RS

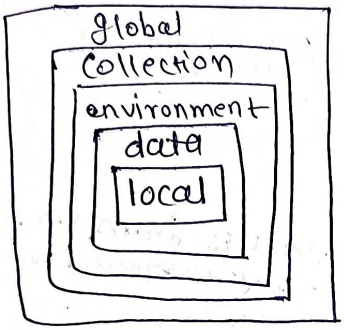
1. Intro to postman tool for testing API's in Collection
2. variables & Environments
3. Postman scripting Editor types and Response Assertion.
4. Data Driven testing & dynamic workflow
5. Oauth 2.0
6. End to End eg. of parsing complex nested JSON
7. Mock server
8. Soap Webservices Automation through postman for xml
9. Newman & importance of Newman for end to end Automation execution
10. Run postman Automation tests through Jenkins with HTML
11. Team Collaboration and version control features in postman
12. End to End Ecommerce API
13. Javascript
14. API testing Basics

Rahul Shetty

Variable scopes ⇒

postman supports the following variable scopes :-

heavily used
* ① Global
* ② Collection
* ③ Environment
④ Data → for data driven testing
⑤ Local → specific to request



pre-request script & tests are editor

Scripting in postman ⇒

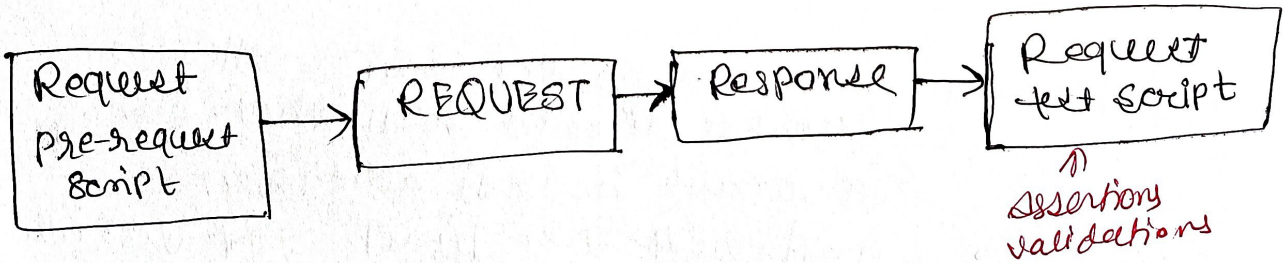
it is allowed to add dynamic behaviours to requests & collections.
This allows you to write test suites, build requests that can contain dynamic parameters, pass data between requests.

Execution order of scripts ⇒

In postman, the script execution order for a single request looks like this :

① A pre-request script associated with a request will execute before the request is sent.

② A test script associated with a request will execute after the request is sent.



The pm object =)

You will carry out most of the postman JS API functionality using "pm" which provides access to request & response data & variables.

Postman uses chai assertion

```
const jsonData = pm.response.json();
pm.test ("validate status code is 200", function()
{
    pm.response.to.have.status(200);
    pm.expect (jsonData).have.property ("Msg");
    pm.expect (jsonData.Msg).to.eql ("successfully added");
})
```

How to create random ID.

prerequest (dynamically generate values from variables into input payload request.

```
console.log (pm.globals.get ("Company Code"));
const code = pm.globals.get ("company code");   ← post
const val = pm.variables.replaceIn('{{$randomInt}}');
pm.collectionVariables.set("isbn, code + val);
pm.collectionVariable.set ("author-name", "Rohini");   ← dynamic variable
```

use => JS reference in postman official website

Assertions in postman

tests =>

```
const jsonData = pm.response.json();
pm.test ("Validate Status Code is 200", function()
{
    pm.response.to.have.Status(200);
    pm.expect (jsonData).have.Property ("msg");
    pm.expect (jsonData.Msg).to.eql ("successfully added");
});
```

post => prerequest => author name set किया collection variable में add किया then {{ }} से variable body में add किया और get response किया उसे validate किया get request में

```
pm.test("Headers cookies responsetime validations", function()
{

    pm.response.to.have.header("content-Type");
    pm.expect(pm.response.headers.get("content-Type")).
        to.eql("application/json; charset=UTF-8");
pm.expect(pm.response.responseTime).to.be.below(1200);
pm.expect(pm.response.text()).to.include("successfully added");
```

giving whole response in string format

Prerequest script में dynamic variable create करके body में get use कर सकते हैं।

Validating json response with the i/p data of other API through Automated code.

