Adapt Ready Assignment

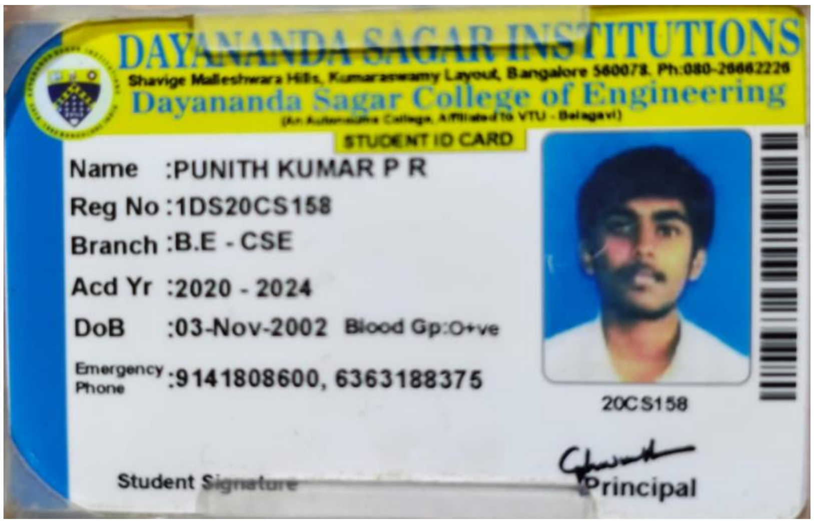
Date: 02/08/2024

Name: Punith Kumar P R
Email: punithkumarpr03@gmail.com
USN: 1DS20CS158
College: Dayananda Sagar College of Engineering

Software development

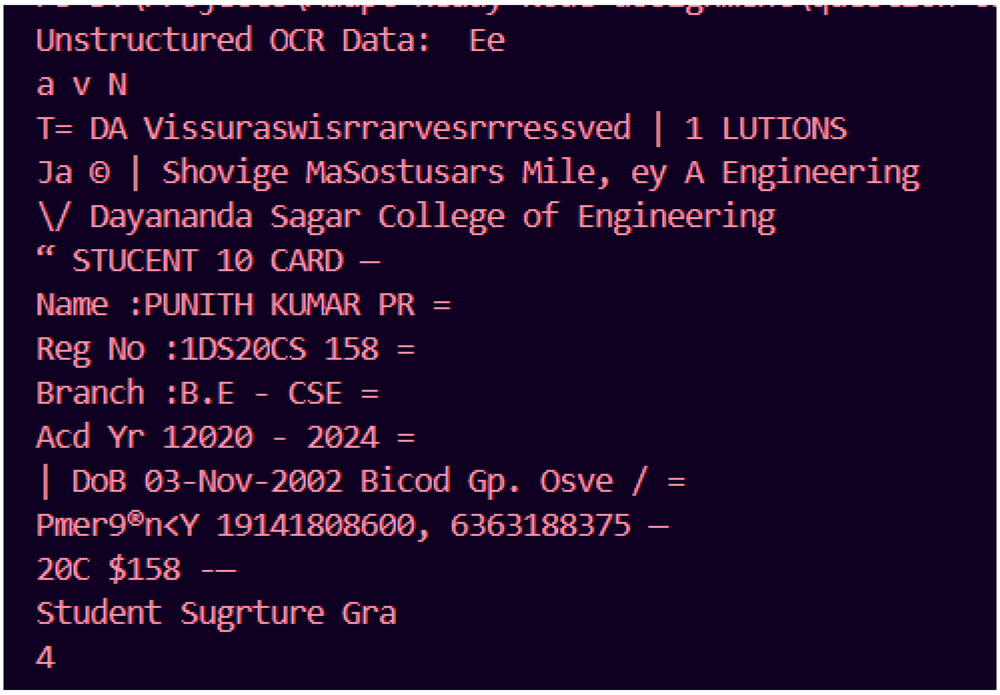
1) How we can parse the data from an unstructured data to structured table format using regular expression algorithms in python/Nodejs? with an example which has to be explaining about the workflow ex: Use the OCR reader for parse the raw text (Unstructured data) from an image (some id card) and then make the regular expression algorithm for creating a set of structured data.

- I have used **my college id card** for this demonstration.
- Used **Tesseract OCR** engine for Optical Character Recognition. Used tesseract.js module. (I also have a project "Lipi" using OCR in node.js. [link]
- ID card Photo



DAYANANDA SAGAR INSTITUTIONS
Shavige Malleshwara Hills, Kumaraswamy Layout, Bangalore 560078, Ph:080-26662226
Dayananda Sagar College of Engineering
(An Autonomous College, Affiliated to VTU - Belagavi)
STUDENT ID CARD
Name :PUNITH KUMAR P R
Reg No :1DS20CS158
Branch :B.E - CSE
Acd Yr :2020 - 2024
DoB :03-Nov-2002 Blood Gp:O+ve
Emergency Phone :9141808600, 6363188375
20CS158
Student Signature
Principal

