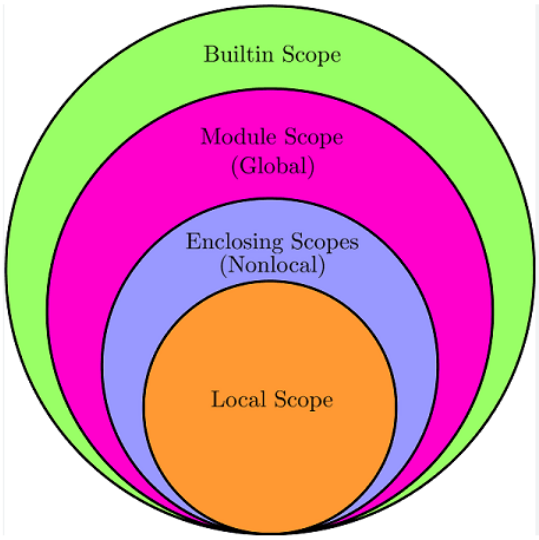
Python Scope Resolution

Variable Scope – where a viable is both visible and accessible.

Scope Resolution – **LEGB**: *Local* → *Enclosed* → *Global* → *Built-in*. When using a variable, there is an order called the LEGB rule which is a kind of name lookup procedure, which determines the order in which Python looks up names.



Local Scope – variables declared within a function have a **local** scope. Functions cannot see inside of other functions.

```
def func1():
    a = 1
    print(a)

def func2():
    b = 2
    print(b)

func1()
func2()
```

Here, variable **a** is **local** to function 1, variable **b** is **local** to function 2. If we placed "print(a)" inside function 2, it would throw an error about the variable a is not defined.

Enclosed Scope – If a **function** is called within another function, it will first try to use the **local** variable, if there is no **local** variable, it will use the **enclosed** variable.

```
def func1():
    x = 1 # <-- enclosed variable to func2
    print(x)
    def func2():
        x = 2 # <-- local variable to func2
        print(x)
    func2()
func1()
```

```
def func1():
    x = 1 # <-- enclosed variable to func2
    print(x)
    def func2():
        #x = 2 # <-- local variable to func2
        print(x)
    func2()
func1()
```

In the first example, func1 has a local variable x set to 1 and embedded in this function is another function, func2, which has its own local variable, also named x.

Func2 will prioritise the use of its local variable x and print 2.

In the second example, the local variable in func2 has been commented out so func2 will now use the value for x in func1, which is called the enclosed variable.

Global Scope – variables outside any functions

```
def func1():
    #x = 1 # <-- enclosed variable to func2
    print(x)
    def func2():
        #x = 2 # <-- local variable
        print(x)
    func2()

x = 3 # <-- global variable
func1()
```

In this example, both func1 and func2 have their **local variable** commented out, so func2 will not find a **local** variable, it will look up into func1 and not find an **enclosed** variable, so it will use the **global** variable. Similarly, func1 has no **local** variable so it will also use the **global** variable x = 3.

Built-in Scope – variables that are predefined like pi or e

```
from math import pi

def func1():
    print(pi)  # <-- this variable pi is built-in

pi = 3.14 # <-- this variable pi is global
func1()
```

Because global scope is prioritised over built-in scope, a global version of pi will be displayed if it exists, if not, the built-in version will be used.