get ⇒

```
pm.test("validate the json response logic function()
{
    console.log(getBookResponse);
    pm.expect(getBookResponse[0].author).to.eql(pm.
    collectionVariable.get("author_name"));
```

// validate author_name from response with set collection variable & pass this variable as a body in get request to validate

```
const getBookResponse = pm.response.json();
```

Body ⇒ {
"isbn": "{{isbn}}",
"aisle": "252857",
"author": {{author_name}}.
}

POST

Post ⇒ add book ⇒ by adding book
with body दिये

Validate response of GET request
with post request

Validate Json Schema → in structured format

↳ Search for JSON schema generator
↳ add your expected schema.
↳ copy schema from internet then validate

get request में जो response मिला है, उसको validate
करना है

const getBookResponse = pm.response.json();
const schema = {extracts from internet}

Test ⇒ pm.test("validate the Json response schema", function ()
{
pm.response.to.have.jsonSchema(schema);
pm.response.to.have.status(200);
pm.response.to.have.jsonSchema(schema);

Postman

- Variables
  dynamically generate value from variables into I/P payload requests.
- Response Assertion
  - Assertions
  - running set of tests together in collection
  - validating Json response with the I/P data of the API through Automated code.
  - JSON schema & its validation

Data Driven Testing
- Functional logic validations with postman scripting Assertions
- Data driven testing
- Handle error handling scenarios in API tests
- Prepare workflow of execution smartly with postman scripting

This tool comes with powerful js. editor (postman object) which help us to perform various assertions inside the tool to validate your tests.

Postman offers various ready-made features of framework like variables, environments, workflows, data driven components from object which helps to quickly setup Automation lab for testing.

postman supports various types of services like Rest APIs, Soap Web Services, GraphQL testing etc.

this tool comes with powerful integration of Newman tool to run the automated tests from CLI which can also be integrated with Jenkins for CI/CD integration.

Excel + Html → can generate Report.

Data driven testing

functional logic validation with postman scripting

Assertions :

post ⇒

How to retrieve Body from request.

```
pm.test("ID logic validation", function()
{
const isbn_val = pm.collectionVariables.get("isbn");
var req = JSON.parse(pm.request.body.raw);
reqraiste # access request object ↑ for validation
Const aisle_val = req.aisle
Const expected_id = isbn_val + aisle_val;
pm.expect(expected_Id).to.eql(bookId);
```

Data driven testing from csv to postman requests scripts:

can import csv data but Can't excel data

odd 100 data / books

eg. Add project

POST ⇒ Pre-request

define variable in collection

```
PM.collectionVariables.set("book_name", pm.iterationData.
  get("BookName"));
```

↑ col name from csv file

```
console.log(pm.collectionVariables.get("book_name"));
PM.collectionVariables.set("author_name", pm.iterationData.
get("Author_name")
console.log(pm.collectionVariables.get("Author"));
```

while reln the file we need to give that file.

