Node is
is usually for a website (run in the browser)

Node is let javascript run on a server, so you can build apps like APIs, chat
pp, backend system.

hat make Node is special:

Asynchronous & Event driven → can handle many requests at the same time
without waiting for one to finish before starti
another.

Scalable → can handle 1000 of users without slowing down.

stallation:

go to offical node js site download file and install. Check with,
de -version. Cmd is that successfully installed.

ode REPL → Node Read Eval Print Loop
is a computed env where the user input are read and
aluated and then the results are retured to the user.

Hate the Node REPL type node in terminal.

a js runtime so we can write anything that javascript.

nnsng.js file using Node

ex.js

onsole.log('hello') cd path\to\folder using Node.

node index.js -> execute the file and log h
terminal.

tive node modules.

(there is lot of modules there to check doc) here is few.

ri ti file.

.js

```
const fs = require("fs");
fs.writeFile ('message.txt', 'hello', (err) => {
    if (err) throw err;
    console.log('saved');
});
```

require used to be load the module.
here fs is the module.

→ This method called common

To read files,

```
fs.readFile ('message.txt', 'utf8', (err, data) => {
    if (err) throw err;
    console.log (data);
});
```

using E6 modules method for this 2 is,

```
import {writeFile} from 'node:fs';
writeFile('message.txt', data, (err) => {
    if (err) throw err;
    console.log('Saved');
});
```

```
import {readFile} from 'node:
readFile('message.txt', 'utf8',
        (err, data) => {
    if (err) throw err;
    console.log(data);
});
```

ES modules (import) → if working with newer projects & binary data.
Common js (require) → for compatibity with all Node.js versions.

ode package manager (npm)
its basically a place which collects modules that people have built
r node, and its created by github org

tialize NPM → npm init

npm i superheroes → after installation creates a package.json file
package-lock.json and node_modules

se ES module need to add 'type': 'module' in package.json file
pe": "commonjs" for use common js require.

```
import superheros from 'superheros";
const name = superheroes.random();
console.log(`I am ${name}`)
```

node index.js → randomly log superhero nam

- QR code scanner project to practice.

used package (inquirer.js , qr-image),

↓ get input from user

↓ generate ~~pno~~ image in pn eps and pdf formats

Express Frame work.

→ A framwerk on top of the Node

ul! node js server.

```
mport http from 'http';
mport url from 'url';

onst server = http. createServer( (req,res)=> {
  const parsedUrl = url.parse(req.url, true);
if (parsedUrl.pathname === "/") {
  res.writeHead (200, {'Content-Type': 'text/plain'});
  res.end ("hello this node server')
}
else {
    res.writeHead (404, {'Content-Type' :'text-plain'});
    res.end ("Not found"); } });

server.listen (3000, () => {
    console.log ( running on http://localhost:3000);
        });
```

node js with Express js

```
import express from 'express'
const app =express();
app.use(express.Json());
app.get ['/', (req, res) => {
   res.send['This is Express
                          Server')
app.listen (3000, () => {
    console.log ( run. . . . . )
});
```

pros :-

* cleaner code
* built in routing
* easy middle ware support

Cons :-

Require installian Express

npm install express

ting an express Server

reate directory

reate index.js file.

tialise NPM.

nstall Express package

te Server Application in index.js

rt server

step installian to skip

npm init → to step by step ingtallian to skip

npm init -y ⟵ ⊗ → It will create a default main : index.js

npm install express → installing express

if want we can change after

index.js

import express from 'express'; → module import

const app = express() → using the module object

const port = 3000

app.listen (port, () => { → using our express app.

console.log (`port : ${port}`).

})

to run , node index.js

(it will start the port, ~~but when we~~

~~check~~ in need to ~~run~~ in brower

we need to add routes, but port

is now open to check using

netstat -ano | findstr "LISTENING" ⟵ the following cmd for windows.

http request (hypertext transfer protocal.

Request vocab.

