir name
path.dirname("location")

To get extension name
path.extname("location")

To get file name
path.basename("location")

To get root, dir, base, ext. & name
path.parse("location")

Date

[M][T][W][T][F][S][S]

Create, import & export Modules

To export a module from any file

module.exports = add

& to import require("file location")
path e.g ./ops

To export more than one module we have to export as an object.

module.exports.add = add;
module.exports.sub = sub;

import as an object and use it like ops.add.

We can also import & export using destructing.

Date:

e.g
module.exports = {add, sub, mul}
import {add, sub, mul} = require("./file name")

Import NPM Modules

To use npm module run in terminal

npm init

It is basically to beautify output in Terminal.

We have to install packages then require it like modules.

Date

M T W T F S

Import Global NPM Module

Global means when you can download any module at once then you can do use anywhere.

nodemon => Nodemon is a tool that helps develop node.js based apps by automatically starting the node app while changes in the directory is detected.

Module Wrapper Function

Date:

HTTP Module in NodeJs

Web Server:

To access web pages of any web apps, you need a web server. Web server will handle all the http requests for web apps.

Nodejs provide capabilities to create own web server which will handle http requests asynchronously. You can use IIS or Apache to run Nodejs web apps but recommended to use Nodejs web server.

To create a server we need to access http module

```
require("http")
```

Date: M T W T F S

To create a web server use a method http.createServer() which includes request & response parameters which is supplied by node.js

```
const server = http.createServer(callback)
                (req, res) => { ... }
```

* **req** => The request object can be used to ~~send a response~~ get information about current HTTP request.

res => The response object can be used to send response for current request.

Date:

in
th
ar

T

e.g

Date:

M T W T F S

in easy words, request is the data which we search and the output is responce.

To get request

server.listen('port_no', 'local-host', callback)

e.g 8000 127.0.0.1

e.g callback => () => { console.log ... }

Serving HTML Files

1st create a server

```
const http = require("http");
const fs = require("fs")
const fileContent = fs.readFileSync("file_name")
```

```
const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end(fileContent)
});

server.listen(80, '127.0.0.1', () => {
  - - - -
});
```

Creating a custom backend

We ca have to use if-else statement from diff pages.

1. Condition is on basis of req.

JSON in NodeJs

- JSON stands for JS obj Notation.
- Light weight format for storing & transporting data.
- often used when data is sent from server to web page.

To convert obj into a JSON.

JSON.stringify(obj)

- To convert JSON in an obj

JSON.parse(JSON)

- We can't access member of JSON like member of obj

API in NodeJs

API is acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook send an instant message, or check the weather on your phone, you're using API.

API is a service which allows us to request about data.

Date: M T W T F S

Events Module in NodeJs

Node.js has a built-in module, called "Events"
We can create-, fire- and listen for- your own events.

We have to create a class

=> const EventEmitter = require("events")

Now create a object

=> const event = new EventEmitter()

Then create a event (emit)

=> event.emit("event_name")

event_name can be anything

Date _____ (M)(T)(W)(T)(F)(S)

Then listen to it

=> event.on('event-name'), () => {

}

- We have to listen it before emitting.

- Event with callback parameter

event.emit("event-name, parameter)

e.g

event.emit("fun", 200, "ok")

HAFIZ

Date:

Streams & Buffer in NodeJs

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

In Node.js, there are four types of streams.

1. Readable => Stream which is used for read operation.
2. Writable => Stream which is used for write operation.
3. Duplex => Stream which is used for both read & write operation.
4. Transform => A type of Duplex stream where the output is computed based on input.

Date

M T W T F S

Each type of stream is an EventEmitter instance & throws several events at different instance of times - e.g, some of commonly used events are-

1. data ⇒ This ~~data~~ event is fired when data is available to read.
2. End ⇒ This event is fired when data is ~~no~~ no more to read.
3. Error ⇒ This event is fired when there is any error is receiving & writing data.
4. Finish ⇒ This event is fired when all the data has been flushed to underlying system.