Handle Error Handling Scenarios in API tests with postman:

Clean up script

use try catch

How to prepare workflow of execution smartly with postman scripting ⇒

post ⇒ Tests ⇒

```
try
{
  pm.response.to.have.status(200);
  pm.expect(jsonData).have.property("msg");
  Pm.expect(jsonData.msg).to.eql("successfully added");
}
catch(err)
{
  if(jsonData.msg.includes("exists"))
  { cleanupScript();
```

(25) **OAuth 2.0** Testing OAuth 2.0 API's using Postman and Rest Assured (Automation)

⇒ Authenticate response
⇒ Standard protocol is desired to Secure APIs called as OAuth 2.0

→ OAuth 2.0 is the industry-standard protocol for authorization.
→ OAuth 2.0 comes with multiple Grant types
→ Authorization code & Client Credentials are the most commonly used Grant types for OAuth.
you can login ~~with~~ through gmail / fb so how it will integrate it with what info will be given by gmail.
if u are going to login with gmail, it will give you
client, ClientID, Client Secret ID, Resource owner, Resource/Authorization Server
Access token
→ Understand the flow of OAuth (Authorization code) Grant type with real world ex.

Understand Grant Type Authorization flow with real time example:-
(1) given authentication to the 3rd party user
Because → (1) No Data breach headaches for application
[all info will be provided by 3rd party
(2) need not maintain any user profile data

Client → client whose need to be serve
eg. BookMyShow → need to register in gmail.
→ Backend implementation of Authorization code with different layers of security.

process → ① user sign into google by hitting google Authorization server & get code
② application will use this code to hit the google resource server in back end to get (Access token, firstname, last name, email)
③ Application grant access to user by validating access token.

Access token जो मिलेगा browser में save होगा when you sign in navigate back to page (ticket page)
every page access of bookmyticket app will be getting by using "Access token".

* flow procedure in achieving OAuth 2.0 Authentication mechanism

choose Authenticatⁿ → OAuth 2.0
request parameter will be → url google का है
① scope (query parameters) → name, email id of client
② Auth url → third party user who will help to authenticate
③ cliend id → eg. bookmyticket (Get at the time of registratⁿ)
④ Response type → code
⑤ redirect url → after login redirect to bookmyticket
⑥ state → verifyfrjsdss (use security testing purpose)
↳ present in applicatⁿ server

① login through/authenticate through gmail
gmail में register करो ⇒ तो client id generate होगा जो google की API में दो फिर कुछ code generate होगा validate होगा उसके बाद तो Access token will generate for bookmyshow then we can access all APIs of it.

mandatory fields for getAuthorization code Request;
this operatⁿ perform on browser
end point → Authorization server URL
query params → scope, Auth-url, client_id, response_type, redirect_url
O/P → code

mandatory fields for generatetoken request →
end point → Access token url
query params → code, client-id, client secret, redirect_uri, grant_type
O/P → Access token

End to end API test

① Collection → variable → save with variable name then set value in that variable from individual request.

eg. pm. const jsonData = pm.response.json();

pm.collectionVariables.set("productId", jsonData.productId);

pm.expect(jsonData.message).to.eql("product added")

Writing tests ⇒

↳ Tests confirms that your API is working as expected, that integrations between services are functioning reliably, and that any changes haven't broken existing functionality.

↳ you might write a test to validate your API's error handling by sending a request with incomplete data or wrong parameters.

↳ test scripts can use dynamic variables, carry out test assertions on response data and pass data between requests.
In test tabs for a request, enter your JS manually or Select snippets next to the code editor.

↳ Post execute after the response is received.

Validating Responses ⇒

To validate the data

Mock server

A mock API server initiates a real API server by providing realistic mock API responses to requests.

Within postman, you can make requests that return mock data defined if you do not have a production API ready. Or you do not want to run your requests against real data yet.

By adding a mock server to your collection & adding examples to your requests, you can simulate the behaviour of a real API.

⇒ If server side code is still not ready yet server side of code is also in the development. So, this side of code cannot be integrated with the client side code so in that case need to create mock server.

So, mock server would have been generated by this client side code

mock server generates the same kind of response and itself backs to the client side, you don't have to depend on server sid code when you hit the request it goes to the mock server, it generates the same kind of response & itself backs to the client side

adv. of using mock server you do not have to depend on the server side code. You can host it locally

Mock server → Add → add method, request path, Response code, Response body → Next → Name to mock server → create → save that url

if you want to modify can edit as well mock server. it very imp when your deadline are very near.