1. ~~GET~~ 1. GET → getting something from server
2. post 2. POST → Senting something to the server
3. put { 3. PUT → Replace resourse
   4. PATCH → patch up the resource
   5. DELETE → removing resource.

date

example, you receiving the bi
from amazon but the one wheel
broken,
need to fix the wheel → patch
need a new one → put.

```
port- express from 'express';

st app = express()
st port = 3000
.get('/', (req, res) => {
   res.send('hello'); })
isten(port, () => {
```

Simple start of our express se

```
app.get('/', (req, res) => {
    console.log(req.rawHeaders);
    res.send('hello')})
```

→ it will print in terminal all the header values, like, host, server, user agent, which platform this request, ... etc

until now if we made any changes, we need to stop and restart

wer, so we can use nodemon → to automate restart when change o
in file.

npm i -g nodemon → g → just for globally install, if no need

ow instead of running the, like.

node index.js to use → nodemon index.js

---

res.send("<h1> hello </h1>"); → to sent html format

res.sendStatus(404); → we can sent status code like this

postman → making request.

clientside → desktop, mobile, laptop, ... etc. → http request.
Server side → Application, Database Server.

ttp Response Status Code.

Informational response → (100 - 199)
Successfull " → (200 - 299)
. Redirection message → (300 - 399)
. Client error message → (400 - 499).
Server error response → (500 - 599)

we download same project that have package.json with the
pedencies we dont need to install every package manually
, npm install.

middleware Express.

runs before sending a response, helping to process request, modify data or handling error.

Commonly used middleware node; express -> Body parser

in Express js its helps read data sent in a request (like JSON or form data) and make it available in req.body

File structure

node_modules

public
↳ index.html

index.js

To sent html file.

```
app.get("/", (req, res) => {
    res.sendFile('full/path/.../index.html');
});
```

↳ but its hard coded not dynamic make it dynamically do following steps.

```
import express fram 'express';
import {dirname} fram 'path';
import {fileURLToPath} from 'url';

const __dirname = dirname(fileURLToPath(import.meta.url));
```

this way can make filepath writing dynamically

we can use custom name but its a common way.

```
...
in route, ..
bnpres.sendFile(__dirname + "/public/index.html");
```

Body parser in html forms.

index.html.

```
<form action='submit' method='POST'>
    <input type='text' name='street'>
    <input type='text' name='pet'>
    <input type="submit" value='submit'>
</form>
```

y added in app.js

```
mport bodyParser from 'body-parser';

pp.use (bodyParser.urlencoded ({extended : true}));

pp.post('/submit', (req, res) => {
    console.log(req.body);
    res.send('submitted')
```

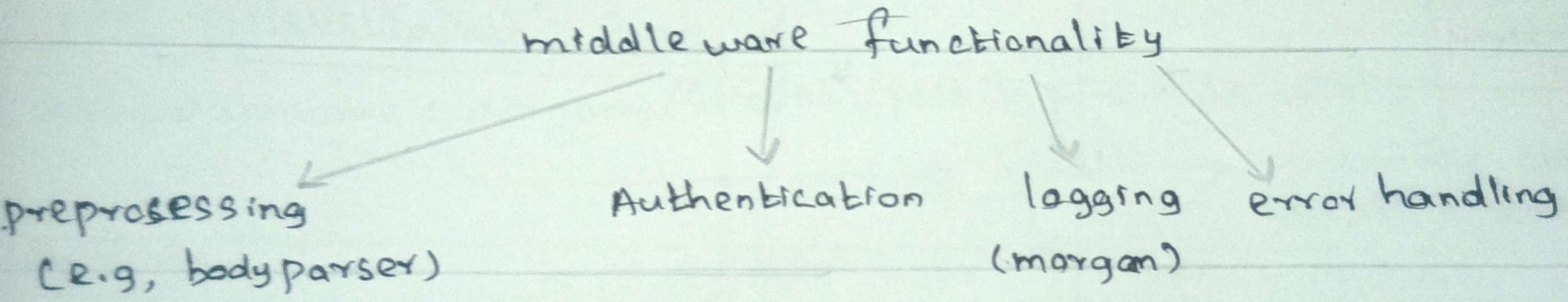
} to use bodyparser to app.

in terminal,

{ street : 'south', pet : 'dog' }

here we using body parser so we get data in req.body.

Custom middle ware.



morgan -> logger.

```
import express from 'express';
import morgan from 'morgan';

const app = express()
const port = 3000
app.use(morgan("combined")); // this will log the request details in
```

(or) tiny (or) dev ... etc.

custom logger implementation, middleware.

```
import express from 'express';
const app = express()
const port = 3000
function logger (req, res, next) {
  console.log(`log: method ${req.method}`)
  console.log(`log: requrl ${req.url}`)
  next()
})
app.use(logger);
app.get('/', (req, res) => {
    res.send('hello'); });
```

custom middleware function

The next() functions is critical in express middleware, it tells express to move next middleware or handler.

