NONLOCAL SCOPES

Inner Functions

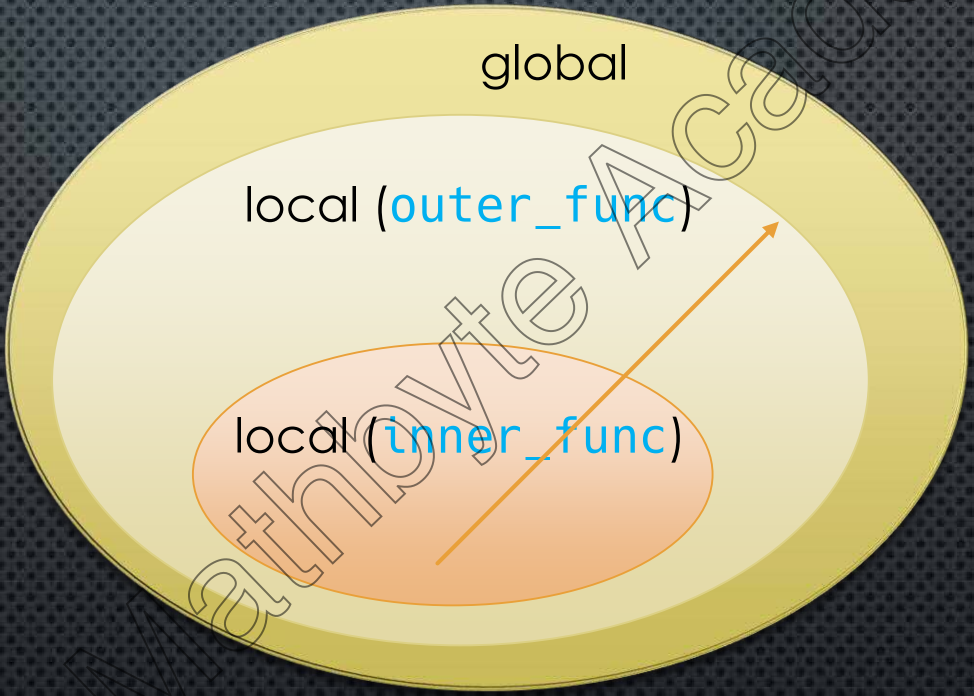
We can define functions from inside another function:

```
def outer_func():
    # some code

    def inner_func():
        # some code

    inner_func()


outer_func()
```



Nested local scopes

Both functions have access to the global and built-in scopes as well as their respective local scopes

But the inner function also has access to its enclosing scope – the scope of the outer function

That scope is neither local (to `inner_func`) nor global – it is called a nonlocal scope

Referencing variables from the enclosing scope

Consider this example we have seen before:

```
module1.py
a = 10

def outer_func():
    print(a)

outer_func()
```

When we call outer_func, Python sees the reference to a

Since a is not in the local scope, Python looks in the enclosing (global) scope

Referencing variables from the enclosing scope

Now consider this example:

```
module1.py

def outer_func():
    a = 10

    def inner_func():
        print(a)

    inner_func()

outer_func()
```

When we call `outer_func`, `inner_func` is created and called

When `inner_func` is called, Python does not find `a` in the local (`inner_func`) scope

So it looks for it in the enclosing scope, in this case the scope of `outer_func`

Referencing variables from the enclosing scope

module1.py

```
a = 10

def outer_func():

    def inner_func():
        print(a)

    inner_func()

outer_func()
```

When we call `outer_func`, `inner_func` is defined and called

When `inner_func` is called, Python does not find `a` in the local (inner_func) scope

So it looks for it in the enclosing scope, in this case the scope of `outer_func`

Since it does not find it there either, it looks in the enclosing (global) scope

Modifying global variables

We saw how to use the `global` keyword in order to modify a global variable within a nested scope

```
a = 10

def outer_func1():
    global a
    a = 1000

outer_func1()
print(a)            → 1000
```

We can of course do the same thing from within a nested function

```
def outer_func2():
    def inner_func():
        global a
        a = 'hello'
    inner_func()

outer_func2()
print(a)            → hello
```

Modifying nonlocal variables

Can we modify variables defined in the outer nonlocal scope?

```
def outer_func():
    x = 'hello'

    def inner_func():
        x = 'python'

    inner_func()

    print(x)

outer_func()   → hello
```

When `inner_func` is compiled, Python sees an assignment to x

So it determines that x is a local variable to `inner_func`

The variable x in `inner_func` masks the variable x in `outer_func`

Modifying nonlocal variables

Just as with global variables, we have to explicitly tell Python we are modifying a nonlocal variable

We can do that using the `nonlocal` keyword

```
def outer_func():
    x = 'hello'

    def inner_func():
        nonlocal x
        x = 'python'

    inner_func()

    print(x)

outer_func()    → python
```

Nonlocal Variables

Whenever Python is told that a variable is nonlocal

it will look for it in the enclosing local scopes chain until it first encounters the specified variable name