Pre-request Script ⇒

You can use pre-request scripts in Postman to execute JS before a request runs.

By including code in the pre-request script tab for a request, collection or folder, you can carry out pre-processing such as setting variables values, parameters, headers & body data.

→ debugging script can be written under either the pre-request script tab or tests tab. with helpful messages logged in the postman console.

Pre-request script include ⇒

You can also use pre-request scripts for debugging code, for example by logging o/p to the console

↳ This allows you to define commonly used pre-processing or debugging steps you need to execute for multiple requests.

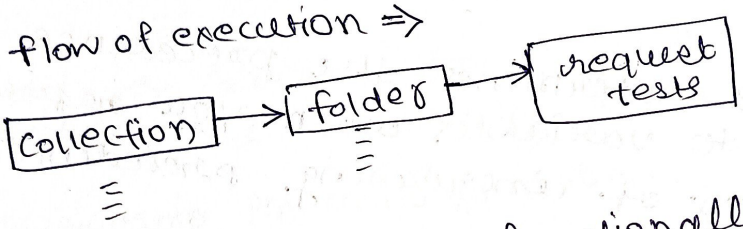
Building Request workflows

`pm.response` ⇒ to validate the data returned by a request.

`pm.tests` ⇒ define tests [returns true or false]

`pm.expects` ⇒ gives your test result messages a different format.

Flow of execution ⇒



pm ⇒ pm objects provides functionality for testing your request and response data, with the postman object providing additional workflow control.

`pm.*` → provides access to request & response data, and variables

You can access and manipulate variables at each scope in postman using the (pm) API.

Dynamic Variables ⇒

postman uses faker library to generate sample data, including random names, addresses, email addresses, & much more. you can use these pre-defined variables multiple times to return diff. values per request.

you can use these variables like any other variable in postman. their values are generated at the time of execution & their names start with $ symbol, dynamic variables whose values are randomly generated during request/collection run.

To use dynamic variables in prerequest or test scripts, you need to use

pm.variables.replaceIn('{{$randomFirstName}}')

pm → object provides methods for accessing global, collection and environment variables. Specifically, ↓ pm.variables methods for accessing variables at diff. scopes & setting local variables.

Variable scope determines the precedence postman gives to variables when you reference them, in order of increasing precedence:-

Global
Collection
Environment
Data
Local

The variable with the closest scope overrides any others.

pm.environment ⇒ methods to access & manipulate variables in the active environment

pm.environment.has(variableName:string): function
check whether the environment has a variable with the specified name.

pm.environment.get ⇒ get the variable with the specified name in the active environ.

pm.environment.set ⇒ Set the variable with the specified name & value in the active environment.

pm.environment.replaceIn ({{ firstname }});
Return the resolved value of a dynamic variable inside a script using syntax {{$ variableName }};

pm.environment.toObject(): # Return all variables with their values in the active environment in a single object

* Remove a ~~variable~~ from a active environment, Specifying the variable by name

~~pm.variable~~ pm.environment.unset()

* Clear all variables in the active environment:

pm.environment.clear()

Note that your ability to edit variables depends on your access level in the workspace

pm.collectionVariables ⇒ methods to access & manipulate variables in the collection.

pm.globals ⇒ methods to access and manipulate variables at global scope within the workspace.

Running collections

① Using the collection Runner ⇒

The collection Runner enables you to run the API requests in a collection in a specified sequence.
It logs your request test results & can use scripts to pass data bet'n requests & alter the request workflow.

⇒ You can alter the flow of execution from your request scripts using setNextRequest to create workflows.

⇒ if any tests in a request script fail during the collect'n run, the whole request fails

Sharing collection runs ⇒ You can share collection run results with others by exporting the results from the collection Runner.

② scheduling runs with monitors ⇒

You can automate collection runs using monitors to schedule runs & receive reports on your request test results.