* without next() the request get stuck inside the middleware.
* The route handler never execute because the request doesnt move forward.

Exercise -> handname generator.
Secret access project.

Numpy.

1. creating and manipulating arrays.

np.array(), np.zeros(), np.ones(), np.arrange(), np.linespace()

np.array.

```
import numpy as np
arr = np.array([1, 2, 3])
print(arr)
```

→ [1, 2, 3]

→ Basic Sytax.

```
arr = np.array([1, 2, 3], dtype=str)
print(arr)
```

→ ['1', '2', '3']

Specifying data type.

[0, 1, 2], dtype = bool → [False, True, True] → 0 as false others tru

Array properties.

```
arr = np.array([1, 2, 3, 4, 5])
print(arr.ndim)
print(arr.shape)
print(arr.size)
print(arr.dtype)
```

→ 1 d in vector.

→ 1 → Number of dimensions

→ (5,) → Shape (raw, column)

→ 5 → total elements

→ int64 → Data type

2d array. (matrix).

```
arr2 = np.array([ [1,2,3,], [4,5,6,] ])
print(arr2.ndim)        → 2
    (arr2.shape)        → (2,3)
    (arr2.size)         → 6
    (arr2.dtype)        → int64.
```

```
arr3 = np.array([
    [[1,2,3], [3,4,5]],
    [[5,6,7], [7,8,9]]
    ]).
```

ndim → 3
shape → (2,2,3)
size → 12
dtype → int.

form shape,

```
3 = np.array([[
    [[1,2,3], [3,4,5]],
    [[5,6,7], [7,8]]
    ])
```

→ Throw an error because Numpy arrays must have a uniform shape in all dimensions.

np.array() → creates numpy array from list or tuples.
Supports multidimensional array (1D, 2D, 3D, ...)
Allow custom data types
used for efficient mathematical operation in python.

np.zeros()
* is a numpy function to create and array filled with zeros.
* useful for initialising array before storing values.

1d array zero values,

arr = np.zeros(5) → 3 → [0 0 0.0 0]
print(arr) → [[[0.0.0] [0.0.0] [0.0.0]]
arr[0] = 1
print(arr) → [1 0 0 0 0]
→ [[0.0.0] [0.0.0]]

2d array

```
arr2 = np.zeros((2,3), dtype=int)
print(arr2).
```
→ [[0, 0, 0], [0, 0, 0]]

(3 — 2)

```
print(ndim).
```
→ 2

```
arr3 = np.zeros((2,3,3), dtype=int)
print(arr3)
print(arr.ndim).
```
→ [[[0,0,0],[0,0,0],[0,0,0]],
[[0,0,0],[0,0,0],[0,0,0]]

→ 3.

) np.ones() → same like zeros() this one value filled with one.

```
arr = np.ones(3)
rint(arr).
```
→ [1., 1., 1.]

np.arrange()

its create a array with evenly spaced within a given range.

```
arr = np.arrange (5)
print (arr)
```
→ [0, 1, 2, 3, 4]

```
arr = np.arrange (1, 5)
print(arr)
```
(1 → start, 5 → end)
→ [1, 2, 3, 4]

```
arr = np.arrange (1, 10, 2)
print(arr)
```
(1 → start, 10 → end, 2 → step)
→ [1, 3, 5, 7, 9]

5. np.linespace()

create an array of evenly spaced numbers between a start and sto

```
arr = np.linespace(1, 5, 1)
print(arr);
```
→ (start, stop, num)
→ [1.] created that
almost of spited array.

```
arr = np.linespace(1, 5, 3)
print(arr)
```
→ [1., 3., 5.]

use case: for creating graphs, charts, or numerical simulation where need fixed num of points between 2 values.

