INTRODUCTION TO PROGRAMMING USING PYTHON

Outline

- Function
- Default parameters
- Unpacking & packing parameters
- Recursion
- Scope
- Lambda
- Error handling

Functions

- Is reusable block of code
- Run when you call it
- Accept parameters to deal with it
- Can return data after job finished
- Types of functions

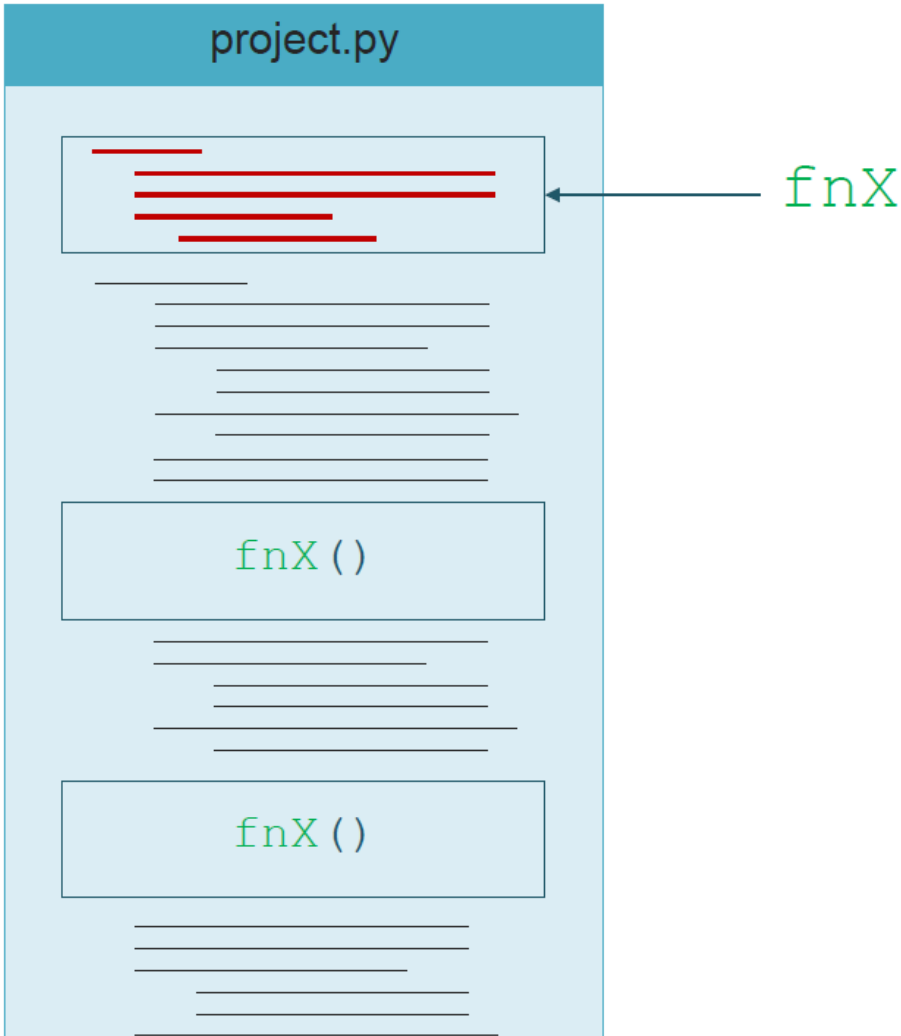
return

Built in

User defined

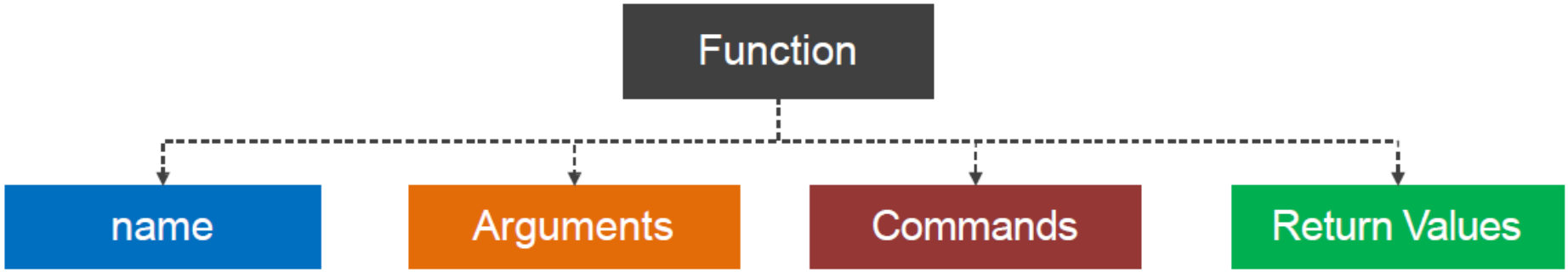
Make your code more generic

Functions



Function (Defining)

```
def fnName:
    pass
```




```
def measureTemp ( temp ):
    if temp < 37:
        return "Too Cold"
    elif temp > 37:
        return "Too Hot"
    return "Normal"
```

Defining example

Calling

```
measureTemp(37)
# "Normal"
```

Function (Default Arguments)

```
def doSum(x, y = 2,z = 3):
    sum = x + y + z
    print(sum)
```

Calling It

```
doSum(2)            # output: 7
doSum(2,4)          # output: 9
doSum(2,4,10)       # output: 16
```