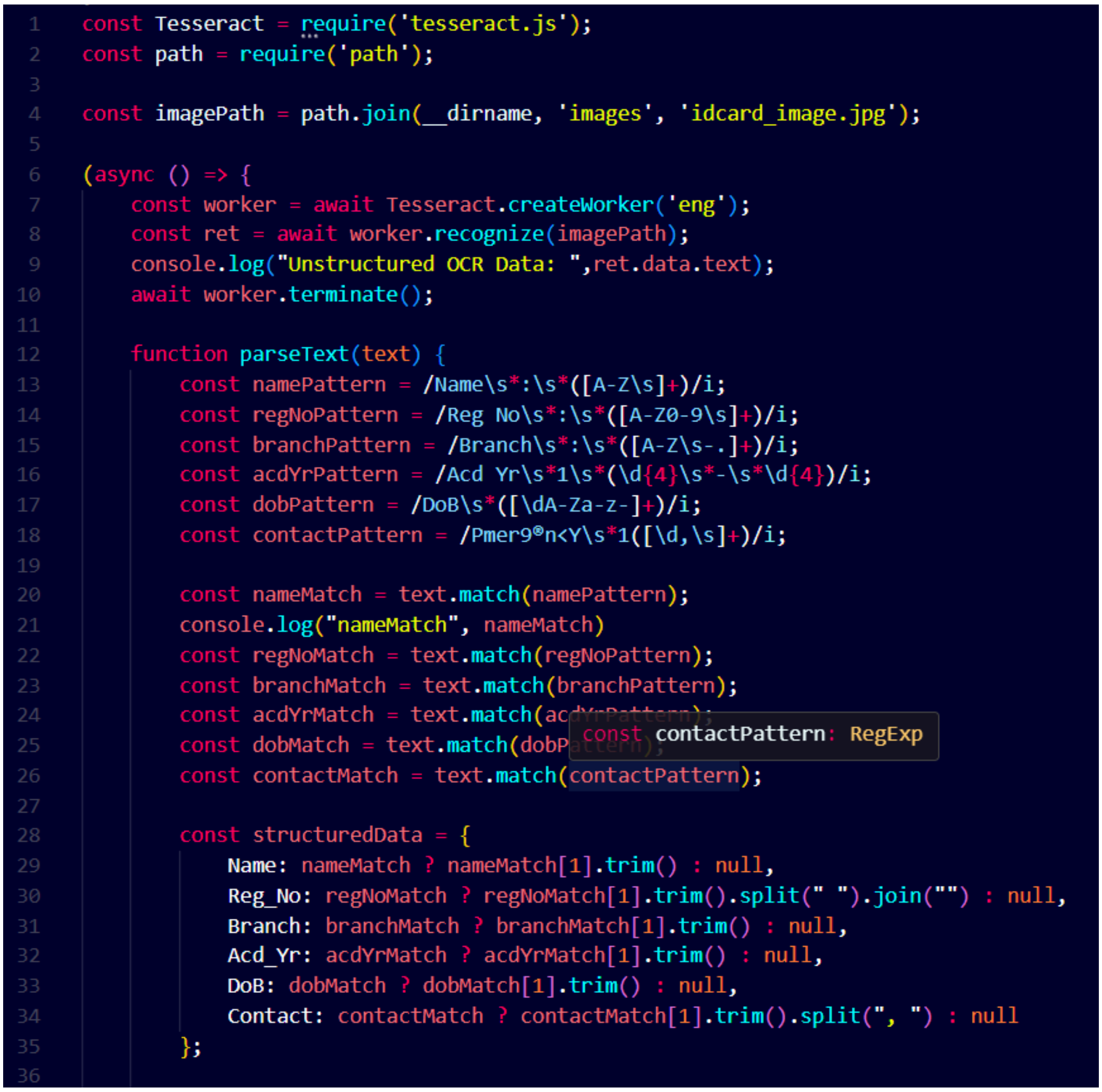
-
- Unstructured data or OCR data

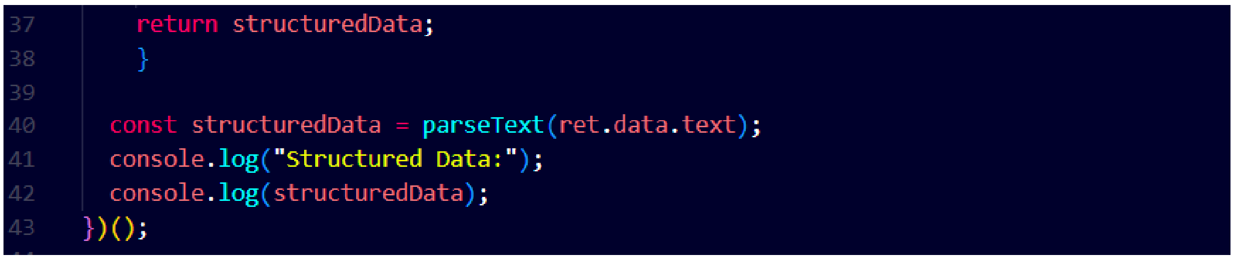


```
Unstructured OCR Data:  Ee
a v N
T= DA Vissuraswisrrarvesrrressved | 1 LUTIONS
Ja © | Shovige MaSostusars Mile, ey A Engineering
\/ Dayananda Sagar College of Engineering
" STUCENT 10 CARD —
Name :PUNITH KUMAR PR =
Reg No :1DS20CS 158 =
Branch :B.E - CSE =
Acd Yr 12020 - 2024 =
| DoB 03-Nov-2002 Bicod Gp. Osve / =
Pmer9®n<Y 19141808600, 6363188375 —
20C $158 ——
Student Sugrture Gra
4
```

- Now parsed the data Using Regular Expression.
  Note: I parsed the data based on the output I obtained. Here ':' is recognised as '1'. Written regular expression accordingly.