In easy words, streaming means
to get data when it is read,
[illegible] the [illegible]
[illegible] the data [illegible] data
[illegible]

Create a readable stream

const rstream = fs.createReadStream(file)

rstream.on('data', (chunkdata) =>

Stream Pipes

stream.pipe(), the method used to take
a readable stream and connect it to
a writeable stream.

where want to write data

rstream.pipe()

e.g rstream.pipe(res)

Date

M T W T F S

Each type of stream is an EventEmitter instance & throws several events at different instance of times. e.g., some of commonly used events are.

1. data ⇒ This event is fired when data is available to read.
2. End ⇒ This event is fired when data is no more to read.
3. Error ⇒ This event is fired when there is any error is receiving & writing data.
4. Finish ⇒ This event is fired when all the data has been flushed to underlying system.

HAFIZ

Date:

In easy words, streaming means to get data when it is read. It returns the amount of data it loads the data if less data is read.

Create a readable stream

```
const rstream=fs.createReadStream("file")
rstream.on('data',(chunkdata)=>
```

Stream Pipes

stream.pipe(), the method used to take a readable stream and connect it to a writeable stream.

where we want to write data

```
rstream.pipe()
```

e.g `rstream.pipe(res)`

Date _____ (M)(T)(W)(T)(F)(S)

Express Js in Node Js

To install express Js

=> npm i express in terminal.

To Create express app.

```
Const express = require("express")
Const app = express()
```

Express.Js is a Node.js framework. It's the most popular framework as of now (the most starred on NPM).

ExpressJs is a web application framework that provides you with a Simple API To build websites, web apps and backends.

F S

Date:

M T W T F S

Try to write a small Rest API in plain NodeJs (that is, using only core modules) and then in expressjs. The latter will take you 5-10x less time & lines of code.

To get Route

Syntax:

app.get(route, callback)

We create API using ExpressJs.

There are some imp. method.

get - read
post - create
put - update
delete - delete

Date (M)(T)(W)(T)(F)(S)

Routing in Express Js

```
app.get("/", (req,res) => {
  res.send("hello World from express");
});

app.listen(8000, () => {
  console.log("listening on port 8000")
});
```

~~Routing in ExpressJs~~

Same

Send HTML & JSON Data as a Response

We can also use HTML in

```
res
app.send();

res
app.send("<h1> Hassaan </h1>")
```

Date:

If want to send multiple lines:

app.write("<h1> Hello world </h1>")
app.write("<h1> My name </h1>")

At the end use app.end()
otherwise host continues to search for more data.

To send json data.

app.send("object")

e.g

app.send({
    id: 1,
    name: "Bobby"
});

To send many files we can send many objects.

Instead of request, we can also use request().

The methods are identical when an object or array is passed, but request() will also send non-objects, such as null or undefined, which are not valid JSON.

Serve HTML, CSS & JS Files in Express.js

To serve static files such as images, CSS files, and JavaScript files, use the Express.static built-in middleware function in Express.

M T W T F S

To send more than one amobject we can send array of objects.

Instead of res.send(), we can also use res.json().

The methods are identical when an object or array is passed. But res.json() will also convert non-objects, such as null & undefined, which are not valid json.

Serve HTML, CSS & JS Files in Express Js

To serve static files such as images, CSS files & js files, use the express.static built-in middleware function in Js.

Syntax:

express.static(root, [options])

1. Open Init to create node modules.
2. Create src folder, place js files in it.
3. Create public folder, place html & css files in it.

To use express.static()

~~app.use(express)~~

~~const app = require~~

```
const path = require('path')
const singlePath = path.join(__dirname, '../public')

app.use(express.static(singlePath))
```

Another method:

res.sendFile(path, file)

Date:

Syntax:

express.static(root, [option])

1. npm init to create node modules.
2. Create src folder, place js files in it.
3. Create public folder, place html & css files in it.

To use express.static()

~~app.use(express~~
~~Const app require~~
Const path = require("path")
const singlePath = path.join(__dirname, "../public")

app.use(express.static(singlePath)

Another method:

res.sendFile(path_of_file)

Serve Static Website in Node JS (Express)

Template Engine (Pug, hbs, EJS)

A template engine enables you to use static template files in your applications. At run time, template engine replaces variables in to template files with actual values, & transform the template into HTML file sent to client.

We are using hbs as template engine.

There must be a folder named views in project folder.