Beware: It will only look in local scopes, it will not look in the global scope

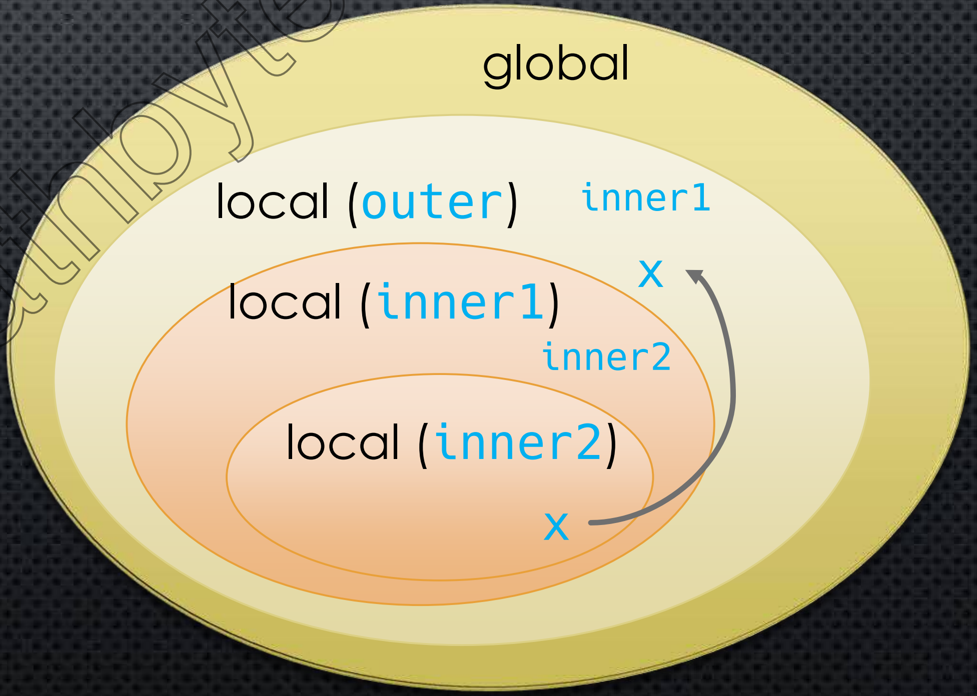
```
def outer():
    x = 'hello'

    def inner1():
        def inner2():
            nonlocal x
            x = 'python'
        inner2()

    inner1()
    print(x)

outer()
```

→ python



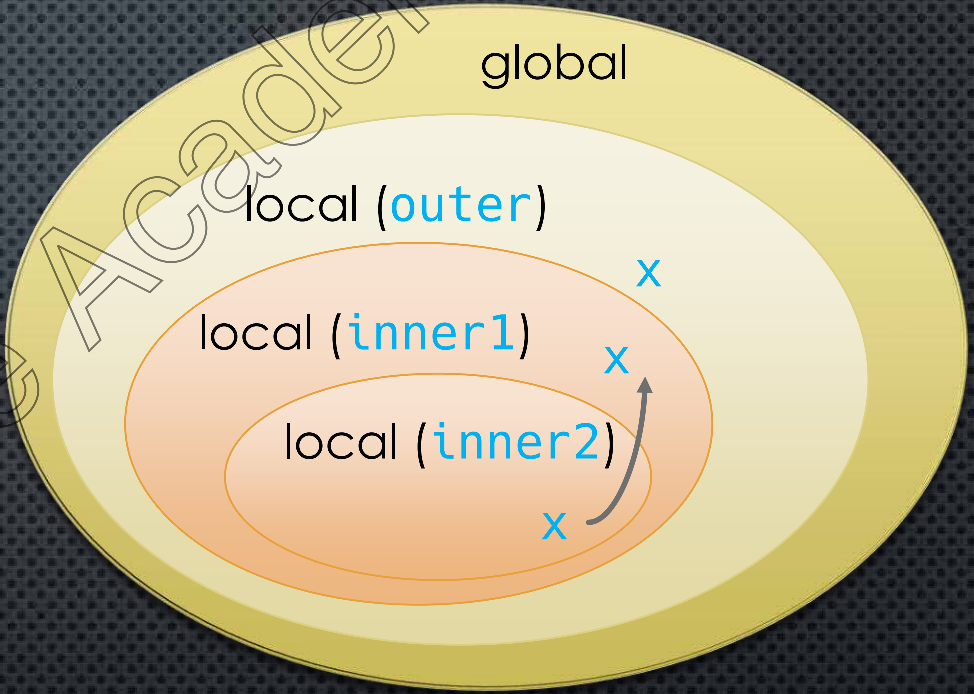
Nonlocal Variables

But consider this example:

```
def outer():
    x = 'hello'

    def inner1():
        x = 'python'
        def inner2():
            nonlocal x
            x = 'monty'
        print('inner(before)', x)     → python
        inner2()
        print('inner(after)', x)      → monty

    inner1()
    print('outer', x)                 → hello
outer()
```