1. start (req)
2. stop (req)
3. num (opt)
4. endpoint (opt)
5. retstep (opt
6. dtype (opt)

Reshaping

i) .reshape()
ii) .ravel()
iii) .flatten()

reshape() → changes the shape of the array without changing its data.
useful where data need to recognize into diff structure.

```
arr = np.arr([1,2,3,4,5,6])
arr = arr.reshape(3,2)
print(arr)
```

→ [[1,2],[3,4,],[5,6]]
6 element reshaped into
3 rows × 2 columns.

if (2,3) → [[1,2,3],[4,5,6]]

to 3d.

```
arr=np.array([1,2,3,4,5,6])
arr = arr.reshape(2,3,1)
```

→ [[[1],[2],[3]],
[[4],[5],],[6]]

use -1 to auto calculate, dimensions,

```
arr = np.array([1, 2, 3, 4, 5, 6])
arr = arr.reshape(-1, 2)
print(arr)
```

→ auto calculate rows.

→ [[1, 2], [3, 4], [5, 6]]

2. flatten() → always returnld array.

```
arr = np.array([[1, 2], [3, 4]]).
arr = arr.flatten()
print(arr)
```

→ [1, 2, 3, 4].

3. ravel() → Similar to flatten() but provide a view (if possible) means changes affect the orginal array.

```
arr = np.array([[1, 2], [3, 4]]).
arr = arr.ravel()
print(arr)
```

→ [1, 2, 3, 4].

copy vs view , .copy() , .view()

.copy() → independed, duplicate and keep the original unchanged.

.view() → lightweight reference to the original. Need to quick alter way access elem
without extra memory usage.

```
arr = np.array([1,2,3])
copy_arr = arr.copy()
copy_arr = 100
print(arr)
print(copy_arr)
```

→ [~~100~~, [1,2,3,] ← original remains unchanged.

→ [100,2,3] ← copy is modified

```
arr = np.array([1,2,3])
view_arr = arr.view()
view_arr[0] = 100
print(arr)
print(view_arr)
```

→ [100,2,3] } both modified.

→ [100,2,3] }

Indexing and slicing.

1. indexing 1d array.

arr = np.~~numpy~~ array([10, 20, 30]).

print(arr[0]) → 10

print(arr[-1]) → 30

2. Indexing 2d arrays. → use~~d~~ arr(row, column) to access elements.

arr = np.array([[1,2,3], [4,5,6], [7,8,9]])

print(arr[0,1] → 2 (first row 2nd column).

print(arr[2,-1]) → 9 (last row last column).

for better performance instead this use like arr[0][1] and arr[2][-1]

3. Slicing 1d array.

arr = np.array([10,20,30,40,50,60,70])

print(arr[1:5) → [20,30,40,50] ← index 1 to 4

print(arr[:4]) → [10,20,30,40] ← first 4

print(arr[::2]) → [10, 30, 50, 70] ← every second element

print(arr[::-1]) → [70, 60, 50, 40, 30, 20, 10] ← reverse array.

. slicing 2d array.

rr2 = np.array([[1,2,3], [4,5,6], [7,8,9]])

int(arr2[0, :]) → [1, 2, 3] ← first row

nt(arr2[:, 1]) → [2, 5, 8] ← Second column

nt(arr2[1:, :2]) → [[4,5], [7, 8]] ← last 2 rows first 2 column.

int(arr2[::2, ::2]) → [[1, 3], [7, 9]] ← every second row and column

Badean indexing (filtering) → filter based on condition.

arr = np.array([10, 20, 30, 40, 50])

print(arr[arr > 25]) → [30, 40, 50] ← element > 25

print(arr[arr % 20 == 0]) → [20, 40] ← multiplies of 20.

6. Fancy Indexing.

arr = np.array([10, 20, 30, 40, 50])

print(arr[[0, 2, 4]]) → [10, 30, 40] ← select indices 0, 2 and 4.

For 2d arrays -

arr2 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(arr2[0, 2], [1, 2]]) → [2, 9] ← picks: (0, 1) & (2, 2)

Basic arithmetic operations.