Date

M T W T F S

Serve Static Website in Node JS (Express)

Template Engine (Pug, hbs, EJS)

A template engine enables you to use static template files in your applications. At run time, template engine replaces variables into template files with actual values, & transform the template into HTML file sent to client.

We are using hbs as template engine.

There must be a folder named views in project folder.

Date: [M][T][W][T][F][S]

Create a index.hbs in it.

We have to identify the engine (we are using) in expressJs.

To set view engine
app.set("view engine", "hbs")

To routing:

```
app.get("", (req, res)=>{
    res.render("index")
});
```

Customizing the view directory in ExpressJs

To change name of default views folder we have to set it in expressJs.

app.set("views", path of folder)

Using partials in ExpressJs

Partials are just like components in reactjs.

First we have to require hbs

Const hbs = require("hbs")

To use partials,

hbs.registerPartials(path of folder where partials are placed)

Date:

To reuse any partial

use {{> partial file name }}

where you want to use.

e.g {{> header }}

To run reload on any change any files of specific extension

nodemon index.js -e js,hbs

Adding 404 error page not in Dynamic Site

Date:

Intro to Mongoose

Mongoose is a used to connect nodejs(backend) to mongoDB(database)

Mongoose is an object Data Modeling(ODM) library for MONGODB & Nodejs. It manages relationship b/w data & provides schema validation & is used to translate b/w objects in code & representation of those code objects in MongoDB.

To install mongoose

npm i mongoose

Date ____ M T W T F S

To access mongoose

Const mongoose = require("mongoose")

To connect database

mongoose.connect("mongodb://localhost:27017/data_base_name")

A2 mongoose.connect("mongodb://localhost:27017/database_name", {useNewUrlParser:true, useUnifiedTopology:true})

{useNewUrlParser:true, useUnifiedTopology:true}

It returns a promise.

A.then(()=>{ }).catch(()=>{ })

~~Create & Insert Document~~

Mongoose Schema & Models Explained | Create collection

A mongoose schema defines structure of document, default values, validators etc.

To define schema

~~const abc = mongoose.~~

```
const abc=new mongoose.Schema({
    name: String,
    number: Number,
})
```

A Mongoose Model is a wrapper on the Mongoose Schema. It provides an interface to database for creating, querying, updating & deleting records etc.

Create & Insert Document

To create a document

```
const reactPlaylist = new Playlist({
    members
})
```

To add document in collection

reactPlaylist.save() => Returns a promise

To Create a collection

const ABC = new mongoose.model("Playlist", Schema)
(name of collection want to create)

e.g.

const Playlist = new mongoose.model("Playlist", abc)

Create & Insert Document

To create a document

(obj of class)
Const reactPlaylist = new Playlist({
  members ...
  ....
})

To add ~~object~~ document in collection

reactPlaylist.save() => Returns a promise

To Create a collection

using var name of collection
const Playlist = mongoose.model('Playlist', Schema)

eg:

const Playlist = new mongoose.model('Playlist', a)

Create & Insert Document

To create a document

eg:- of class
const reactPlaylist = new Playlist({
    members ...
    ...
})

document
To add ~~object~~ in collection

reactPlaylist.save() => Returns a promise

Recomended Method

By using async await method.

```
Const createDocument = async () => {
  try {
    Const reactPlaylist = new Playlist({
      ------
    })
    const result = await reactPlaylist.save()
  } catch (err) {
    console.log(err);
  }
}

createDocument
```

Date: M T W T F S

Insert Multiple Documents

To insert multiple

Playlist.insertMany(js Playlist, mongo Playlist)

Syntax:

P... .insertMany(doc1, doc2 ...)
Collection-name

Read or Querying the Documents

```
const getDocument = async () => {
    Const result = await Playlist.find();
}

getDocument()
```

Comparison Query operators using Mongoose & nodejs

Playlist.find({views: {$eq: 50}})

return document in which value of views is 50.

Playlist.find({views: {$gt: 50}})

return documents in which value of views is greater than 50.

Playlist.find({views: {$gte: 50}})

returns if any value of views equals to 50 in every document.

Date:

Comparsion Query operators using Mongoose & NodeJs

Playlist.find({videos:50})

return document in which value of videos is 50.

