Scope (1A)

Young Won Lim
10/22/23

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Python Scope

A variable is only available
from inside the region it is created.

This is called scope.

https://www.w3schools.com/python/python_scope.asp

Local Scope

A variable created inside a function
belongs to the local scope of that function,
and can only be used inside that function.

A variable created inside a function
is available inside that function:

```
def myfunc():
    x = 300
    print(x)

myfunc()
```

https://www.w3schools.com/python/python_scope.asp

Function Inside Function

the variable **x** is not available outside the function,
but it is available for any function inside the function:

The local variable can be accessed
from a function within the function:

```
def myfunc():
    x = 300
    def myinnerfunc():
        print(x)
    myinnerfunc()

myfunc()
```

https://www.w3schools.com/python/python_scope.asp

Global Scope

A variable created in the main body of the Python code
is a global variable and belongs to the global scope.

Global variables are available
from within any scope, global and local.

A variable created outside of a function is
global and can be used by anyone:

```
x = 300

def myfunc():
    print(x)


myfunc()

print(x)
```

https://www.w3schools.com/python/python_scope.asp

Naming Variables

If you operate with the same variable name
inside and outside of a function,
Python will treat them as two separate variables,
one available in the global scope (outside the function) and
one available in the local scope (inside the function):

The function will print the local **x** (= 300),
and then the code will print the global **x** (= 200):

```
x = 300

def myfunc():
    x = 200
    print(x)                # local x (=200)

myfunc()

print(x)                    # global x (= 300)
```

https://www.w3schools.com/python/python_scope.asp

Global Keyword (1)

If you need to create a global variable,
but are stuck in the local scope,
you can use the global keyword.

The global keyword makes the variable global.

If you use the global keyword,
he variable belongs to the global scope:

```
def myfunc():
    global x
    x = 300                # global x (= 300)


myfunc()

print(x)                   # global x (= 300)
```

https://www.w3schools.com/python/python_scope.asp

Global Keyword (2)

Also, use the global keyword
if you want to make a change
to a global variable inside a function.

To change the value of a global variable
inside a function, refer to the variable
by using the global keyword:

```
x = 300

def myfunc():
    global x
    x = 200

myfunc()

print(x)                # global x (= 200)
```

https://www.w3schools.com/python/python_scope.asp

Variable Scope

how to initialize a variable.

the scope of these variables

Not all variables can be accessed
from anywhere in a program.

The part of a program where a variable is accessible
is called its scope.

here are four major types of variable scope and
is the basis for the LEGB rule.

LEGB stands for Local -> Enclosing -> Global -> Built-in.

https://www.datacamp.com/tutorial/scope-of-variables-python

Local Scope

Whenever you define a variable within a function,
its scope lies ONLY within the function.

It is accessible from the point at which it is defined
until the end of the function and
exists for as long as the function is executing

Which means its value cannot be changed
or even accessed from outside the function.

https://www.datacamp.com/tutorial/scope-of-variables-python

Enclosing Scope

What if we have a nested function
(function defined inside another function)?

```
def outer():
    first_num = 1
    def inner():
        second_num = 2
        # Print statement 1 - Scope: Inner
        print("first_num from outer: ", first_num)
        # Print statement 2 - Scope: Inner
        print("second_num from inner: ", second_num)
    inner()
    # Print statement 3 - Scope: Outer
    print("second_num from inner: ", second_num)

outer()
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Enclosing Scope

```
first_num from outer:  1
second_num from inner:  2



---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-4-13943a1eb01e> in <module>
     11     print("second_num from inner: ", second_num)
     12
---> 13 outer()


<ipython-input-4-13943a1eb01e> in outer()
      9     inner()
     10     # Print statement 3 - Scope: Outer
---> 11     print("second_num from inner: ", second_num)
     12
     13 outer()


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

https://www.datacamp.com/tutorial/scope-of-variables-python

Enclosing Scope

an error

because you cannot access second_num from **outer**()
(# Print statement 3). It is not defined within that function.

However, you can access first_num from **inner**()
(# Print statement 1),
because the scope of first_num is larger, it is within **outer**().
This is an enclosing scope.

Outer's variables have a larger scope and
can be accessed from the enclosed function **inner**().

https://www.datacamp.com/tutorial/scope-of-variables-python

Global Scope

Whenever a **variable** is defined outside any function,
it becomes a global variable, and its scope is
anywhere within the program.

