Terraform:
=========
Terraform is a tool for IAC.

IAC --> Insfrastructure as a code
Terraform is an Open source tool.
Terraform is developed by HashiCorp.

Cloudformation is also an IAC tool.

CF vs Terraform:
--> CF is used/restricted only to Aws.
--> Terraform is used to provision on multiple cloud providers.

Terraform uses HCL language (HCL --> hashi corp language)

How to install Terraform:?

1) Download Terraform
https://www.terraform.io/downloads.html
2) Unzip the terraform package
Extract the downloaded zip file, after extracting the zip file you can see terraform.exe file.
Extracted path can be Example "C:\Users\devops\Downloads\terraform_0.12.23_windows_amd64"
3) Configure environment variables for terraform
This PC(MyComputer)—>properties —>advanced system settings–>environment variables—>system var
iables—>path–edit–>new
paste the path "C:\Users\devops\Downloads\terraform_0.12.23_windows_amd64"
Click on Ok.
4) Verify terraform version
Open gitbash and enter
> terraform version
  Terraform v0.12.23

How a terraform configuration looks like?

```
<block> <resource_type> {
 option1
 option2
}
```

.tf --> extension for the terrform resource files.

```
resource "local_file" "my_pet" {
filename = "pets.txt"
content = "I love pets!"
}
```

Resource --> block
local --> provider
type --> type of resource
my_pet --> name for terraform
filename and content --> attributes used for the resource

Terraform Provider --> is a complete package of api calls to communicate with our resource.
idempotent

3 types of providers are available in terraform:
1) official --> provided by terraform
2) partner --> provided by third party vendors
3) community --> individual who can create.

Configuration Directory:
==========
Main.tf -->main configuration file containing resource definition
variables.tf --> contains varibles declaration
output.tf --> contains outputs from resources
provider.tf --> contains the provider definition

Terraform mutable vs immutable infrastructure:
==========
Terraform as a IAC tool uses immutable infrastructure stratgey.
Immutable means deleting the older infra and creating a newer one with new update.
Mutable means using the existing infra and updating the system with newer version.

Lifecycle rules:
==========
create_before_destroy
prevent_destroy
ignore_changes

Variables:
==========
```
variable "filename" {
   default =
   type = string
   description = This is optional (Used to user understanding)
}
```

Type	Example
String	I love pets
Number	1
bool	true/false
any	default value
list	["cat","dog"]
map	pet1= cat pet2=dog
object	complex data structure
tuple	complex data structure

Using of variables:
==========
1) By using varibles.tf file
2) By using interactive mode (This will get activated if we dont pass default value in variable.tf file)
3) Command line flags
--> terraform apply - var "filename=/root/pets.txt" -var "prefix=MR"

4) Environment variables
--> export TF_VAR_filename="/root.pets.txt"
--> export TF_VAR_prefix= "MR"
--> Set-Item -Path env:TF_VAR_filename -Value 'wild.txt'
terraform apply
5) varibale definition file (Should be end with terraform.tfvars/terraform.tfvars.json)
--> for automatically loaded file name *.auto.tfvars/*.auto.tfvars.json
--> if we are saving the file with other name like varible.tfvars then we need to pass this in CLI
--> terraform apply -var-file varibale.tfvars

Varible definition precedence:
==========
If we use multiple ways to define varibles for the same file then terraform uses varible definition preceden
ce

Example:
--> main.tf

```
resource local_file pet {
filename = var.filename
}
```

--> variable.tf

```
variable filename {
type = string
}
```

--> export TF_VAR_filename="/root/cat.txt"
--> Set-Item -Path env:TF_VAR_filename -Value 'wild.txt'
--> terraform.tfvars
filename = "/root/pets.txt"
--> varible.auto.tfvars
filename = "/root/mypet.txt"

--> terraforma apply -var "filename=/root/best-pet.txt"

Precedence order:
==========

in the above example we have passed all the possible varibles,which will terraform laod first and which wil
l override ?

Order	Option
1	Environment variables
2	Terraform.tfvars
3	*.auto.tfvars(alphabetical order)
4	-var or -var-file (Command line flags)

Resource Attribute reference:
==========
If i want to link two rerouces together by using resource attributes.

main.tf
======

```
resource "local_file" "pet" {
filename = "/root/pets.txt"
content = "My cat is MR.Cat"
}
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}
```

When we execute terraform apply it will create random id with pet name,
now i want to add this pet name in my content file (using output of one resource as input for another resou rce).

main.tf
======
```
resource "local_file" "pet" {
filename = "/root/pets.txt"
content = "My cat is ${random_pet.mypet.id}"      (random_pet = resource type,mypet = resource name, id = attribute)
}
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}
```

Output variables:
==========
These are used to display the output of the resources.
Ex:
```
resource "random_pet" "mypet" {
prefix = "MR"
separator = "."
length = "1"
}

output my-pet {
value = random_pet.my-pet.id
description = optional name
}
```

when we use terraform apply we can see the id as output.
we can use terraform output command to see the output of the resource.

Terraform state:
==========
Terraform state file will have the complete record of the infra created by terraform.
State file is considered as a blue print of all the resources terraform manages.
terraform.tfstate will be the name of the file and this will created only after using terraform apply command .

When we excute terraform apply then terraform will check for the state file config and main.tf configuration and make the changes.

If both the files are in sync and we are again trying to execute terraform apply then terraform will not make the changes but show
"Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed."

Each resource created by terraform will have the unique ID.
State files also capture the Metadata of the configuration file.
State file will helps for better performance because of the cache of the data.
state file benifits in collborating with different team members.
State files should be shared in the remote backend place so that team can access the state file.
State files also store the sensitive data so not recommended to store in public repo's like github,gitlab.
Terraform state is a json format file,never try to edit the state file manually.