

## Lesson 08 Demo 05

### Implementing HCL Functions

**Objective:** To demonstrate the use of HCL functions such as string manipulation, collections, and encoding

**Prerequisites:** Terraform installed

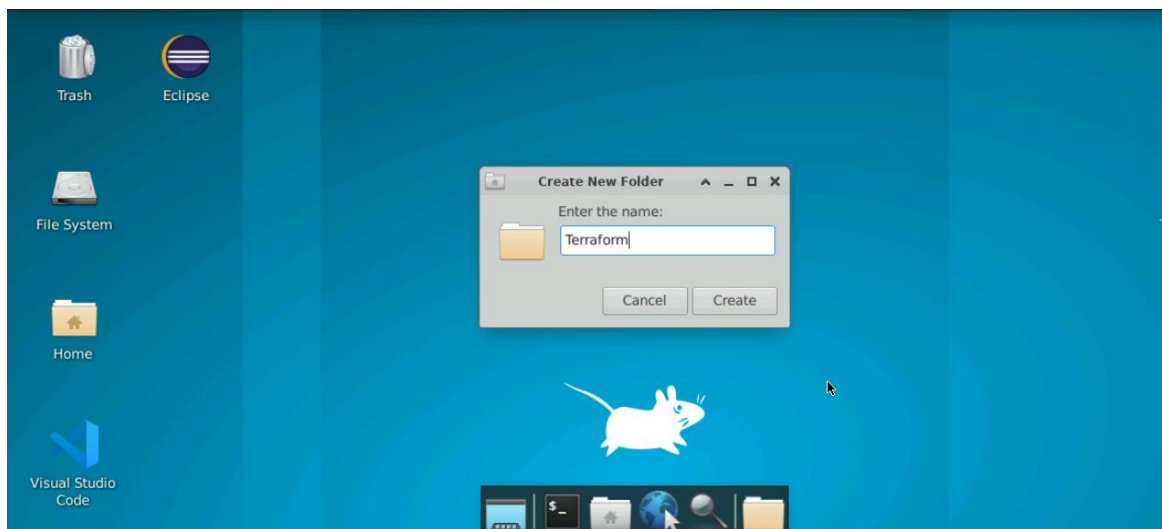
**Tools required:** Terraform and VS code

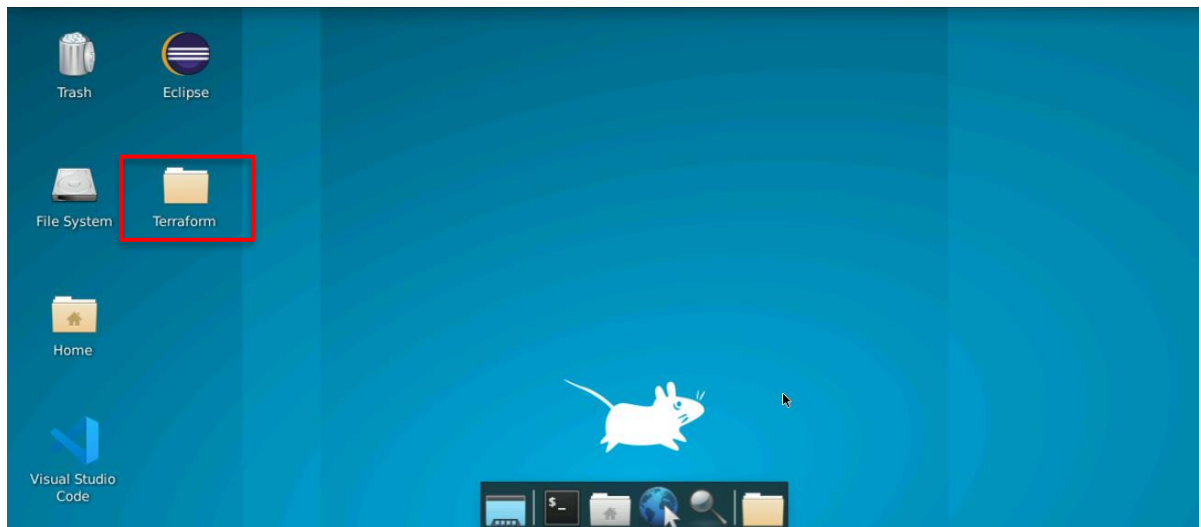
Steps to be followed:

1. Prepare files and set up Terraform
2. Write the HCL configuration using functions
3. Validate and apply the Terraform configuration

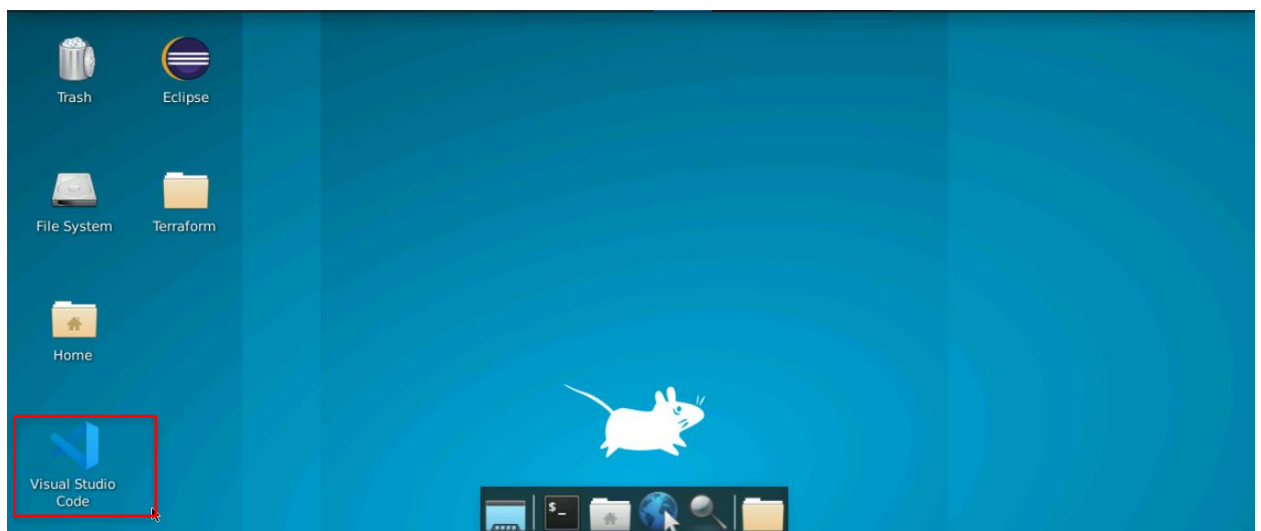
#### Step 1: Prepare files and set up Terraform

##### 1.1 Create a folder named **Terraform** on the desktop

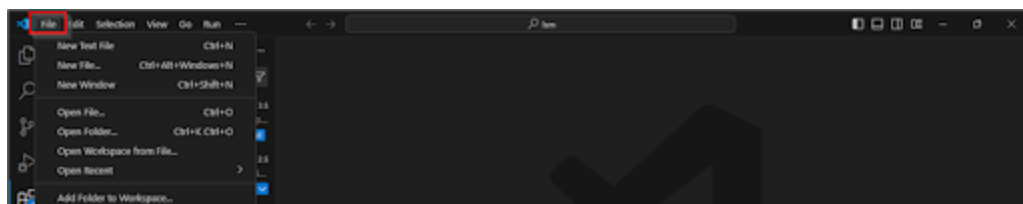




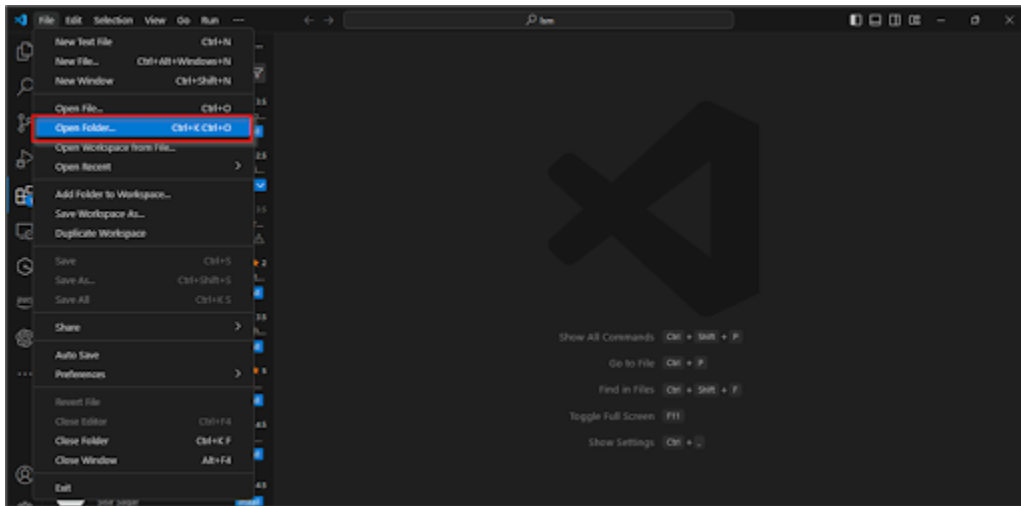
1.2 Double-click on the **VS Code editor** icon present on the desktop to open it



