TASKS ON TERRAFORM

- **NAME: S.MUZZAMMIL HUSSAIN**
- **DATE: 22/07/2025**
- **BATCH: 11**
- **NO.OF TASKS: 1**

Task 1. Terraform Data Types

A **data type** defines what kind of data a variable can hold. Terraform supports multiple built-in data types, helping ensure structured, robust, and validated configurations. These are grouped into primitive and complex (composite/advanced) types.

Primitive Data Types

string

- **Description**: Alphanumeric text, can include symbols, wrapped in double quotes (").
- **Multiline**: Use \n for line breaks.
- **Definition**

```
variable "sample" {
  type    = string
  default = "test123"
}

variable "multiline" {
  type    = string
  default = "test123\nshgf"
}
```

number

- **Description**: Integer or floating-point numeric values.
- **Definition**

```
variable "int_example" {
  type    = number
  default = 125
}

variable "float_example" {
  type    = number
  default = 10.5
}
```

bool

- **Description**: Boolean value, true/false.
- **Definition**

```
variable "flag" {
  type    = bool
  default = true
}
```

any

- **Description**: Flexible, accepts any of the above types; is the default if type is not specified.
- **Usages**

```
variable "any_int"   { default = 10 }
variable "any_str"   { default = "test" }
variable "any_bool"  { default = false }
```

Complex / Composite / Advanced Data Types

list

- **Description**: Ordered sequence of values of the same type.
- **Definition**

```
# List of mixed types (with type = list)
variable "sample_list" {
  type    = list(any)
  default = ["test", 123, true, "test", 123]
}

# List of numbers
variable "num_list" {
  type    = list(number)
  default = [1,2,3,4,5,2,4,7,1,2]
}

# List of lists (matrices)
variable "matrix" {
  type    = list(list(number))
  default = [[1,2],[3,4],[5,6]]
}
```

- **Indexing**

```
var.num_list[2]     # 3
```

- **Error example**: If a list has ``, var.sample_list will error (index out of bounds).
- **Injecting via tfvars**

```
num_list = [1,2,5,6.7]
```

set

- **Description**: Unordered collection of unique values, duplicates removed automatically. Indexing is not guaranteed.
- **Definition**

```
variable "sample_set" {
```

```
  type    = set(number)
  default = [1,2,3,4,5,2,4,7,1,2]    # Stored as [1,2,3,4,5,7]
}
```

map

- **Description**: Key-value pair collection, keys must be strings.
- **Definition**

```
# Mixed value types
variable "sample_map" {
  type = map(any)
  default = {
    name     = "adi"
    id       = 123
    isactive = true
  }
}

# All string values
variable "map_str" {
  type = map(string)
  default = {
    name     = "adi"
    id       = "123"
    isactive = "yes"
  }
}

# All number values
variable "map_num" {
  type = map(number)
  default = {
    id    = 12345
    phone = 43154431
  }
}
```

- **Accessing values**

```
var.sample_map["name"]  # "adi"
var.sample_map.id       # 123
```

- **Error scenario**

```
var.sample_map.phoneno  # Error: key not found
```

- **Injecting via tfvars**

```
sample_map = { name = "test", dob = 123 }
```

- In Terraform, a map(list(string)) is:

A **map** (dictionary) where:

Each **key** is a string

- Each **value** is a list of strings

Think of it like a categorized list of string arrays.

Example:

```
variable "names_map" {
  type = map(list(string))
  default = {
    group1 = ["apple", "banana"]
    group2 = ["carrot", "date"]
    team3  = ["egg", "fig", "grape"]
  }
}
```

Practical Scenario

You want to create a file for each **group** (key) and write each item (value) in the file.
For the above names_map, we want to create:

Filename	**Content**
group1.txt	apple banana
group2.txt	carrot date
team3.txt	egg fig grape

Declaring the Variable

variables.tf

```
variable "names_map" {
  description = "Map of group names to lists of items"
  type        = map(list(string))
  default = {
    fruits  = ["apple", "banana"]
    veggies = ["carrot", "spinach"]
  }
}
```

tuple

- **Description**: Ordered sequence with fixed element types by position.
- **Definition**

```
text
variable "sample_tuple" {
  type    = tuple([string, number, bool])
  default = ["hello", 42, true]
}
```

object

- **Description**: User-defined type, grouping named attributes with explicit value types. Behaves like a structured map.
- **Definition**

```
variable "person" {
  type = object({
    name     = string
    id       = number
    isactive = bool
  })
  default = {
    name     = "adi"
    id       = 123
    isactive = true
  }
}
```

Summary Table

Type	Structure	Description	Example
String	Single value	Alphanumeric and symbols	"test123\nshgf"
number	Single value	Integer or float	125, 10.5
bool	Single value	true / false	true
any	Single value (any type)	Any primitive or composite	10, "test", false
list(type)	Ordered sequence	Same-type elements	``
set(type)	Unordered, unique collection	Same-type, duplicates removed	`` (even if default had repeats)
map(type)	Key-value pairs	Keys: strings, values: same type	{name="adi", id=123}
tuple([...])	Indexed, varying types	Each element's type is defined	["a", 2, true]
object({...})	Named attributes, fixed type	Structured, user-defined	{name="adi", id=123, isactive=true}