Triggering runs with webhook ⇒

A webhook provides a way to automatically send data from one application to another. using a webhook, you can trigger a collection run in postman at a specific time or when an event occur.

The request body sent to the webhook must use JSON format only.

Building request Workflows ⇒

In collection runner, you have the option to change the order of the requests before starting a run.

You can automate this behaviour using the `postman.setNextRequest()` function.

enables you to specify which request postman runs next, following the current request

⇒ add this on test tab

⇒ postman runs the specified request after completing the current request.

Stopping workflow ⇒ `postman.setNextRequest(null);`

The collection run will stop after postman completes the current request.

→ setNextRequest() only works in collection Runner

→ use setNextRequest() in Pre request or test scripts

If more than one value is assigned, the last value that is set takes precedence.

→ Specify the next request using the request ID

→ setNextRequest() scope is limited to the collection.

→ SetNextRequest() always executes last

Importing data files

You can use data files to pass postman sets of values to use in a collection run.

By selecting a JSON or CSV data file in the collection Runner, you can test your requests with multiple different values as part of a single run.

Accessing data file values ⇒

To use values from the data file in your tests or pre-request script code, use the iterationData, which provides access to the current data file record being used to run the request.

`pm.iterationData.get("value")`

Scratch pad ⇒

The scratch pad is a space where you can work if you are not connected to postman servers.
when you are not logged in or don't have a n/w connection, you can still access postman features such as creating collections & requests or sending requests.
All your work in the scratch pad is stored locally.

HTTP Status Codes

1XX Informational

100 - Continue
101 - Switching protocol
102 - processing

2XX Success

* 200 - OK
* 201 - Created
202 - Accepted
203 - Non-Authoritative info
* 204 - No content
205 - Reset Content
206 - partial Content
207 - multi status
208 - Already Reported
226 - IM used

4XX - Client Error

* 400 - Bad Request
* 401 - Unauthorized
402 - payment Required
* 403 - forbidden
* 404 - Not found
405 - Method Not Allowed
406 - Not Acceptable
407 - proxy Authentication Required
408 - Request timeout
409 - Conflict
410 - Gone
411 - Length Required
412 - Precondition failed
413 - Request Entity Too large

3XX - Redirection

300 - Multiple choices
301 - moved permanently
302 - found
303 - See other
* 304 - Not Modified
305 - Use Proxy
306 - Unused
307 - Temporary redirect
308 - permanant Redirect

500XX - Server Error

* 500 - Internal Server Error
501 - Not Implemented
502 - Bad Gateway
503 - Service Unavailable
504 - Gateway Timeout
505 - HTTP version Not Supported.

Automate response (grab the access token into Actual request)

① const ~~token~~ jsonData = pm.response.json() # Response से token लिया (parse the response)

tests → const token = ~~json~~ jsonData.access_token

jsonData से / response से token extract किया और उस variable में save किया

then उस value को variable में save कर लिया (global var में)

Environments ⇒ global ⇒ define variable

tests ⇒ pm.globals.set("token", token);

Preparing Automation Scripting to grab the Access token into Actual request ⇒

⇒ give access token in Authorization select OAUTH 2.0

```
const jsonData = pm.response.json();
const access_token = jsonData.access_token;
pm.globals.set("access_token", access_token);
```

Variable can do replacement from body, request, test, even from Authorization i/p also

can bring these global variables.

⑧ How to parse complex Nested Json response through Json object

→ verify if there is an entry called cypress
→ verify if cypress entry has course title & price properties / keys
→ verify if sum of API courses equals to 90
→ verify if web automation course titles are shown as expected titles.

If there is {} means, it is nested object.