- **Code and explanation:** (missed blood group here, updateded code is available in github [link])

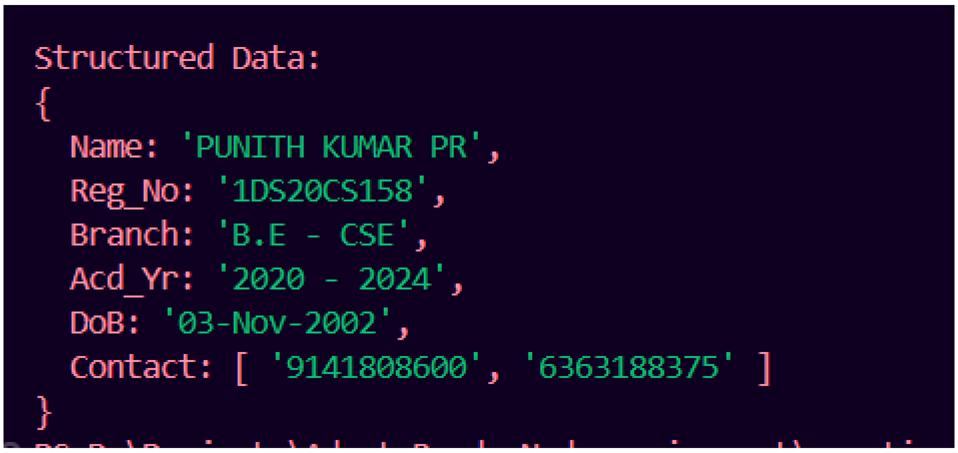




```
1   const Tesseract = require('tesseract.js');
2   const path = require('path');
3
4   const imagePath = path.join(__dirname, 'images', 'idcard_image.jpg');
5
6   (async () => {
7       const worker = await Tesseract.createWorker('eng');
8       const ret = await worker.recognize(imagePath);
9       console.log("Unstructured OCR Data: ",ret.data.text);
10      await worker.terminate();
11
12      function parseText(text) {
13          const namePattern = /Name\s*:\s*([A-Z\s]+)/i;
14          const regNoPattern = /Reg No\s*:\s*([A-Z0-9\s]+)/i;
15          const branchPattern = /Branch\s*:\s*([A-Z\s-.]+)/i;
16          const acdYrPattern = /Acd Yr\s*1\s*(\d{4}\s*-\s*\d{4})/i;
17          const dobPattern = /DoB\s*([\dA-Za-z-]+)/i;
18          const contactPattern = /Pmer9®n<Y\s*1([\d,\s]+)/i;
19
20          const nameMatch = text.match(namePattern);
21          console.log("nameMatch", nameMatch)
22          const regNoMatch = text.match(regNoPattern);
23          const branchMatch = text.match(branchPattern);
24          const acdYrMatch = text.match(acdYrPattern);
25          const dobMatch = text.match(dobPattern);
26          const contactMatch = text.match(contactPattern);
27
28          const structuredData = {
29              Name: nameMatch ? nameMatch[1].trim() : null,
30              Reg_No: regNoMatch ? regNoMatch[1].trim().split(" ").join("") : null,
31              Branch: branchMatch ? branchMatch[1].trim() : null,
32              Acd_Yr: acdYrMatch ? acdYrMatch[1].trim() : null,
33              DoB: dobMatch ? dobMatch[1].trim() : null,
34              Contact: contactMatch ? contactMatch[1].trim().split(", ") : null
35          };
36
37          return structuredData;
38      }
39
40      const structuredData = parseText(ret.data.text);
41      console.log("Structured Data:");
42      console.log(structuredData);
43  })();
```

- Line 9: *ret.data.text* gives the extracted data from image.
- Line 40: getting the structured data by passing the *ret.data.text* to function *parseText()*
- In *parseText* function, there are regular expression pattern for Name, reg no, etc.
- Using string *match()* we use *text.match(regularExpression)*. The *match()* method matches a string against a regular expression. The *match()* method returns an array with the matches.
- Using the returned values, constructed and returned the *structuredData* object.

- **Obtained output:**



```
Structured Data:
{
  Name: 'PUNITH KUMAR PR',
  Reg_No: '1DS20CS158',
  Branch: 'B.E - CSE',
  Acd_Yr: '2020 - 2024',
  DoB: '03-Nov-2002',
  Contact: [ '9141808600', '6363188375' ]
}
```

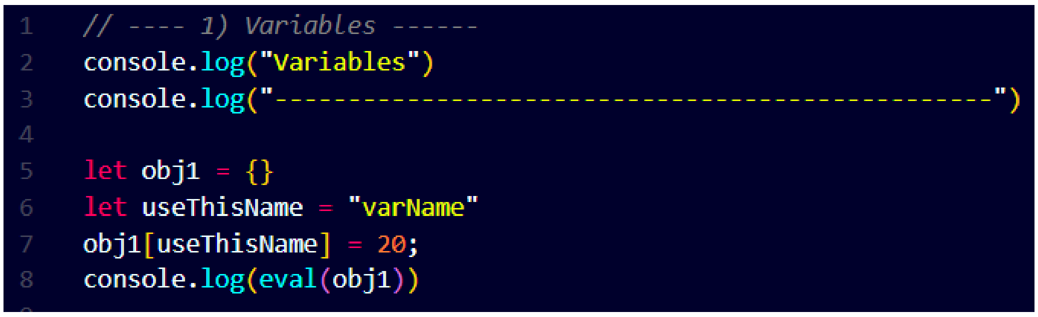
3) Dynamic variable declaration and execution in different forms (variables, multi-dimensional arrays) with examples. And how we can perform an operation/action using eval with Nodejs/python.

Dynamic Variable Decleration and execution.

- Dynamic variable names don't have a specific name hard-coded in the script. They are named dynamically with string values from other sources.
- Dynamic variable declaration and execution involve creating variables, arrays, or other data structures at runtime rather than at compile time.
- Can use *eval(), map data structure, using modern JS object property syntax*.