Playlist.find({videos:{$gt:50}})

return return document in which value of videos is greater than 50.

{ctype:{$in:[a,b]}}

returns if ctype value of ctype equals to any in array [a,b].

Date M T W T F S

Logical Query Operators

For operators.

1. $and
2. $not
3. $nor
4. $or

```
{$or : [{videos:50}, {ctype:"Database"}]}
```

operator — Condition 1 — condition 2

Sorting & Count Query Methods

.countDocuments() is used to calculate no. of documents according to given query.

.sort() is used to sort documents.

```
.sort({name : 1})
```

// sorts the document alphabetically (numbers) on names basis.

.sort({name: 1})

In ascend from a to z.

// prefer uppercase first then lowercase.

Update the Document

```
Const updateDocument = async () => {
try {const result = await Playlist.update({_id},
  { $set: {
      name: "Javascript"
    }
  }); } catch(err) { console.log(err)}
}

updateDocument(id)
```

Date

M T W T F S

To get document ~~&~~ ~~after~~ before update

.updateOne => findByIdAndUpdate

~~It gives the data~~

To data after updating, we have to add a property

new: true

Deleting Document

Similar as update

~~const deleteDocument = async (~~

```
const deleteDocument = async (id) => {
try {
    const result await. Playlist. deleteOne({_id})
} catch (err) { console.log (err) }
```

Date:

Mongoose Built-in Validation

In writing schema, we can add validation.

e.g

```
name: {
    type: String,
    lowercase: true,   -> Validations
    required: true
}
```

Date:

PYTHON:

To show anything on ~~screen~~ Console.

print()

For comments hash (#) is used.
For multi line comment enclose all lines in triple quotation marks.

(""" """)

To ingore next line

print(end="")

Data which you want to enter at the end of line.

Variables, datatypes & Typecasting

To declare a variable

var1 = 4

No need to declare datatypes.

To get Type of variable

Type(var1) #<class 'int'>

To convert string into int

int(var1)

- str()
- float()
- int()

Date: M T W T F S

Variables, datatypes & Typecasting

To declare a variable

```
var1 = 4
```

No need to declare datatypes.

To get Type of variable

```
type(var1)  # <class 'int'>
```

To convert string into int

```
int(var1)
```

- str()
- float()
- int()

HAFIZ

To print 'Hello world' 10 times

```
print(10* "HelloWorld")
```

String Functions

To get a character of string.

```
myStr: "Hassaan"
print(myStr[index])
```

To get a part of string

```
myStr[0:3]    # Has.
```

including excluding

To get length.

```
len(myStr)
```

Date:

To print "Hello world" 10 times

```
print(10*"Hello World")
```

String Functions

To get a character of string.

```
myStr = "Hassaan"   # 0 1 2 3 4 5 6
print(myStr[index])
```

To get a part of string

```
myStr[0:3]   # Has
```

including excluding

To get length

```
len(myStr)
```

Date

myStr[:3]

empty start means 0

myStr[0:]

empty end means len

myStr[0:10:2]

Skips one letter after each character.

Can also use negative numbers in slicing

myStr[::-1]

Used to reverse the string.

myStr.isalnum()

To check whether string is alphanumeric or not.

To check substring

myStr.isalnum()

To check end of string

myStr.endswith('h')

To get count of any character in string

myStr.count('a')

To capitalize the string.first character of string.

myStr.capitalize()

Date:

M T W T F S

True => Alphanumeric
False => Not alphanumeric

To check alphabet string

myStr.isalpha()

To check end of string.

myStr.endswith("str")

To get count of any character in string.

myStr.count("char")

To capitalize the ~~string~~ first character of string.

myStr.capitalize()

To find anything in string.

myStr.find("...")

To Convert string into lowercase.

myStr.lower()

To Convert string into uppercase.

myStr.upper()

To replace a part of string.

print myStr.replace("old", "new")

Lists & Lists Functions

Just like an array

List is mutable

Accessing list items

List[2] #3

Just like indexes & arrays

To access some items

list[0:3]

To sort a list

list.sort()

Date

NOTES

To reverse a list

```
list.reverse()
```

To skip item

```
list[0:4:2]
```

Don't use -ve string except -1

To get max

```
max(list)
```

To get min

```
min(list)
```

To add item at end of list

```
list.append(item)
```

To reverse a list

list.reverse()

To skip item

list[0:4:2]

Don't use -ve slicing except -1

To get max

max(list)

To get min

min(list)

To add item at end of list

list.append(item)

Date:

To insert an item at given index

list.insert(index, item)

To remove an item

list.insert(item)

To remove last item

list.pop()

To make const away we use tuple.