1.3 Open the **File** option present on the top console

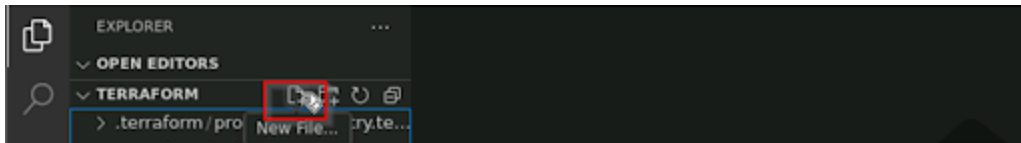


1.4 Click on the **Open Folder** from the drop-down menu to open the Terraform folder

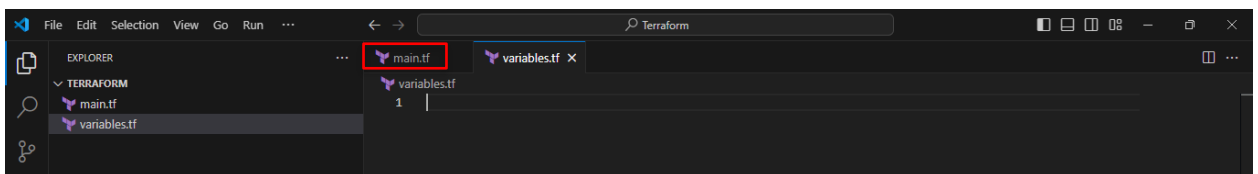


The **Terraform** folder will be opened in the VS Code editor.