Function (*arguments)

```
def doSum(*args):
    sum = 0
    for i in args:
        sum += i;
    print(sum)
```

Calling It

```
doSum(2,6)          # output: 8
doSum(2,4,5,15)     # output: 26
```

Function (**keywords)

```
def doSum(**kwargs):
    for k in kwargs:
        print(kwargs[k])
```

Calling It

```
doSum(x = 2, y = 26)     # output: 2
                                  26
```

Recursion

- function calls itself in order to solve a problem

2 usages

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

num = 5
result = factorial(num)
print(f"The factorial of {num} is {result}")
```

calls itself

```
The factorial of 5 is 120
```

Scope

To know your limits

Lexical Scope

Output:

Global Scope

Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
```

Output:

Global Scope

```
name = "Ahmed"
```

Lexical Scope

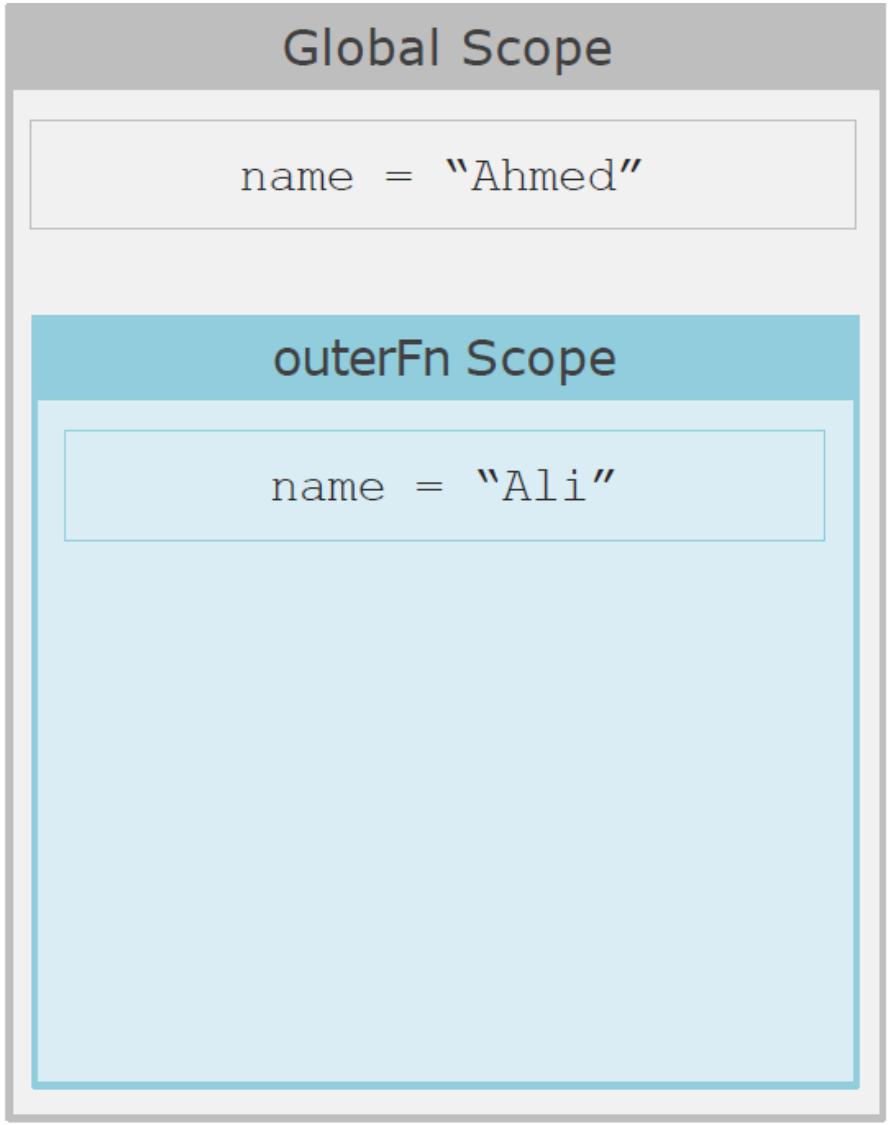
```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

Output:

Lexical Scope