Tests ⇒ Verify if there is an entry called cypress

```
const jsonData = pm.response.json();
pm.test("Course Verification", function () {
const cypressObject = jsonData.courses.webAutomation.filter(m => m.courseTitle
    == 'Cypress');  # if this condn matches then that
                    Variable will be retrieved
pm.expect(cypressObject).to.be.an("object", "Cypress not found");
```

it should be object (verify structure of object)

Verify if cypress entry has course title and price properties # How to parse complex Nested json through Json object

```
pm.test("Properties verification", function(){
const cypressObject = jsonData.courses.webAutomation
.find(m => m.courseTitle == 'Cypress');
pm.expect(cypressObject).to.have.all.keys('courseTitle',
'price');
```

Verify sum of API courses equals to 90

Solving functional validations by parsing nested Json through JS functions.

```
var i;
pm.test("Validate sum of API Courses", function() {
let sum = 0
for (let i=0; i<jsonData.courses.api.length; i++)
{
    var K = parseInt(jsonData.courses.api[i].price);
    sum = sum + K
}
console.log(sum);
pm.expect(sum).to.be.eq(90);
});
```

Verify if Web Automation course titles are Shown as expected ~~lines~~ titles.

```
const jsonData = pm.response.json(); //parse json
const expectedCourses = ["Selenium Webdriver Java", "Cypress",
                         "protractor"];

Const actualCourses = [ ];

pm.test("Validate web course titles", function()
{
    ~~actualCourses~~ jsonData.courses.webAutomation.map(m =>
actualCourses.push(m.courseTitle));
                                  ↑ extracting courseTitle
    console.log(actualCourses);
  pm.expect(expectedCourses).to.be.eql(actualCourses);
});
```

iterating

⇒ Can do verification on the basis of schema

JSON Response & generate schema from given json ⇒

Json schema we will get from dev

JSON ⟷ Json schema (online).

How to automate the json schema validation through Postman

```
Const schema = { json schema};

pm.test('validate schema', () =>
{
  pm.response.to.have.jsonSchema(schema)
});
```

liquid-technologies.com ⇒ can get Json Schema copy from response then convert

Javascript

```
console.log("Hello world");  # display on console
```

keywords (variables for ES6)

```
var  a=6;
let  a=6;
const a=6;
```

```
Console.log(typeof(a)) # to see type of data type
```

Null, undefined, number, string, boolean

JS fundamentals

1. JS Hello world program
2. Declaring the variables in JS
3. Understanding the datatypes in JS
4. Loops & condn
5. logical operators & A
6. Arrays & its methods with detailed eg.
7. functn in JS
8. Understanding of var, let, const keyword
9. String & its related functn
10. JS object
11. classes & its properties, methods in JS

1. download VS.
2. new file with basic.js

```
console.log("Hello World");
```

execute cmd ⇒ node basic.js
↳ node is engine to help to execute JS related files

diff. betn var, let, const

typeof() # what kind of data it is.

We cannot redeclare variable with let keyword but possible with var

! not apply on boolean.

reassign is allowed with let.

Const ⇒ cannot reassign

Loops⇒ const flag = true;
if () { else }

Console.log(!required)

const ⇒ you cannot reassigned

Conditions⇒

Loops⇒

```
for(let k=0; k<10; i++){
{
  Console.log(k)
}
```

let required = true

Arrays ⇒ collection of variable

let marks = Array(6);

var a = new Array(20, 30, 35, 23) # object is created

```
var marks = [20, 33, 45, 12]
  Consoler.log (marks[2])
   marks[3] = 14
  Consol.log (marks)
  console.log (marks.length);
 marks.push(65); # append data in array
 marks.pop(); # delete last ele of array
 marks.unshift(12) # add element at beginning of array (first ele
 marks.
```

Array & its methods (KW)

```
console.log(marks.includes(120))  # verify 120 is in array

marks.slice(2,5) # return sub array

var sum = 0
for(let i=0 ; i<marks.length; i++)
{
  console.log(marks[i])
  sum = sum + marks[i]
}

console.log(sum)  # min sum of ele in array
```

Reduce, filter, map

```
let total = marks.reduce((sum, mark) => sum+mark, 0)
                          ↑ array          ↓ func^n logic
console.log(total)
```