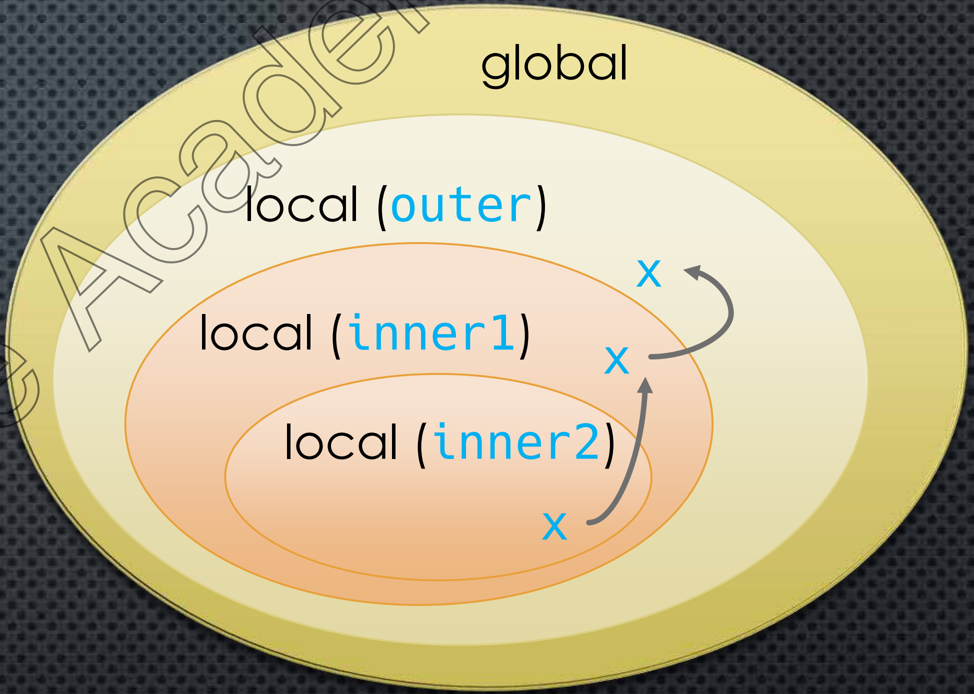
Nonlocal Variables

```
def outer():
    x = 'hello'

    def inner1():
        nonlocal x
        x = 'python'
        def inner2():
            nonlocal x
            x = 'monty'
        print('inner(before)', x)    → python
        inner2()
        print('inner(after)', x)     → monty

    inner1()
    print('outer', x)                → monty


outer()
```



Nonlocal and Global Variables

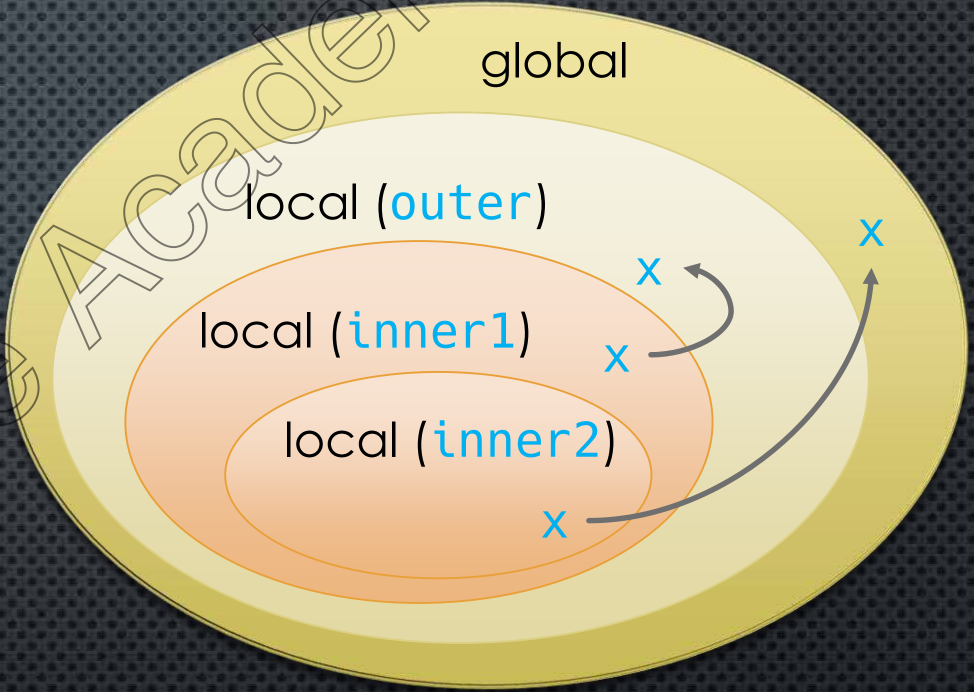
```
x = 100
def outer():
    x = 'python'

    def inner1():
        nonlocal x
        x = 'monty'
        def inner2():
            global x
            x = 'hello'
        print('inner(before)', x)     → monty
        inner2()
        print('inner(after)', x)      → monty

    inner1()
    print('outer', x)                 → monty

outer()

print(x)                              → hello
```



Code