1.5 Navigate to the **Terraform** folder on the editor and click on the **New File** option to create the main file



1.6 Create a file named **main.tf**



## Step 2: Write the HCL configuration using functions

2.1 In the **main.tf** file, copy the following code to manipulate strings:

```
variable "greeting" {
  default = "Hello"
}

variable "name" {
  default = "World"
}

output "message" {
  value = "${var.greeting}, ${upper(var.name)}!"
}

variable "list_example" {
  default = ["one", "two", "three"]
}

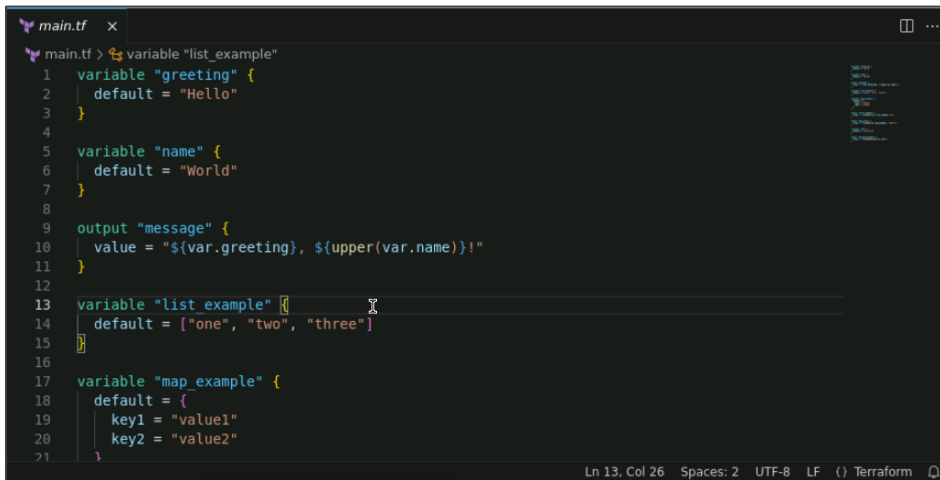
variable "map_example" {
  default = {
    key1 = "value1"
    key2 = "value2"
  }
}

output "first_element" {
  value = "${element(var.list_example, 0)}"
}

output "map_value" {
  value = "${lookup(var.map_example, "key1")}"
}

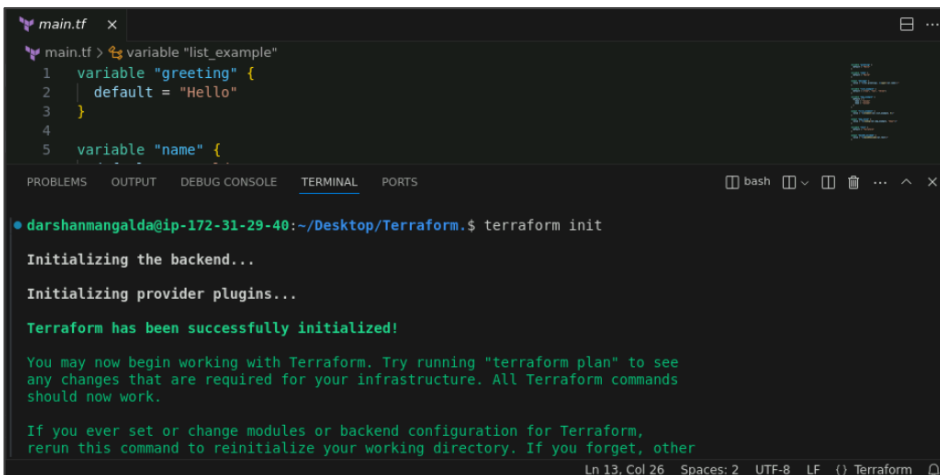
variable "text" {
  default = "Terraform"
}
```

```
output "base64_encoded" {  
  value = "${base64encode(var.text)}"  
}
```



```
main.tf  
1 variable "greeting" {  
2   default = "Hello"  
3 }  
4  
5 variable "name" {  
6   default = "World"  
7 }  
8  
9 output "message" {  
10  value = "${var.greeting}, ${upper(var.name)}!"  
11 }  
12  
13 variable "list_example" {  
14   default = ["one", "two", "three"]  
15 }  
16  
17 variable "map_example" {  
18   default = {  
19     key1 = "value1"  
20     key2 = "value2"  
21 }
```

- 2.2 Open the terminal in VS Code and run the following command to initialize terraform:  
**terraform init**