1) <u>Variables</u>

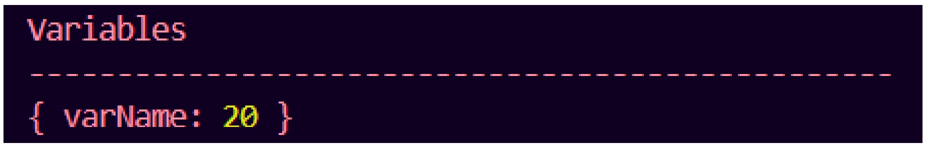
Code:



```
// ---- 1) Variables ------
console.log("Variables")
console.log("-------------------------------------------------")

let obj1 = {}
let useThisName = "varName"
obj1[useThisName] = 20;
console.log(eval(obj1))
```

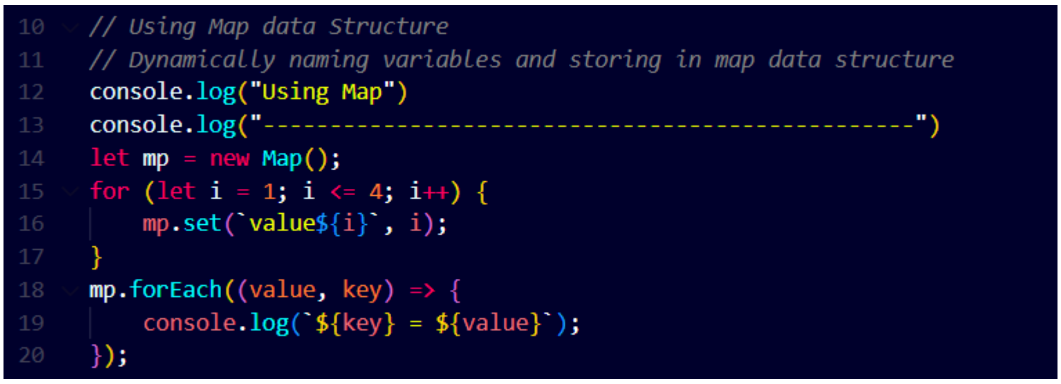
Output:



```
Variables
-------------------------------------------------
{ varName: 20 }
```

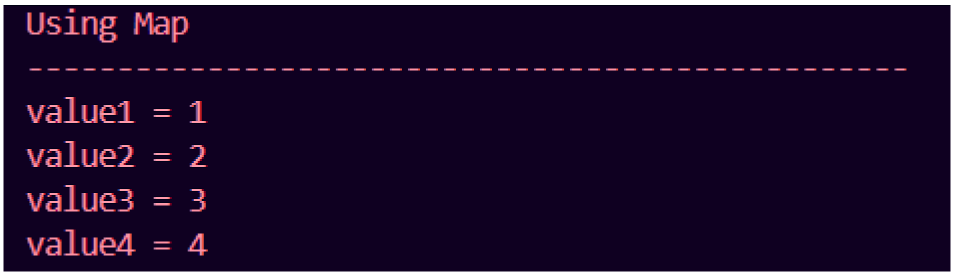
- **<u>Using Map Data Structure</u>**

Code:



```
// Using Map data Structure
// Dynamically naming variables and storing in map data structure
console.log("Using Map")
console.log("-------------------------------------------------")
let mp = new Map();
for (let i = 1; i <= 4; i++) {
    mp.set(`value${i}`, i);
}
mp.forEach((value, key) => {
    console.log(`${key} = ${value}`);
});
```

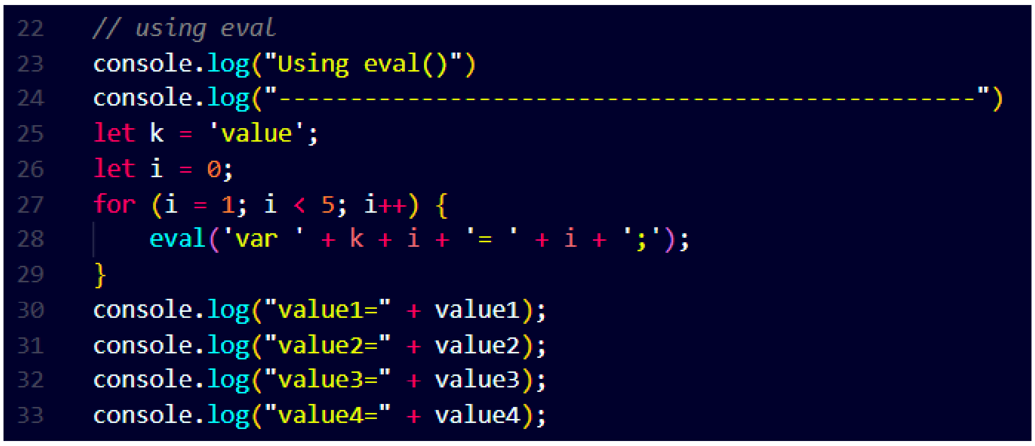
Output:



```
Using Map
-------------------------------------------------
value1 = 1
value2 = 2
value3 = 3
value4 = 4
```

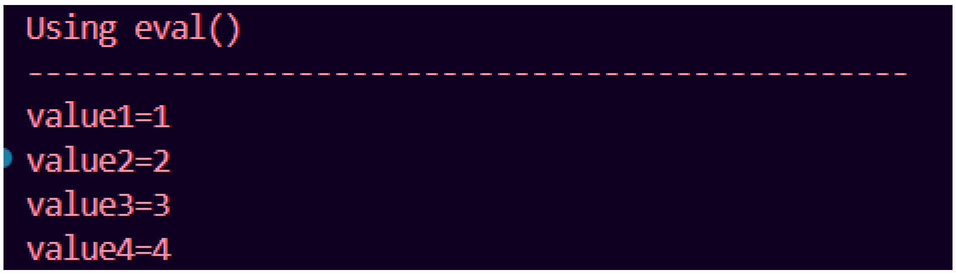
- **Using eval()**

  Code:



```javascript
// using eval
console.log("Using eval()")
console.log("--------------------------------------------------")
let k = 'value';
let i = 0;
for (i = 1; i < 5; i++) {
    eval('var ' + k + i + '= ' + i + ';');
}
console.log("value1=" + value1);
console.log("value2=" + value2);
console.log("value3=" + value3);
console.log("value4=" + value4);
```

