**Name:- Kunal Pandey**

**Reg no:- 2141020**

Local, Non-local, and Global Variables in Python

In Python, variables can have different scopes, which determine where they can be

accessed and modified within the code. The three main scopes are local, non-local, and

global.

1. Local Variables:

Local variables are defined within a specific function or block of code and are only

accessible within that function or block. They are created when the function is called

and destroyed when the function returns. Local variables cannot be accessed from

outside the function in which they are defined.

```python

def example\_function():

x = 10 # This is a local variable

print("Inside the function: x =", x)

example\_function()

# Attempting to access x outside the function will raise an error:

# print("Outside the function:", x) # This will result in an error

```

Explanation:

In the above code, we have a function `example\_function()` that defines a local variable

`x`. When the function is called, it prints the value of `x` inside the function. However,

trying to access `x` outside the function will result in an error because it is a local

variable and is only accessible within the function's scope.

2. Non-local Variables:

Non-local variables are used within nested functions and are defined in an enclosing

function. They are accessible within the nested function but not available outside the

enclosing function.

```python

def outer\_function():

y = 20 # This is a non-local variable

def inner\_function():

nonlocal y

print("Inside the inner function: y =", y)

y = 30 # Modifying the value of the non-local variable

inner\_function()

print("After inner function: y =", y)

outer\_function()

```

Explanation:

In the above code, we have an `outer\_function()` that defines a non-local variable `y`.

Inside `outer\_function()`, there is another nested function called `inner\_function()`. By

using the `nonlocal` keyword, we can access and modify the non-local variable `y` from

within the `inner\_function()`. After calling `outer\_function()`, it will print the value of `y`

before and after the `inner\_function()` call, showing the modification.

3. Global Variables:

Global variables are defined at the top level of the script and are accessible throughout

the entire code, including within functions. To modify a global variable inside a function,

you need to use the `global` keyword.

```python

global\_var = 50 # This is a global variable

def example\_function():

global global\_var

global\_var += 10 # Modifying the value of the global variable

print("Inside the function: global\_var =", global\_var)

example\_function()

print("Outside the function: global\_var =", global\_var)

```

Explanation:

In the above code, we have a global variable `global\_var`. The `example\_function()`

accesses and modifies the global variable using the `global` keyword. When the

function is called, it prints the updated value of `global\_var`, and after the function call,

the modified value can be seen outside the function as well.

Variables Starting with a Single Underscore and Double Underscore

In Python, variables starting with a single underscore `\_variable` and variables starting

with a double underscore `\_\_variable` have different conventions and behaviors.

1. Variable starting with a Single Underscore `\_variable`:

Variables starting with a single underscore are considered as a weak "internal use"

indicator. It is a convention rather than a strict rule enforced by Python. The variable is

still accessible and can be modified just like any other variable.

```python

def example\_function():

\_internal\_var = 100 # Variable starting with a single underscore

print("Inside the function: \_internal\_var =", \_internal\_var)

example\_function()

# Variables starting with a single underscore can be accessed outside the function:

print("Outside the function:", \_internal\_var) # This is allowed but not recommended

```

Explanation:

In the above code, we have a variable `\_internal\_var` that starts with a single

underscore. This variable is used inside the `example\_function()` and can also be

accessed outside the function. However, using a single underscore is a convention to

indicate that this variable is intended for internal use within the module, and it is not

meant to be accessed directly from outside the module.

2. Variable starting with a Double Underscore `\_\_variable`:

Variables starting with a double underscore are used for name mangling to make the

variable name more unique. This feature is mostly used in the context of class attributes

to avoid accidental name clashes in inheritance.

```python

class ExampleClass:

def \_\_init\_\_(self):

self.\_\_private\_var = 42

def get\_private\_var(self):

return self.\_\_private\_var

obj = ExampleClass()

print(obj.get\_private\_var())

# Attempting to access the \_\_private\_var directly will raise an error:

# print(obj.\_\_private\_var) # This will result in an error

```

Explanation:

In the above code, we have a class `ExampleClass` that defines a variable

`\_\_private\_var`. This variable is used to store a private value within the class. However,

Python performs name mangling on such variables, so accessing `\_\_private\_var`

directly from outside the class will raise an error. To access this variable outside the

class, we use a getter method (`get\_private\_var()` in this case) that provides controlled

access to the private variable.