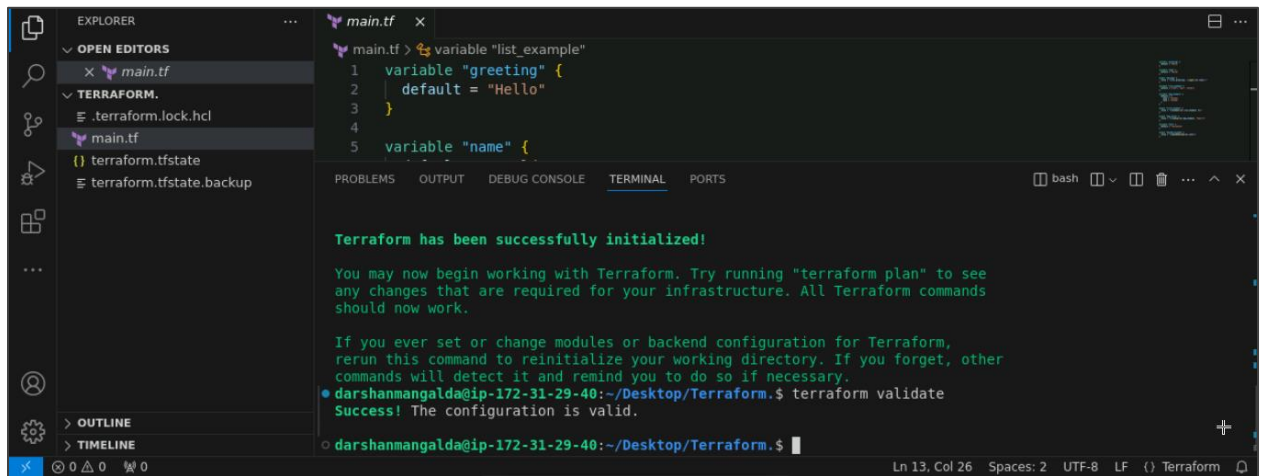
```
main.tf  
1 variable "greeting" {  
2   default = "Hello"  
3 }  
4  
5 variable "name" {  
6   default = "World"  
7 }  
8  
9 output "message" {  
10  value = "${var.greeting}, ${upper(var.name)}!"  
11 }  
12  
13 variable "list_example" {  
14   default = ["one", "two", "three"]  
15 }  
16  
17 variable "map_example" {  
18   default = {  
19     key1 = "value1"  
20     key2 = "value2"  
21 }  
22 }  
23 }  
24 }  
25 }  
26 }  
27 }  
28 }  
29 }  
30 }  
31 }  
32 }  
33 }  
34 }  
35 }  
36 }  
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
56 }  
57 }  
58 }  
59 }  
60 }  
61 }  
62 }  
63 }  
64 }  
65 }  
66 }  
67 }  
68 }  
69 }  
70 }  
71 }  
72 }  
73 }  
74 }  
75 }  
76 }  
77 }  
78 }  
79 }  
80 }  
81 }  
82 }  
83 }  
84 }  
85 }  
86 }  
87 }  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }
```

```
darshanmangal@ip-172-31-29-40:~/Desktop/Terraform$ terraform init  
  
Initializing the backend...  
  
Initializing provider plugins...  
  
Terraform has been successfully initialized!  
  
You may now begin working with Terraform. Try running "terraform plan" to see  
any changes that are required for your infrastructure. All Terraform commands  
should now work.  
  
If you ever set or change modules or backend configuration for Terraform,  
rerun this command to reinitialize your working directory. If you forget, other
```

## Step 3: Validate and apply the Terraform configuration

3.1 Run the following command to validate the configuration:

**terraform validate**



The screenshot shows the VS Code interface with a Terraform configuration file named `main.tf` open in the editor. The configuration defines two variables: `greeting` with a default value of "Hello" and `name`. The Explorer sidebar on the left shows the project structure, including `main.tf`, `.terraform.lock.hcl`, `terraform.tfstate`, and `terraform.tfstate.backup`. The Terminal panel at the bottom displays the output of the `terraform validate` command, which confirms that the configuration is valid. The output text is as follows:

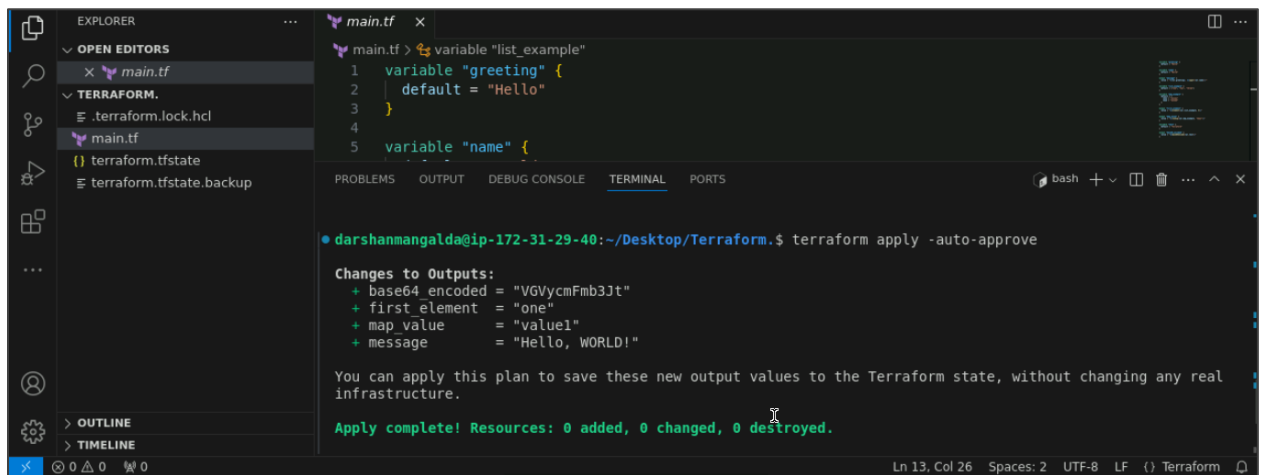
```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
• darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform.$ terraform validate
Success! The configuration is valid.
```