  Output:

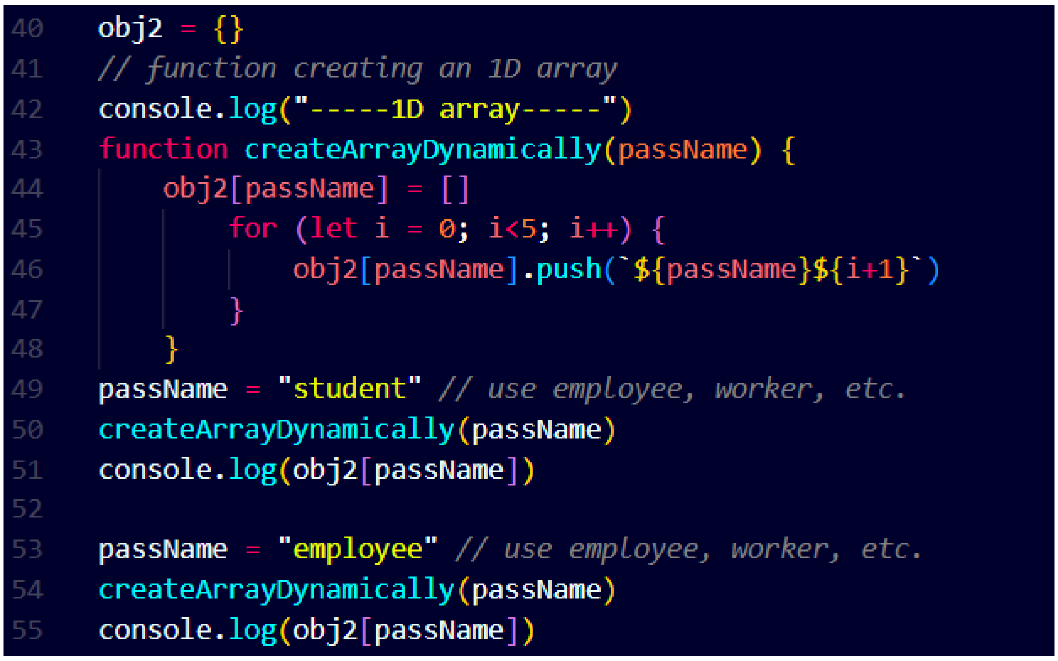


```
Using eval()
-------------------------------------------------
value1=1
value2=2
value3=3
value4=4
```

2) Arrays

- **1D array**

  Code:

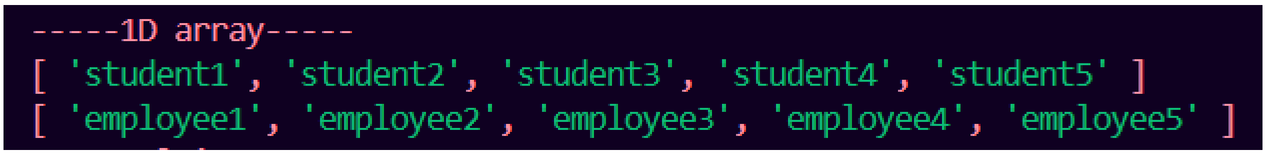


```javascript
obj2 = {}
// function creating an 1D array
console.log("-----1D array-----")
function createArrayDynamically(passName) {
    obj2[passName] = []
        for (let i = 0; i<5; i++) {
            obj2[passName].push(`${passName}${i+1}`)
        }
    }
passName = "student" // use employee, worker, etc.
createArrayDynamically(passName)
console.log(obj2[passName])

passName = "employee" // use employee, worker, etc.
createArrayDynamically(passName)
console.log(obj2[passName])
```

- Here function *createArrayDynamically()* creates the array with the name passed in the function as a param passName.

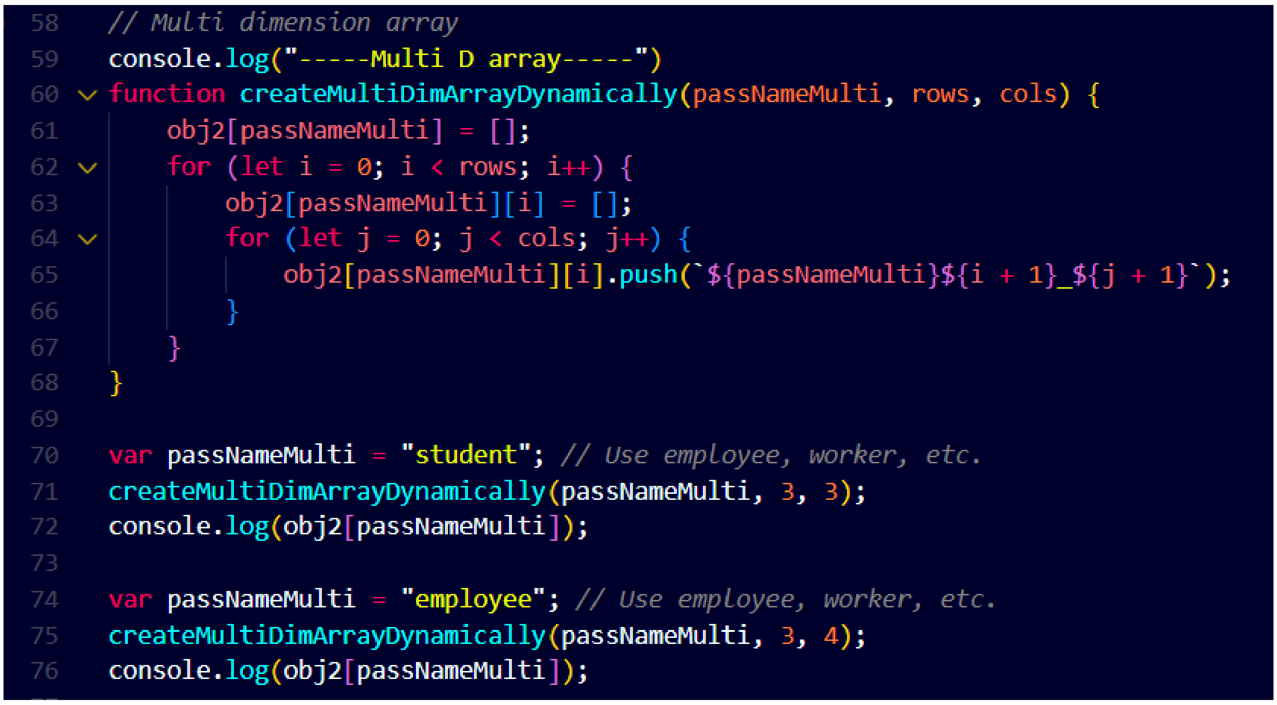
Output:



```
-----1D array-----
[ 'student1', 'student2', 'student3', 'student4', 'student5' ]
[ 'employee1', 'employee2', 'employee3', 'employee4', 'employee5' ]
```