arr1 = np.array([1, 2, 3, 4, 5])

arr2 = np.array([10, 20, 30, 40, 50])

print(arr1 + arr2) → [11, 22, 33, 44, 55] ← Adding

print(arr1 - arr2) → [-9, -18, -27, -36, -45] ← Subtraction

print(arr1 * arr2) → [10, 40, 90, 160, 250] ← Multiply

print(arr1 / arr2) → [0.1, 0.1, 0.1, 0.1, 0.1] ← Division

print(arr1 ** 2) → [1, 4, 9, 16, 25] ← power

print(arr1 % 2) → [1, 0, 1, 0, 1] ← Modulo.

```
arr = np.array([1, 2, 3, 4, 5])
print(np.sum(arr))    → 15  ← Sum of elements.
print(np.mean(arr))   → 3.0 ← Mean (average).
print(np.std(arr))    → 1.41421... ← Standard deviation (measure of speed
print(np.var(arr))    → 2.0        ← Square of Stanards.

print(np.min(arr)  →  1   ← min
print(np.max(arr)  →  5   ← max.
```

Operations on 2d adkray.

```
arr2D = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(np.sum(arr2D)  → 45  ← Sum of all
print(np.sum(arr2D, axis=0)) → [12, 15, 18] ← Column wise Sum
print(np.sum(arr2D, axis=1)) → [6, 15, 24] ← Raw wise Sum.
```

Cumulative operations.

```
arr = np.array([1, 2, 3, 4])
print(np.cumsum(arr)) → [1, 3, 6, 10]   ← cumlative sum
print(np.cumprod(arr)) → [1, 2, 6, 24]  ← cumlative product.
```

Array Joining and spliting.

np.concatenate() → general purpose concatenation along any axis
np.vstack() → vertical stacking (row-wise)
np.hstack() → Horizontal stacking (column-wise).

concatenate.

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
nt(np.concatenate((arr1, arr2)))  → [1, 2, 3, 4, 5, 6]   default axis
```

2d arrays.

```
arr1 = np.array([[1, 2], [3, 4]])
```

print(np.concatenate((arr1, arr2), axis=1))

→ [[1, 2], [3, 4], [5, 6], [7, 8]]

→ [[1, 2, 5, 6], [3, 4, 7, 8]]

(shape of the array must match along axis they are concatenated)

stack → Row wise. (behave like concatenate() with axis = 0).

stack → column wise (" " " () with axis = 1).

split an array,

split() → array to be split equally.

, array_split() → avoid error for uneven split.

plit()

rr = np.array([1, 2, 3, 4, 5, 6])

plit_arrays = np.split(arr, 3) → group value.

r sub_arr in split_arrays :
print(sub_arr)

→ [1, 2],
[3, 4],
[5, 6]
} evenly divided into

do like,

r = np.array([1, 2, 3, 4, 5])

: split(arr, 3) → Error because we can't spit 5 element as 3 e

allow unequal splits uses ,array_split()

= np.array([1, 2, 3, 4, 5])

_arrays = np.array_split(arr, 3)

for sub_arr in split_arrays :

→ [1, 2],
[3, 4],
} still works.

Sorting → sort()

Summary →

p.sort(arr) → sort arr in asending

p.sort(arr, axis=0) → sort along columns

.sort(arr, axis=1) → sort along row (default).

.sort(arr)[::-1] → sort in decending order.

.sort(arr, order='column name') → sort structured array by a sp
column

rr = np.array([5, 2, 3])

rr = np.sort(arr)

rint(arr) → [2, 3, 5]

rr = np.array([[8, 2, 5], [1, 9, 3]])

rr = np.sort(arr)

rint(arr) → [[2, 5, 8], [1, 3, 9]] each row sorted individua

orting along Specific axis.

axis = 0 → column (vertical)

= 1 → row (horz. ..) default.

= np.array([[8,2,5], [1,9,3]])

r = np.sort(arr, axis=0)

int(arr)

→ [[1,2,3], [8,9,5]]

ending order.

r = np.array([5, 2, 3])

r = np.sort(arr)[::-1]

t(arr)

→ [5,3,2].

ing in structured array.

np.array([(1, 'Apple', 50), (2, 'orange', 30), (3, 'mango', 40)

dtype = [('id', int), ('name', 'U10'), ('price', int)

= np.sort(arr, order='price') → sort by price.

print(arr) → [(2, 'orange', 30), (3, 'mango', 40), (1, 'Apple', 50)]

earching in numpy.

```
rr = np.array([10, 25, 17, 30, 40])
licies = np.where(arr > 20)
nt(indicies)
```

→ [1, 3, 4] ← where arr > 20.

place values.