Reduce is anymeous funct^n

```
var scores = [12,13,14,15,16] ; var evenscores = []
for(let i=0 ; i<scores.length ; i++)
{ if(scores[i] % 2 == 0) {
# console.log(score[i], "is a even no.")
    evenscores.push(scores[i])
}

Console.log(evenscores)
scores.filter(score => score % 2 == 0)
```

↓ it will iterate & store value of array in each iterate

```
# create new array with even no. of scores & multiply each value with 3
let newfilterEvenScores = scores.filter(score => score % 2 == 0)
                                                  ↑ array
                                  ↓ condn
console.log(newfilterEvenScores)
```

Map ⇒ for every score it will do some operat^n and give new ele.

```
let mappedarray = newfilterEvenScores.map(score => score * 3)  # mapped each node from one value to other
console.log(mappedarray) #
```

sorting optimized logic using JS

Sorting → ① String ② Int

① String

```
let fruits = ["mango", "bamana", "orange"]
fruits.sort(); # gives ascending order
console.log(fruits) #
fruits.reverse() # display in descending order
```

② Int

```
var scores1 = [12,03, 14, 19, 16]
console.log(scores1.sort()) # (12,14,16,19,3)   (wrong o/p)

scores1.sort(function (a,b){
        return a-b
})
console.log(scores1.sort((a,b) => a-b)) # bubble sort logic
                                  b-a)) # give reverse order
```

Functions in JS ⇒

block of code can execute together by wrapping them, a module called functions.

#block of code

Anonymous funcn ⇒ funcn without name

```
let sumofIntegers = function(c,d)
{
  return c+d
}
```

Reduce code ↓

```
let SumofNumbers = (c,d) => c+d
#SumofNumbers(2,3)
Console.log(SumofNumbers(2,3))
```

CWH

```
var number1 = 25
var number2= 12;
Console.log(number1+number2);
```

data types in JS

#numbers

```
var num1 = 455;
var num2 = 56.76;
```

String

```
var str1 = "this is a string";
var str2 = "this is also string";
```

object

```
var marks = {
        ravi: 34,
        raghav: 78.88,
        harry: 99.977
    }
console.log(marks);
```

booleans

```
var a = true;
var b = false;
console.log(a, b);
```

```
var und = undefined
Console.log(und) # o/p undefined
var und;
Console.log(und) # o/p → undefined
var n = null;
```

Arrays ⇒ collect of elements (string, boolean .. etc)

JS ⇒ 2 types of data type → At a very high level, these 2 types of datatypes

① premitive ⇒

② Reference.

① premitive data type ⇒
undefined, null, number, string, boolean, symbol

② Reference data types ⇒ Arrays and object

Arrays ⇒ var arr = [1, 2, 3, 4, 5]
(0) (1) (2) (3) (4)

```
console.log(arr[0]) // 1

var arr = [1, 2, "Rohini", 3]

console.log(arr);
```

operators ⇒

i) arithmetic operators ⇒ +, -, *, /

ii) assignment operators ⇒ =, +=, *=, /=, -=, ==

== ⇒ comparision operator returns true / false

iii) logical operators ⇒ operate on boolean
&&, ||, &! → logical Not

alt + click ⇒ multiple cursor

functions ⇒

var ⇒ scope of var is global if declared in global
—— " —— function —— " —— function

can redeclare
can reinitialize

const & let variable
scope ⇒ to that function only

let ⇒ can't redeclare
can reinitialized

const ⇒ can't reinitialized

String ⇒ let day = "tuesday" # collection of characters

```
let sun = day.slice(0, 4);
    console.log(sun)
console.log(day[1])
    # tue day
let splitday = day.split("s");
    console.log(splitDay[1].trim())

let date = '23'
let nextdate = '27'
let diff = parseInt(nextDate) - parseInt(date) # string to num
    console.log(diff)
    diff.toString()
# concatenate 2 String
let newQuote = day + "is funday"
    Console.log(newQuote)
    newQuote.indexOf("day",5)
    console.log(val)
```