- **Multi-Dimensional array**

  Code: Implementation similar to 1D array

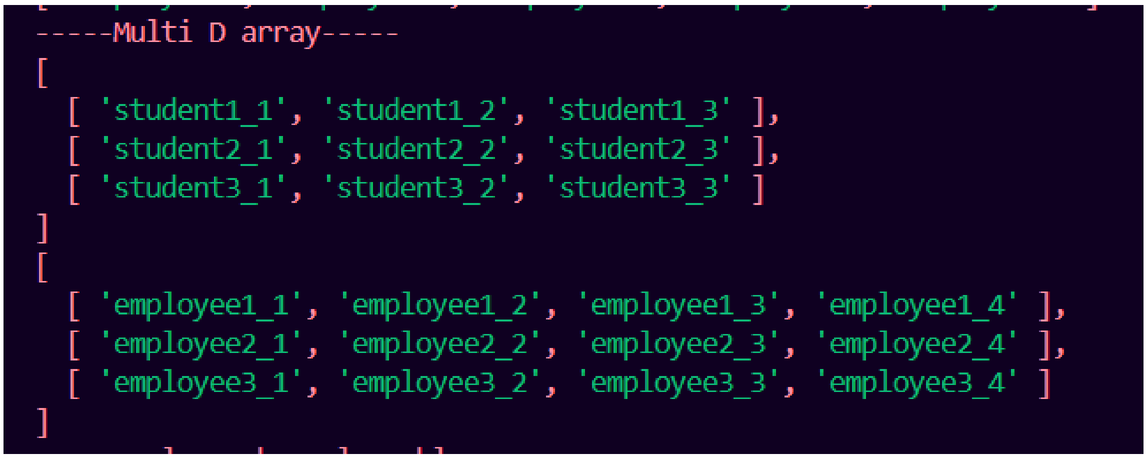


```javascript
// Multi dimension array
console.log("-----Multi D array-----")
function createMultiDimArrayDynamically(passNameMulti, rows, cols) {
    obj2[passNameMulti] = [];
    for (let i = 0; i < rows; i++) {
        obj2[passNameMulti][i] = [];
        for (let j = 0; j < cols; j++) {
            obj2[passNameMulti][i].push(`${passNameMulti}${i + 1}_${j + 1}`);
        }
    }
}

var passNameMulti = "student"; // Use employee, worker, etc.
createMultiDimArrayDynamically(passNameMulti, 3, 3);
console.log(obj2[passNameMulti]);

var passNameMulti = "employee"; // Use employee, worker, etc.
createMultiDimArrayDynamically(passNameMulti, 3, 4);
console.log(obj2[passNameMulti]);
```

Output:

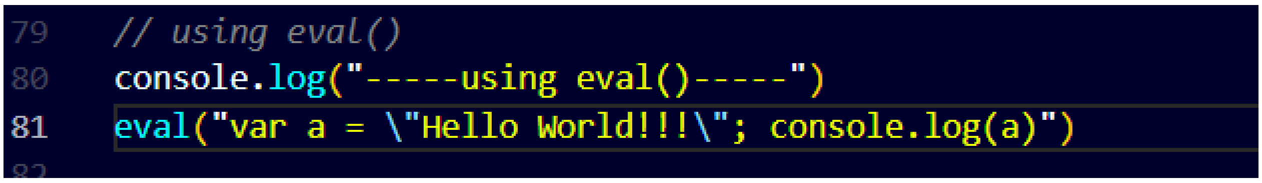


```
-----Multi D array-----
[
  [ 'student1_1', 'student1_2', 'student1_3' ],
  [ 'student2_1', 'student2_2', 'student2_3' ],
  [ 'student3_1', 'student3_2', 'student3_3' ]
]
[
  [ 'employee1_1', 'employee1_2', 'employee1_3', 'employee1_4' ],
  [ 'employee2_1', 'employee2_2', 'employee2_3', 'employee2_4' ],
  [ 'employee3_1', 'employee3_2', 'employee3_3', 'employee3_4' ]
]
```

3) **Using *eval():***

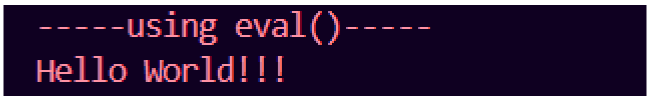
The eval() function evaluates JavaScript code represented as a string in the parameter. A string is passed as a parameter to eval(). If the string represents an expression, eval() evaluates the expression. Inside eval(), we pass a string in which variable value i is declared and assigned a value of i for each iteration. The eval() function executes this and creates the variable with the assigned values. The code given below implements the creation of dynamic variable names using eval().