```
name = "Ahmed"
def outerFn():
    → name = "Ali"
    def innerFn():
        print(name)
    innerFn()
outerFn()
```

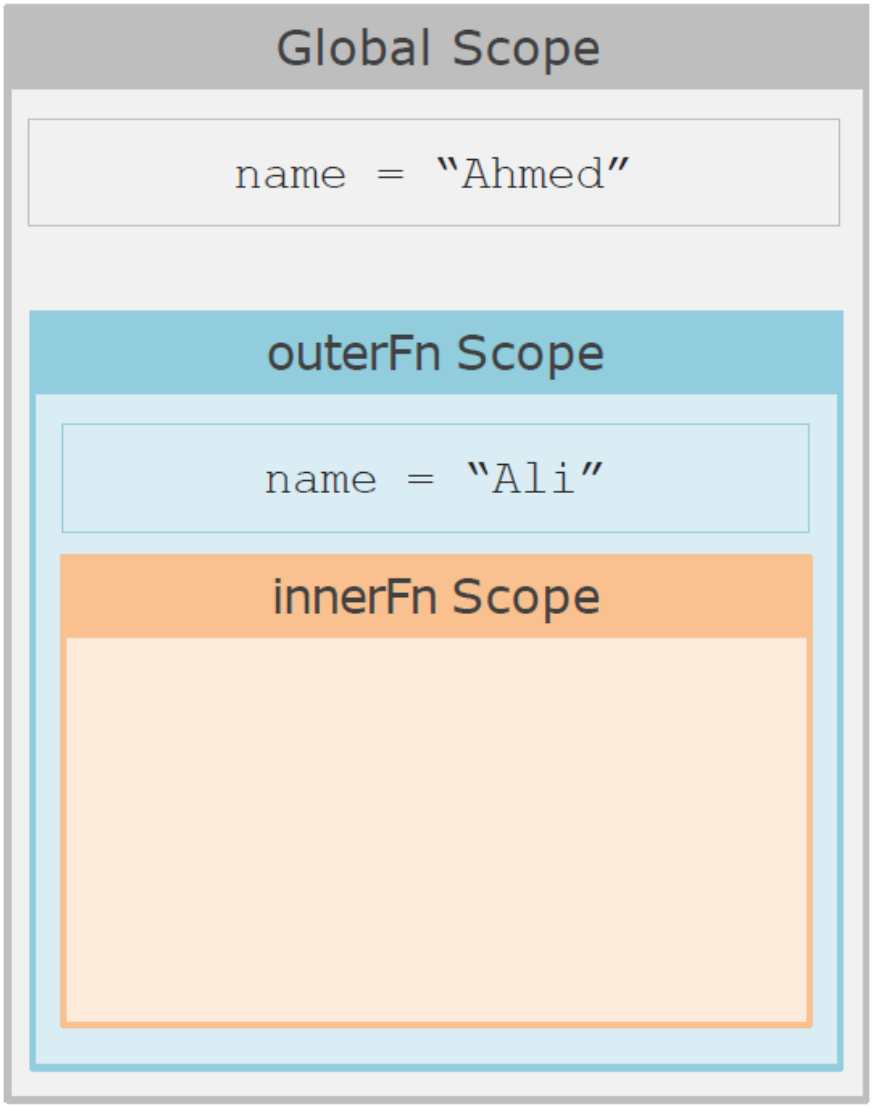
Output:



Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
→               print(name)
        innerFn()
outerFn()
```