```
r = np.array([10, 25, 17, 30, 40])
w_arr = np.where(arr > 20, "High", "Low")
nt(new_arr)
```

→ ['Low', 'High', 'Low', 'High', 'Low

↓ Replaced values > 20 with 'High' else 'Low'

gmax(). → find the index of max value.

rgmin → Find the indexc of min value.

umpy random.

andom.rand()

~~arr = np.array(~~

rr = np.random.rand(3,2) → [[0.342, 0.654],
[0.123, 0.987],
[0.756, 0.453]]

int(arr)

3×2 array with random float be
0 to 1.

andom.randint()

and_int = np.random.randint(10,20) → 15 random num between 10

int(rand_int)

andom.choice()

rr = np.arr([10, 20, 30, 40, 50])

n = np.random.choice(arr) → 30. (picked a random).

int(n)

n1 = np.random.choice(arr, 3) → [50, 10, 30] (multiple ra
choic

int(n1)

Choose without replacement.

```
n = np.random.choice(arr, 3, replace=False)   → No repeating values.
print(n)                                      → [20, 50, 10]  (ensures no duplicate)
```

Matrix manipulation.

np.dot() → for dot product of vectors.

np.matmul() → for strict matrix manipulation (better for higer

```
= np.array([1,2,3])
= np.array([4,5,6])
sult = np.dot(a, b)   → 1*4 + 2*5 + 3*6. => 32.
int(result)           → 32
```

Creating & loading df.

```
import pandas as pd
data = {'Name': ['dhin', 'velam',],
        'Age' : [25, 50],
        'City' : ['tn', 'tn'] }
df=pd.DataFrame(data)
print(df)
```

→ simple df creation.

loading.

```
pd.read_csv('data.csv')
.read_excel('data.xlsx', sheet_name='sheet1')
.read_json('data.json')
```

Viewing.

df.head() → First 5 row, df.head(10) → first 10 row,

df.tail() → Last 5 row, df.tail(10) → Last 10 row,

df.info() → Summary of DF structure.

df.describe() → Summary statics of numerical columns

Indexing & Selecting data.

print (df['Agge']) → 0 25
1 50
} Single col selection.

print (df[['Name', 'Agge']]) →
	Name	Agge
0	dhin	25
1	velam	50

} multiple col selection.

Selecting raw & columns.

using df.loc[] → label based selection. (df.loc[row_label, column_labe

print (df.loc[0]) →
Name	dhin
Age	25
City	tn

print (df.loc[0, 'Name']) → dhin ← Select a specific value

print (df.iloc[0:1, ['Name', 'Age']]) →
	Name	Agge
0	dhin	25
1	velam	50

} multiple col

using df.iloc[] → position based selection.
(df.iloc[row_index, column_index]

print (df.iloc[0]) → Name Ahin
Age 25
City tn
Name: 0, dtype: object.

print(df.iloc[1,1]) → 50 (select value at (row 1, colum 2)

print(df.iloc[0:2, 1:3]) → Age City
0 25 tn
1 50 tn
} multiple row and column.
Row 0-1, columns 1-2

key diff..
df.loc[] → label based → df.loc[1, 'City'] → 'tn'
df.iloc[] → position based → df.iloc[1, 2] → 'tn'

Boolean filtering filter row based on condition.
high_age = df[df['Age'] > 30]
print(high_age('Age') → 50

df.dropna() → Removing missing values.

print(df) →

	Name	Age	Salary
0	Alice	25	50000
1	Bob	NaN	60000
2	Charlie	35	NaN
3	NaN	40	75000
4	Eve	29	80000

df = df.dropna()
print(df) →

	Name	Age	Salary
0	Alice	25	50000
1	Eve	29	80000

} removed -NaN containi

df.fillna(value) → Replace missing values,

df = df.fillna(0)
print(df) → print the full df and replaced NaN

Advanced fill options,

df['Age'].fillna(df['Age'].mean(), inplace=True)
print(df) → print the full df with skipped Age con

df.ffill(inplace=True) → skip all the Nan and print the

df.bfill(inplace=True) → same like this

ffill → from forward

bfill → from back

hanging Data types.

f['Age'] = df['Age'].astype(int) → change it to int

f['Salary'] = df['Salary'].astype(float) → change it to floa

) other example Syntax.

- int → df['col'].astype(int)
- float → df['col'].astype(float)
- string → df['col'].astype(str)
- boolean → df['col'].astype(bool)

Removing duplicates.

