1. What is the scope?

The concept of scope rules how variables and names are looked up in our code. It determines the visibility of a variable within the code. The scope of a name or variable depends on the place in our code where we create that variable. The Python scope concept is generally presented using a rule known as the LEGB rule.

The letters is the acronym **LEGB** stand for **Local, Enclosing, Global, and Built-in** scopes.

- Local : Variables defined within a function
- Enclosing : Variables in the local scope of enclosing function
- Global : Variables defnied at the top of a module or explicitly declared global using the global keyword
- Built-in : Predifined variables

2. Global vs Local scope

In [10]:
```
# This is global because this code is in main body
x = 'global x'

# This is local because this variable is in test function
def test():
    y = 'local y'
    # print(y)
    print(x)

# Local scope(X) -> Enclosing scope(X) -> Global scope(O) : print global x
test()
```

```
global x
```

In [11]:
```
print(y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-d9183e048de3> in <module>
----> 1 print(y)

NameError: name 'y' is not defined
```

In [12]:
```
def test():
    x = 'local x'
    print(x)

# Return local x variable
test()

# Return global x variable
print(x)
```

```
local x
global x
```

2.1 global statement

global statment explicitly tell Python that the x variable we want to work with is the global variable

In [13]:
```
def test():
    global x
    x = 'local x'
    print(x)

test()
print(x)
```

```
local x
local x
```

3. Enclosing scope

Enclosing scope is a special scope that only exists for nested functions. If the local scope is a inner or nested function, then the enclosing scope is the scope of the outer or enclosing function. This scope contains the names that we define in the

In [18]:
```
def outer():
    x = 'outer x'

    def inner():
        x = 'inner x'
        print(x)

    # This x variable is in inner local scope
    inner()
    # This x variables is in outer enclosing scope
    print(x)

outer()
```

```
inner x
outer x
```

In [16]:
```
def outer():
    x = 'outer x'

    def inner():
        # x = 'inner x'
        print(x)

    # This x variable is in enclosing scope
    inner()
    # This x variables is in enclosing scope
    print(x)

outer()
```

```
outer x
outer x
```

3.1 nonlocal statement

The nonlocal statement causes the listed identifier to refer to previous bound variables in the nearest enclosing scope excluding globals. This is important because the default behavior for binding is to search the local namespace first.

In [19]:
```
def outer():
    x = 'outer x'

    def inner():
        nonlocal x
        x = 'inner x'
        print(x)

    # This x variable is in enclosing scope
    inner()
    # This x variables is in enclosing scope
    print(x)

outer()
```

```
inner x
inner x
```

4. Wrap all scopes

In [21]:
```
# x variable in global scope
x = 'global x'

def outer():
    # x variable in enclosing scope
    x = 'outer x'

    # x variable in inner scope
    def inner():
        x = 'inner x'
        print(x)

    inner()
    print(x)

outer()
print(x)
```

```
inner x
outer x
global x
```