Which means it can be used by any function.

```
greeting = "Hello"

def greeting_world():
    world = "World"
    print(greeting, world)

def greeting_name(name):
    print(greeting, name)

greeting_world()
greeting_name("Samuel")
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Built-in Scope

This is the widest scope

All the special reserved keywords
are under built-in scope.

We can call the keywords
anywhere within our program
without having to define them before use.

keywords are simply special reserved words.

They are kept for specific purposes
and cannot be used for any other purpose in the program.

These are the keywords in Python:

Keywords in Python

https://www.datacamp.com/tutorial/scope-of-variables-python

Python Keywords

False
None
True
And
as
assert
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield

https://www.datacamp.com/tutorial/scope-of-variables-python

LEGB Rule

LEGB (Local -> Enclosing -> Global -> Built-in)
is the logic followed by a Python interpreter
when it is executing your program.

Let's say you're calling **print(x)** within **inner()**,
which is a function nested in **outer()**.

Then Python will first look
if "**x**" was defined locally within **inner()**.

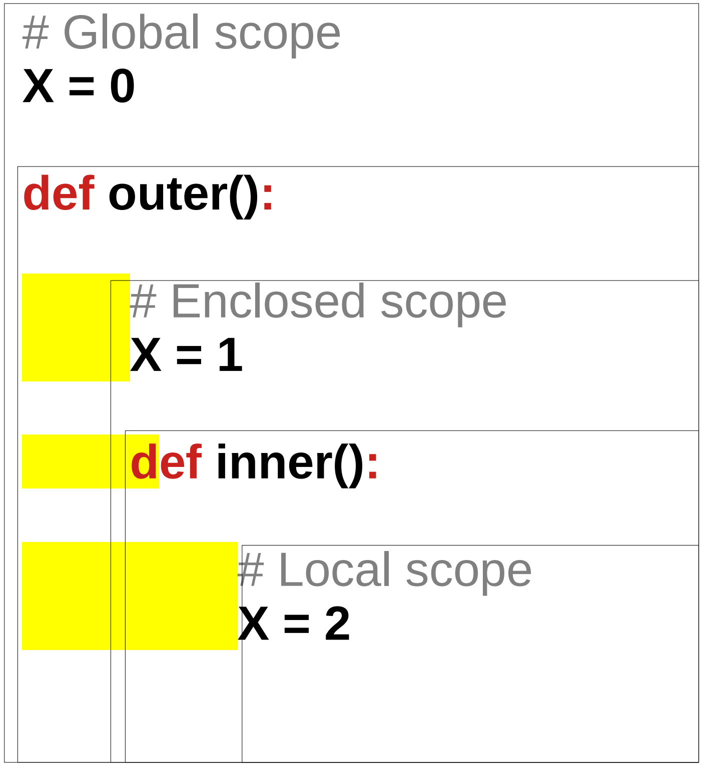
If not, the variable defined in **outer()** will be used.
This is the enclosing function.

If it also wasn't defined there,
the Python interpreter will go up another level
- to the global scope.

Above that, you will only find the built-in scope,
which contains special variables reserved for Python itself.

https://www.datacamp.com/tutorial/scope-of-variables-python

LEGB Rule



```
# Global scope
X = 0

def outer():

    # Enclosed scope
    X = 1

    def inner():

        # Local scope
        X = 2
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Global scope example

to change the global variable **greeting** ("Hello")
to set a new **value** ("Hi")

```
greeting = "Hello"

def change_greeting(new_greeting):
    greeting = new_greeting

def greeting_world():
    world = "World"
    print(greeting, world)

change_greeting("Hi")
greeting_world()
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Global scope example

because when we set the value of **greeting** to "Hi",
it created a new local variable **greeting**
in the scope of change_greeting().

It did not change anything for the global greeting.

This is where the global keyword comes in handy.

https://www.datacamp.com/tutorial/scope-of-variables-python

Global keyword example

With global keyword, can use the globally defined variable
instead of locally creating one.

```
greeting = "Hello"

def change_greeting(new_greeting):
    global greeting
    greeting = new_greeting

def greeting_world():
    world = "World"
    print(greeting, world)


change_greeting("Hi")
greeting_world()
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Non-local keyword example

The nonlocal keyword is useful in nested functions.
It causes the variable to refer to the previously bound variable
in the closest enclosing scope.

it will prevent the variable from trying to bind locally first,
and force it to go a level 'higher up'.

```
def outer():
    first_num = 1
    def inner():
        nonlocal first_num
        first_num = 0
        second_num = 1
        print("inner - second_num is: ", second_num)
    inner()
    print("outer - first_num is: ", first_num)

outer()
```

https://www.datacamp.com/tutorial/scope-of-variables-python

Scoping rule (1)

Actually, a concise rule for Python Scope resolution,
from Learning Python, 3rd. Ed.

these rules are specific to variable names, not attributes.
If you reference it without a period, these rules apply.

LEGB Rule

Local
Names assigned in any way within a function (def or lambda),
and not declared global in that function

Enclosing-function
Names assigned in the local scope of any and all
statically enclosing functions (def or lambda), from inner to outer

Global (module)
Names assigned at the top-level of a module file,
or by executing a global statement in a def within the file

Built-in (Python)
Names preassigned in the built-in names module:
open, *range*, *SyntaxError*, etc

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (2)

```
code1
class Foo:
    code2
    def spam():
        code3
        for code4:
            code5
            x()
```

The **for** loop does not have its own namespace.
In LEGB order, the scopes would be

- L: Local in def spam (in code3, code4, and code5)
- E: Any enclosing functions (if the whole example were in another def)
- G: Were there any **x** declared globally in the module (in code1)?
- B: Any builtin **x** in Python.

x will never be found in code2
(even in cases where you might expect it would, see Antti's answer or here).

