```
"""
Namespace:
A namespace is a collection of all the names in the program.
It is a mapping of each variable to the objects.

There are 4 types of Namespaces:
1. Built-in Namespace:
   When the python interpreter is started, it comes with a
   default namespace.
   That is the reason we are able to use the functions like print, id etc.
2. Global Namespace (Module Namespace)
   Namespace defined by the module.
3. Local Namespace (Function Namespace)
   Namespace created for each function.
4. Enclosing Namespace (Nonlocal Namespace => Nested Functions)
   If a function has nested function, then a nested scope will be created.

LEGB Rule:
When a variable is referenced, then python searches for the local namespace first.
If the variable is not found in the local namespace,
Then it will search in the enclosing namespace.
If the variable is not found in the enclosing namespace,
Then it will search in the global namespace.
If the variable is not found in the global namespace,
then it will search in the built-in namespace.

Usage:
We can redefine a variable with the same name multiple times in multiple parts of
the program.
"""
from numpy.ma.core import outer

print("Learning Namespace")

a = 10
print(a)

def foo():
    b = 20
    print(b)

foo()


# Searching in Namespaces
type = 10

def foo():
    type = 20
    print("Inside Function: type = ", type) # 20

print("Before Function Call: ")
print("Outside Function: type = ", type) # 10

foo()

print("After Function Call: ")
print("Outside Function: type = ", type) # 10
```

```
id = 10

def foo():
    print("Inside Function: type = ", id) # 10

print("Before Function Call: ")
print("Outside Function: type = ", id) # 10

foo()

print("After Function Call: ")
print("Outside Function: type = ", id) # 10


def foo():
    print("Built In: ", range)


# Nested Functions
def foo():
    a = 10

    def bar():
        a = 20
        print("Inside Inner Function: a = ", a) # 20

    print("Inside Outer Function: ")
    print("Before Calling Inner Function: a = ", a) # 10
    bar()
    print("Inside Outer Function: ")
    print("After Calling Inner Function: a = ", a) # 10

foo()


# Nonlocal
def foo():
    a = 10

    def bar():
        nonlocal a
        a = 20
        print("Inside Inner Function: a = ", a) # 20

    print("Inside Outer Function: ")
    print("Before Calling Inner Function: a = ", a) # 10
    bar()
    print("Inside Outer Function: ")
    print("After Calling Inner Function: a = ", a) # 20

foo()


b = 100
def foo():
    b = 200
    print("Inside Function: b = ", b) # 200
```

```
print("Outside Function: ")
print("Before Calling Function: b = ", b) # 100

foo()

print("Outside Function: ")
print("After Calling Function: b = ", b) # 100


# global keyword
b = 100
def foo():
    global b
    b = 200
    print("Inside Function: b = ", b) # 200

print("Outside Function: ")
print("Before Calling Function: b = ", b) # 100

foo()

print("Outside Function: ")
print("After Calling Function: b = ", b) # 200


def outer_function():
    def inner_function():
        # print(non_existant_name)
        pass

    inner_function()

outer_function()



"""
Scope:
In what parts of the program can we access a variable.
There are three types of scopes.
1. local scope (function)
   The variables that are defined inside the function can be accessible within the
same function.
2. Enclosing (nonlocl scope)
   If there is a nested function, the variables that are defined
   inside the parent function can be accessed within the function and
   its nested functions using nonlocal keyword.
2. global scope (global scope)
   These can be accessed anywhere in the program and inside the function,
   we use global keyword to access them.

Lifetime:
To what duration the variable will be alive is known as the lifetime.
1. local variables:
   They are alive till the execution of the function.
   Once the function execution is complete,
   all the variables created inside the function will be destroyed.
2. Enclosing Variables (non local Variables):
   They are alive till the execution of the enclosing function.
```

```
   Once the enclosing function execution is complete,
   all the variables created inside the enclosing function will be destroyed.
3. Global Variables:
    They are alive till the execution of the program.
    Once the program execution is complete,
    all the variables created inside the module will be destroyed.
"""
# Local
x = 3
def foo():
    x = 30

    def bar():
        x = 300
        print(x) # 300
    bar()
    print(x) # 30


foo()


# Enclosing
x = 3
def foo():
    x = 30
    def bar():
        print(x) # 30
    bar()
    print(x) # 30


foo()

# Global
x = 3
def foo():
    def bar():
        print(x) # 3
    bar()
    print(x) # 3


foo()


# Local Variable
def baz():
    y = 10
    print(y)

baz()
# print(y)

# Enclosing Variable
def foo():
    p = 10

    def bar():
```

```
        print("Inside Inner Function: ", p) # 10

    bar()
    print("Inside Outer Function: ", p) # 10


foo()
# print(p)


# Global Variable
z = 30

def foo():
    print("Inside Outer Function: ", z)

    def bar():
        print("Inside Inner Function: ", z)

    bar()


foo()
print("Outside Function: ", z)


# global keyword
s = "Python"
def foo():
    s = "Program"
    print("Inside Function: ", s) # Program

foo()
print("Outside Function: ", s) # Python


s = "Python"
def foo():
    global s
    s = "Program"
    print("Inside Function: ", s) # Program

foo()
print("Outside Function: ", s) # Program


#
def foo():
    s = "Python"

    def bar():
        s = "Program"
        print("Inside Inner Function: ", s) # Program

    bar()
    print("Inside Outer Function: ", s) # Python

foo()
```

```
def foo():
    s = "Python"

    def bar():
        nonlocal s
        s = "Program"
        print("Inside Inner Function: ", s) # Program

    bar()
    print("Inside Outer Function: ", s) # Program

foo()


# Global and nonlocal
s = "Python"
def foo():
    s = "Java"

    def bar():
        nonlocal s
        s = "JavaScript"
        print("Inside Inner Function: ", s) # JavaScript

    bar()
    print("Inside Outer Function: ", s) # JavaScript

foo()
print("Outside Function: ", s) # Python


s = "Python"
def foo():
    s = "Java"

    def bar():
        global s
        s = "JavaScript"
        print("Inside Inner Function: ", s) # JavaScript

    bar()
    print("Inside Outer Function: ", s) # Java

foo()
print("Outside Function: ", s) # JavaScript
```