```
# Global Variable
    1. Global variables  are declared outside the function
    2. Global variables can accessed inside the function and outside the function

Eg1:
x = 10 # Global Variable
print("Global V: ", x)

def d1():
   print("Local V: ", x) # Global Variable declared in function

d1()

Output
Global V:  10
Local V:  10
```

```
# Local Variables
    1. Local Variables are declared inside the function, we cannot access local
       variables outside the function

Eg2:
def d1():
   a = 10  # Local Variable
   b = 20
   print(a, b)
d1()
# print(a) # NameError: name 'a' is not defined
```

# same variable for local and global
globals() method does not take any parameters, we can modify global variable using global() method

def globals() -> dict[str, Any]
Return the dictionary containing the current scope's global variables.

Note: Updates to this dictionary *will* affect name lookups in the current global scope and vice-versa.

Eg3:
```
x = 10
print("Global V: ", x) Global V:  10

def d1():
   x = 20
   print("Local V: ", x) Local V:  20
   print("Global V:", globals()["x"]) Global V: 10 , globals()["key"])
d1()
```

Eg4:
```
x = 10
print("Global V: ", x) Global V:  10

def d1():
   x = 20
   print("Local V: ", x) Local V:  20
   print("Global V:", globals()["x"]) Global V: 10
   globals()["x"] = 30
   print("Global V:", globals()["x"]) Global V: 30
d1()
```

# locals() method
when we are using locals() method it will not modify the actual things

```
def locals() -> dict[str, Any]
Return a dictionary containing the current scope's local variables.

Note: Whether or not updates to this dictionary will affect name lookups in
the local scope and vice-versa is *implementation dependent* and not
covered by any backwards compatibility guarantees.
```

Eg5:
```
def d1():
   x = 20
   print("Local V: ", x) Local V:  20
   print("Local V:", locals()["x"]) Local V: 20
   locals()["x"] = 30 # using local we cannot modify actually things
   print("New Local V: ", x) New Local V:  20

d1()
```

```
Local V:  20
Local V: 20
New Local V:  20
```

```
# return locals and globals methods
Eg6:
x,y,z = 40,50,60
def g():
   return globals() # globals will execute all the modules data as key and value
def l():
   a,b,c = 10,20,30
   return locals()  # locals will execute only the key and value data
print('globals:', g())
print('locals:', l())

globals: {'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x000000070A636D00>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>,
 '__file__': 'E:\\Development\\PythonWorkspace\\13B_Functions\\F5.py',
'__cached__': None, 'x': 40, 'y': 50, 'z': 60, 'g': <function g at
0x000000070A67CF70>, 'l': <function l at 0x000000070A831160>}
locals: {'a': 10, 'b': 20, 'c': 30}
```

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\lenovo>py
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations
__': {}, '__builtins__': <module 'builtins' (built-in)>}
>>> locals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<class '_frozen_importlib.BuiltinImporter'>, '__spec__': None, '__annotations
__': {}, '__builtins__': <module 'builtins' (built-in)>}
>>> globals() == locals()
True
>>> globals() is locals()
True
>>> locals() is globals()
True
>>>
```

Eg7:

```
# globals and locals methods scope
print(globals() is locals()) # True
print(locals() is globals()) # True

def scopeof():
   print(globals() is locals()) # False # Here, False because of their scopes are different
   print(locals() is globals()) # False # Here, False because of their scopes are different
scopeof ()
```

# Global Keyword
global keyword allows us to modify the scope of local variable to global variable

Eg8:

```
x = 10
print("Accessing Global V: ", x)

def d1():
   global y
   y = 30
   print("Accessing Local V: ", y)

d1()
print('Calling Local V: ', y)
```

```
Output
Accessing Global V:  10
Accessing Local V:  30
Calling Local V:  30
```

Eg:9

```
# Assigning a function to a variable
def d1(a,b):
   print(a+b)

c = d1
c(5,10) # 15
print(type(c)) # <class 'function'>
```

Eg10:

```
# Function inside another function
def d1():
   print("d1 Outer Function") # d1 Outer Function
   def d2():
      print("d2 Inner Function") # d2 Inner Function
   d2()
d1()
```

```
d1 Outer Function
d2 Inner Function
```

Eg11:

```
# Function inside another function with parameters
def d1(a, b):
   def d2():
      print(a+b) # 15
   d2()

d1(10,5)
```

Eg12:

```
# Function inside another function with parameters
def d1(a, b):
   def d2():
      print('Addition: ', a + b) # Addition:  15
   d2()
   def d3():
      print('Subtraction: ', a - b) # Subtraction:  5
   d3()
d1(10,5)
```