Code:



```
// using eval()
console.log("-----using eval()-----")
eval("var a = \"Hello World!!!\"; console.log(a)")
```

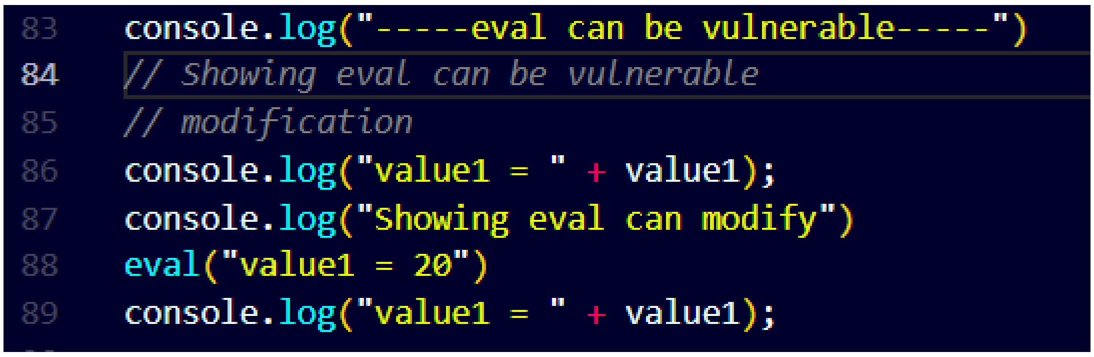
Output:



```
-----using eval()-----
Hello World!!!
```

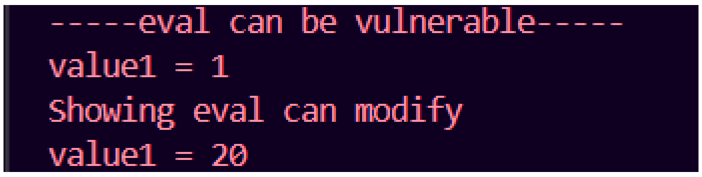
- <u>Using *eval()* can be vulnerable. Attackers can exploit it.</u>
  - Modifying the data

  Code:



```
console.log("-----eval can be vulnerable-----")
// Showing eval can be vulnerable
// modification
console.log("value1 = " + value1);
console.log("Showing eval can modify")
eval("value1 = 20")
console.log("value1 = " + value1);
```

  Output:



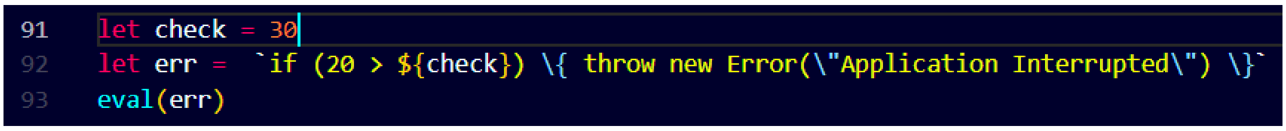
```
-----eval can be vulnerable-----
value1 = 1
Showing eval can modify
value1 = 20
```

  - Attackers can make problems to application.

  Ex: One can make application throw an error

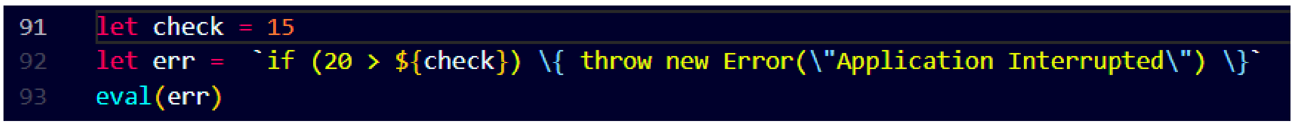
  Code:

    - Application works fine if *check > 20.*



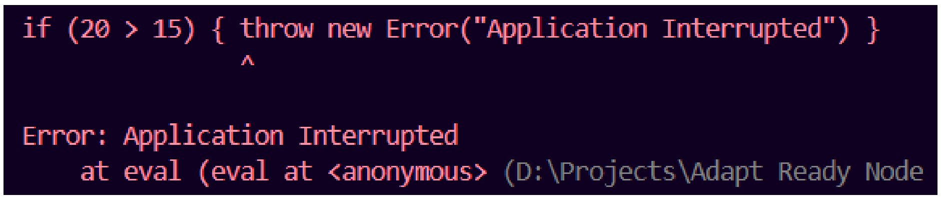
```
let check = 30
let err =  `if (20 > ${check}) \{ throw new Error(\"Application Interrupted\") \}`
eval(err)
```

    - If *check < 20*



```
let check = 15
let err =  `if (20 > ${check}) \{ throw new Error(\"Application Interrupted\") \}`
eval(err)
```

  Output:



```
if (20 > 15) { throw new Error("Application Interrupted") }
               ^

Error: Application Interrupted
    at eval (eval at <anonymous> (D:\Projects\Adapt Ready Node
```

2) What is the purpose of ssh keys and how we can use the ssh keys in server? explain about authorized keys in ssh with example.

- SSH stands for Secure Shell/ Secure Socket Shell.
- “The Secure Shell Protocol (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.”
- In general words “SSH keeps publicly transported messages private from public.”
- SSH use public key pairs or asymmetric cryptography to authenticate hosts to each other.