3.2 Execute the following command to apply the configuration:

**terraform apply -auto-approve**



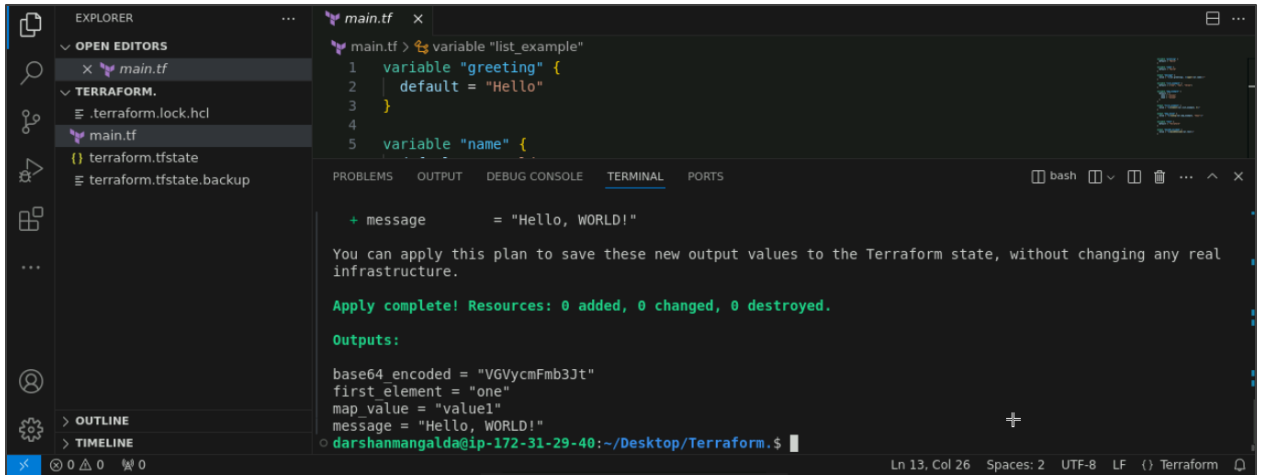
The screenshot shows the VS Code interface with the same `main.tf` file open. The Terminal panel at the bottom displays the output of the `terraform apply -auto-approve` command. The output shows the changes to the outputs and the successful application of the plan. The output text is as follows:

```
• darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform.$ terraform apply -auto-approve

Changes to Outputs:
+ base64_encoded = "VGVycmFmb3Jt"
+ first_element  = "one"
+ map_value      = "value1"
+ message        = "Hello, WORLD!"

You can apply this plan to save these new output values to the Terraform state, without changing any real
infrastructure.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```



The screenshot shows the Visual Studio Code interface with a Terraform configuration file named `main.tf` open in the editor. The file contains the following HCL code:

```
main.tf > variable "list_example"
1  variable "greeting" {
2    | default = "Hello"
3  }
4
5  variable "name" {
```

The Explorer sidebar on the left shows the project structure with files: `main.tf`, `.terraform.lock.hcl`, `terraform.tfstate`, and `terraform.tfstate.backup`. The bottom panel displays the `TERMINAL` output, which shows the successful execution of a Terraform plan and apply command. The output includes the message "Apply complete! Resources: 0 added, 0 changed, 0 destroyed." and the following outputs:

```
Outputs:
base64_encoded = "VG9yYmFmb3Jt"
first_element = "one"
map_value     = "value1"
message       = "Hello, WORLD!"
```

The terminal prompt is `darshanmangalada@ip-172-31-29-40: ~/Desktop/Terraform.$`. The status bar at the bottom indicates the current line and column as `Ln 13, Col 26`, with `Spaces: 2`, `UTF-8`, `LF`, and `Terraform` as the active language.

By following these steps, you have successfully demonstrated various HCL functions, including string manipulation, collections, and encoding using Terraform.