```
df = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie', 'Alice'],
    'Age': [25, 30, 35, 25],
    'Salary': [50000, 60000, 70000, 50000]});
```

	Name	Age
0	Alice	25
1	Bob	30
2		35

options for, drop_duplicates()

subset = ['col1', 'col2'] → Consider Specific columns only

keep = 'first' → keep first occurence (default)

keep = 'last' → keep last occurence

Keep = False → removing all duplicates

Renaming column.

df = df.rename (columns = {'Name' : 'Employee Name', 'Salary' : 'Month

print (df.head(1)) →

	Employee Name	Age	Monthly Salary
0	Alice	25	50000

) change the colum

options,

columns = {'old' : 'new'} → Rename specific column.

inplace = True → Apply changes directly.

ata transformation.

Sorting → df.sort_values(by='column')

f = df.sorted_values(by='Age')

rint(df) → print the df sorted by age

or decending order,

f = df.sort-values(by='Age', accending=False)

f.sort_values(by=['col1', 'col2'], asscnding = [True, False])

→ Sort by multiple columns different orders.

df.sort_values(by='column', na_position='first') → Move NaN values to th

pplying functions.

df.apply() → working with multiple columns (or) applying function ro

df.map() → when modifying single column.(series) element-wise.

df = pd.DataFrame({'Name': ['Alice', 'Bob', 'Charlie'],
'Salary': [5000, 10000, 20000]})

df['updated_salary] = df['Salary'].apply(lamda x : x * 2)
int(df).

→ we created the new col with 2x salary row wise.

plying functions to each row(axis=1)

calculate_tax(row):
return row['Salary']*0.10 if row['Salary'] > 50000 else row['Salary']*0.05

'tax'] = df.apply(calculate_tax, axis=1)

nt(df)

apply row wise.
added the column tax in row
check salary > 50k → tax salary
else - salary * 0.05.

df.apply() → for complex row wise (or)
column wise operations.

f['After_Bonus] = df['Salary'].map(lamda x : x +5000)
rint(df)

→ Created new column with 5000 added Salary.

ic replacing values:
df['Category] = df['Name'].map({'Alice': 'intern, 'Bob': 'employee, 'Charlie': Ma
Print(df) →

Name	Salary	Category
Alice	5000	Intern
bob	10000	Employee

rouping → .groupby('column').mean()

groups data by column compute the mean for each loop.

can a

: p.d.DataFrame ({'Department': ['HR', 'IT', 'HR', 'Finance', 'IT', 'Finance'
'Salary': [50000, 70000, 52000, 65000, 75000, 62000]}

f = df.groupby('Department').mean()

rint(df)

→

Department	Salary
Finance	63500.000
HR	51000.000
IT	72500.000

dr.

→ here, groupby('Department')
data and .mean() calculates th

'Gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],
'Salary': [50000, 70000, 52000, 65000, 75000, 62000]})

n(df)

Department	Gender	Salary
Finance	Female	62000.0
	Male	65000.0
HR	Female	52000.0
	Male	50000.0
IT	Female	70000.0
	Male	75000.0

. groupby (['Department', 'Gen
groups data by both dep an
and . mean() computes the
Salary for each group.

Reset index for a cleaner dataframe.

By default, groupby() → returns multiindex df. if want to reset to
df.

df.reset_index(inplace = True)

print(df) → This will flatten the index and make Department
gender regular columns.

ther ~~Aggressions~~, Aggregations,

re can replace .mean() → with other functions (

- sum()
- count()
- median()
- min() / max()

} Example:

df.groupby('Department').agg({'Salary': ['mean', 'sum'

» This calculate mean, total, sum and count Salaries

ing Dataframes.

pd.concat() , ii) pd.merge().

↓ stack df, vertically (rows) (or) horizontally columns.

↘ joining df based on common colum (keys) (like SQL Joins).

pd.DataFrame({'ID': [1, 2], 'Name': ['Alice', 'Bob']})
pd.Dataframe({'ID': [3, 4], 'Name': ['charlie', 'David']}),

= pd.concat([df1, df2], ignore_index=True).

→ ignore index =True re

	ID	Name
0	1	Alice
1	2	Bob
2	3	Charlie
3	4	David.