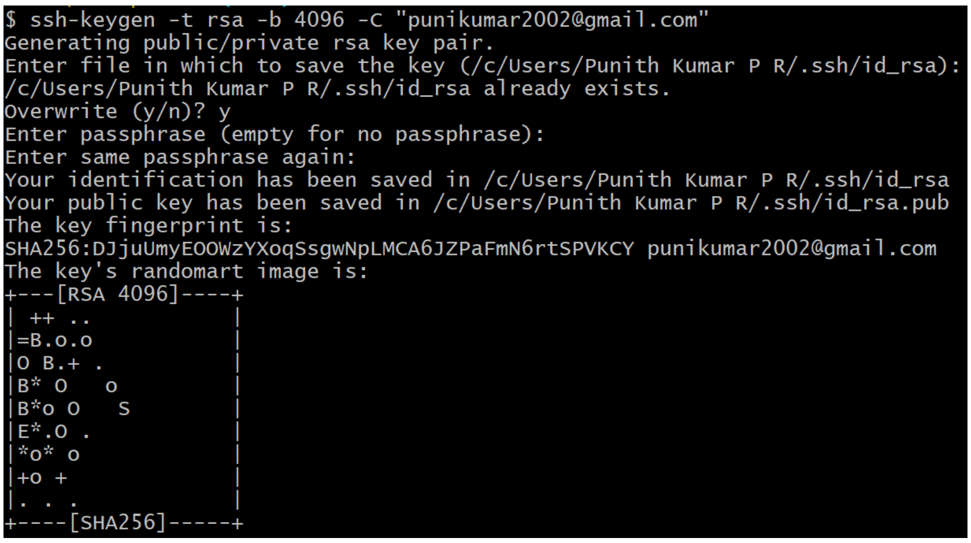
Purpose of SSH Keys

- SSH keys are used to access servers securely. They provide a way to authenticate without using passwords, which increases security and convenience.
- SSH keys comes in pairs of **private key** and **public key**.
- Private Key: This key stays on your local machine and should be kept secret. It's used to decrypt messages that were encrypted with the public key.
- Public Key: This key is placed on the server and can be shared openly. It’s used to encrypt messages that only the corresponding private key can decrypt.

How to generate and use SSH keys.

1) Generate SSH key

- Open git bash in local machine and enter the following command
  - ssh-keygen -t rsa -b 4096 -C "myemail@email.com"



```
$ ssh-keygen -t rsa -b 4096 -C "punikumar2002@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Punith Kumar P R/.ssh/id_rsa):
/c/Users/Punith Kumar P R/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Punith Kumar P R/.ssh/id_rsa
Your public key has been saved in /c/Users/Punith Kumar P R/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:DJjuUmyEOOWzYXoqSsgwNpLMCA6JZPaFmN6rtSPVKCY punikumar2002@gmail.com
The key's randomart image is:
+---[RSA 4096]----+
| ++ ..           |
|=B.o.o           |
|O B.+ .          |
|B* O   o         |
|B*o O   S        |
|E*.O .           |
|*o* o            |
|+o +             |
|. . .            |
+----[SHA256]-----+
```

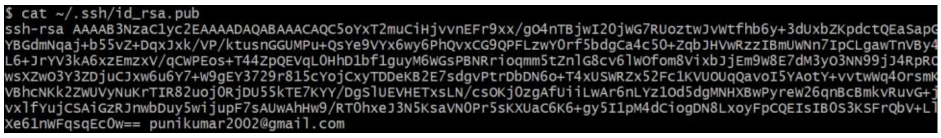
- This will generate a private key (id_rsa) and a public key (id_rsa.pub) in the ~/.ssh directory.
- Check for it using the ‘ls’ command
  - ls ~/.ssh

2) Copy public key to servers ‘~/.ssh/authorized_keys’

What are **authorized_keys**?

The authorized_keys file is used to store the public keys that are allowed to connect to the server. Each line in the authorized_keys file contains a single public key. When an SSH client attempts to connect, the server checks if the client’s public key is in the authorized_keys file.

- Following command will copy public key into *authorized_keys.*
    - ssh-copy-id user_name@server_ip_address
- Or append manually with cat command
    - cat ~/.ssh/id_rsa.pub | ssh user@server 'cat >> ~/.ssh/authorized_keys'



```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC5oYxT2muCiHjvvnEFr9xx/gO4nTBjwI2OjWG7RUoztwJvWtfhb6y+3dUxbZKpdctQEaSapG
YBGdmNqaj+b55vZ+DqxJxk/VP/ktusnGGUMPu+QsYe9VYx6wy6PhQvxCG9QPFLzwYOrf5bdgCa4c5O+ZqbJHVwRzzIBmUWNn7IpCLgawTnVBy4
L6+JrYV3kA6xzEmzxV/qCWPEos+T44ZpQEVqLOHhD1bf1guyM6WGsPBNRrioqmm5tZnlG8cv6lWOfom8VixbJjEm9W8E7dM3yO3NN99jJ4RpRC
wsXZwO3Y3ZDjuCJxw6u6Y7+W9gEY3729r815cYojCxyTDDeKB2E7sdgvPtrDbDN6o+T4xUSWRZx52Fc1KVUOUqQavoI5YAotY+vvtwWq4OrsmK
VBhcNKk2ZWUVyNuKrTIR82uojORjDU55kTE7KYY/DgSlUEVHETxsLN/csOKjOzgAfUiiLwAr6nLYz1Od5dgMNHXBwPyreW26qnBcBmkvRuvG+j
vxlfYujCSAiGzRJnwbDuy5wijupF7sAUwAhHw9/RTOhxeJ3N5KsaVNOPr5sKXUaC6K6+gy5I1pM4dCiogDN8LxoyFpCQEIsIBOS3KSFrQbV+Ll
Xe61nWFqsqEcOw== punikumar2002@gmail.com
```

- Or copy manually the public key into authorized_keys in server.

3) Connect to server

- connect to server with below command
    - ssh user_name@server_ip_address