Diff b/w list & tuple is

List	Tuple
Can change	Can't change
[]	()

To swap two variables.

a = 1

b = 1

a, b = b, a

Dictionary & its Functions

To make a dictionary

d1 = {}

Works like an object.

e.g

d2 = {"Hany": "A", "abc": "B"}

Can also be make dictionary in dictionary

HAFIZ

To add something in dictionary

d2["Awais"] = "Junk Food"

It adds "Awais" : "Junk Food" at end.

To only delete any item key pair

del d2["Awais"]

d2 = {"Harry": "A", "abc": {"a": "x", "x": 3}}

To access sub-direct sub dictionary

print(d2["abc"]["x"])

To add something in dictionary

d2["Awais"] = "Junk Food"

It adds "Awais": "Junk Food" at end.

To any delete any item key value pair

del d2["Harry"]

Date _____ M T W T F S

d3 = d2

Means d3 is pointing to d2.

Any changes in d3, will be changed in d2.

To ignore it

d3 = d2.copy()

To get value of an item key

d2.get("Harry")

To update

d2.update({"Leena": "Potter"})

Date:

To print keys

d2.keys()

For items

d2.items()

To get input from user

input()

Sets in Python

To declare a set

s=set()

We can make a set from list

s_from_list=set([1,2,3,4])

To add element in set

s.add(e)

For union

s1.union(s2)

For intersection.

s1.intersection(s2)

To remove element

s.remove(element=>e)

If Else Elif Conditional

```
num1 = 6
num2 = 3

if num1 == num2:
    print(num1)
else:
    print(num2)
```

To place if else if in one line we use elif in between

To check that is any element available in list or not

```
list = [1,2,3]

if 3 in list:
    print(5)
```

As in keyword, we also have not in keyword.

For Loops

```
list1 = ["Harry", "Larry", "Cary", "Marie"]

for item in list1:
    print(item)
```

It prints the whole list.

For list of list 1

```
for i, item1, item2 in list1:
```

It works with list & tuple.

For dictionary

for item1,item2 in dictionary.items():

To get key of dictionary

for item in dictionary:

To check numeric or not

item.isnumeric()

While Loops

while(i>=5):

Date:

For dictionary

for item1,toitem2 in dictionary.items().

To get key of dictionary

for item in dictionary:

To check numeric or not

item.isnumeric()

While Loops

while(i>45).

Date _____ M T W T F S

Break & Continue

break statement used to break ~~to continue~~ the loop.

continue statement ignores the further lines in loop & again iterate for next item.

Operators

Two get power (**) is used.

Eg

```
5**3 = 125
```

and is used for and operation

Or is used for OR operation

Is , is not

Membership operator: in, not in

Functions & Docstrings

To define user-define function

def fun_name():

tab

. . . .

Try Except Exception Handling

try:

. . . .

except Exception as e:

print(e)

it print error as a string.

Scope, Global Variables & Global Keywords

1st function search for variable in local scope then move to the global scope.

Built-in Modules

Consider a module to a same as a code library.

A file containing a set of functions you want to include in your app.

e.g of python modules.

1. Math
2. Random
3. OS
4. time

Scope, Global Variables & Global Keywords

1st function search for variable in local scope then move to the global scope.

Built-in Modules

Consider a module to a same as a code library.

A file containing a set of functions you want to include in your app.

e.g of python modules.

1. Math
2. Random
3. OS
4. time

M T W T F S

Scope, Global Variables & Global Keywords

1st function search for variable in local scope then move to the global scope.

Built-in Modules

Consider a module to a same as a code library.

A file containing a set of functions you want to include in your app.

e.g of python modules.

1. Math
2. Random
3. OS
4. time

To use any module.

import math

To access any function write

math. → press tab ~~then~~ to ~~use~~ have list of functions.