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (3-1)

```
from __future__ import print_function  # for python 2 support

x = 100
print("1. Global x:", x)
class Test(object):
    y = x
    print("2. Enclosed y:", y)
    x = x + 1
    print("3. Enclosed x:", x)

    def method(self):
        print("4. Enclosed self.x", self.x)
        print("5. Global x", x)
        try:
            print(y)
        except NameError as e:
            print("6.", e)
```

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (3-2)

```
    def method_local_ref(self):
        try:
            print(x)
        except UnboundLocalError as e:
            print("7.", e)
        x = 200 # causing 7 because has same name
        print("8. Local x", x)
```

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (4)

```
inst = Test()
inst.method()
inst.method_local_ref()
```

output:

```
1. Global x: 100
2. Enclosed y: 100
3. Enclosed x: 101
4. Enclosed self.x 101
5. Global x 100
6. global name 'y' is not defined
7. local variable 'x' referenced before assignment
8. Local x 200
```

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (5-1)

Essentially, the only thing in Python
that introduces a new scope is a function definition.

Classes are a bit of a special case
in that anything defined directly in the body
is placed in the class's namespace,
but they are not directly accessible
from within the methods (or nested classes) they contain.

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

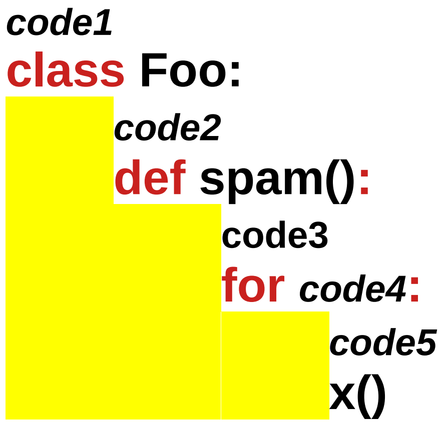
Scoping rule (5-2)

In your example there are only 3 scopes
where x will be searched in:

spam's scope - containing everything defined
in code3 and code5 (as well as code4, your loop variable)

The global scope - containing everything defined in code1,
as well as Foo (and whatever changes after it)

The builtins namespace. A bit of a special case -
this contains the various Python builtin functions and types
such as len() and str().
Generally this shouldn't be modified by any user code,
so expect it to contain the standard functions and nothing else.



code1
class Foo:
code2
def spam():
code3
for *code4*:
code5
x()

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (6)

More scopes only appear
when you introduce a nested function (or lambda).

These will behave pretty much as you'd expect however.

The nested function can access everything in the local scope,
as well as anything in the enclosing function's scope.

```
def foo():
    x=4
    def bar():
        print x         # Accesses x from foo's scope
    bar()               # Prints 4
    x=5
    bar()               # Prints 5
```

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (7)

Restrictions:

Variables in scopes other than the local function's variables can be accessed,
but can't be rebound to new parameters without further syntax.

Instead, assignment will create a new local variable I
nstead of affecting the variable in the parent scope.

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (7)

```
global_var1 = []
global_var2 = 1

def func():
    # This is OK: It's just accessing, not rebinding
    global_var1.append(4)

    # This won't affect global_var2. Instead it creates a new variable
    global_var2 = 2

    local1 = 4
    def embedded_func():
        # Again, this doen't affect func's local1 variable.  It creates a
        # new local variable also called local1 instead.
        local1 = 5
        print local1

    embedded_func() # Prints 5
    print local1    # Prints 4
```

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Scoping rule (8)

In order to actually modify the bindings of global variables
from within a function scope, you need to specify
that the variable is global with the global keyword. Eg:

```
global_var = 4

def change_global():
    global global_var
    global_var = global_var + 1
```

Currently there is no way to do the same
for variables in enclosing function scopes,

but Python 3 introduces a new keyword, "**nonlocal**"
which will act in a similar way to global, but for nested function scopes.

https://stackoverflow.com/questions/291978/short-description-of-the-scoping-rules

Non-local (1)

Definition and Usage

The nonlocal keyword is used to work with variables inside nested functions, where the variable should not belong to the inner function.

Use the keyword nonlocal to declare that the variable is not local.

https://www.w3schools.com/python/ref_keyword_nonlocal.asp

Non-local (2)

Make a function inside a function, which uses the variable x as a non local variable:

```
def myfunc1():
    x = "John"
    def myfunc2():
        nonlocal x
        x = "hello"
    myfunc2()
    return x

print(myfunc1())
```

https://www.w3schools.com/python/ref_keyword_nonlocal.asp

Non-local (2)

Same example as above, but without the nonlocal keyword:

```
def myfunc1():
    x = "John"
    def myfunc2():
        x = "hello"
    myfunc2()
    return x

print(myfunc1())
```

https://www.w3schools.com/python/ref_keyword_nonlocal.asp