Output:

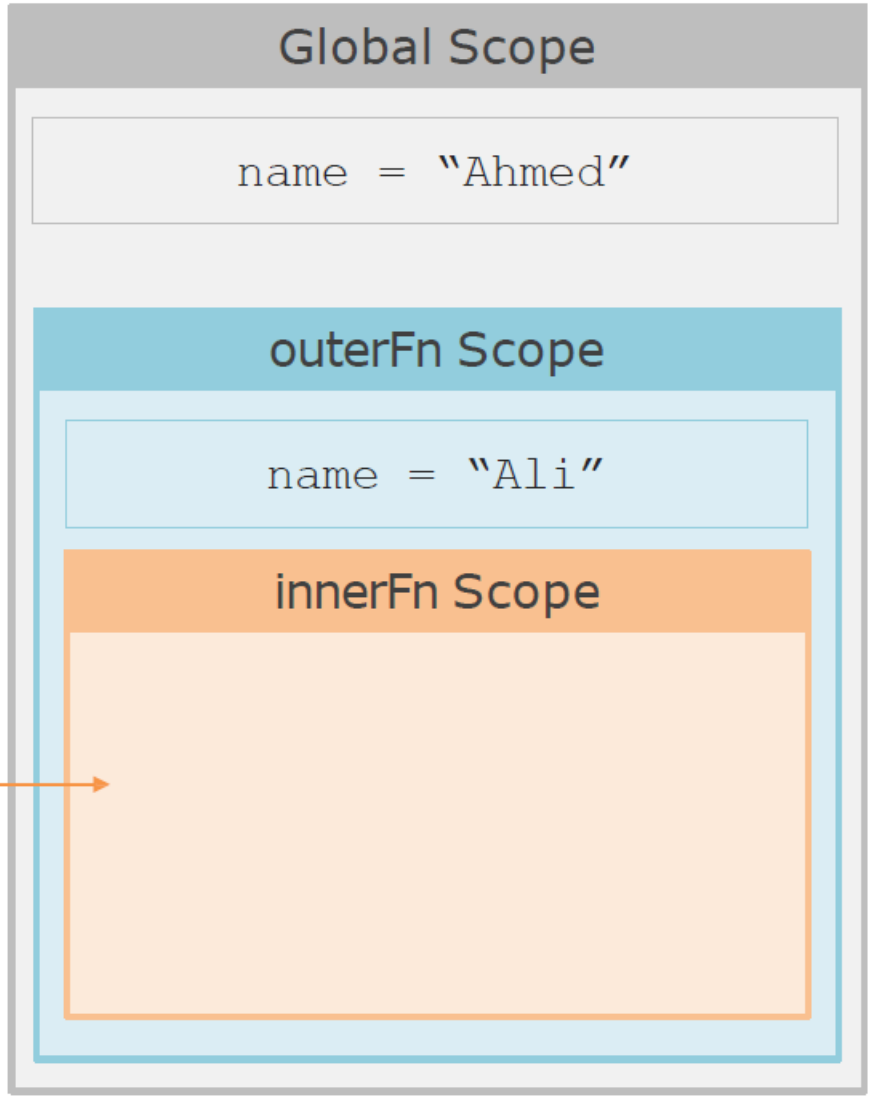


Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
→               print(name)
        innerFn()
outerFn()
```

```
Output:
```

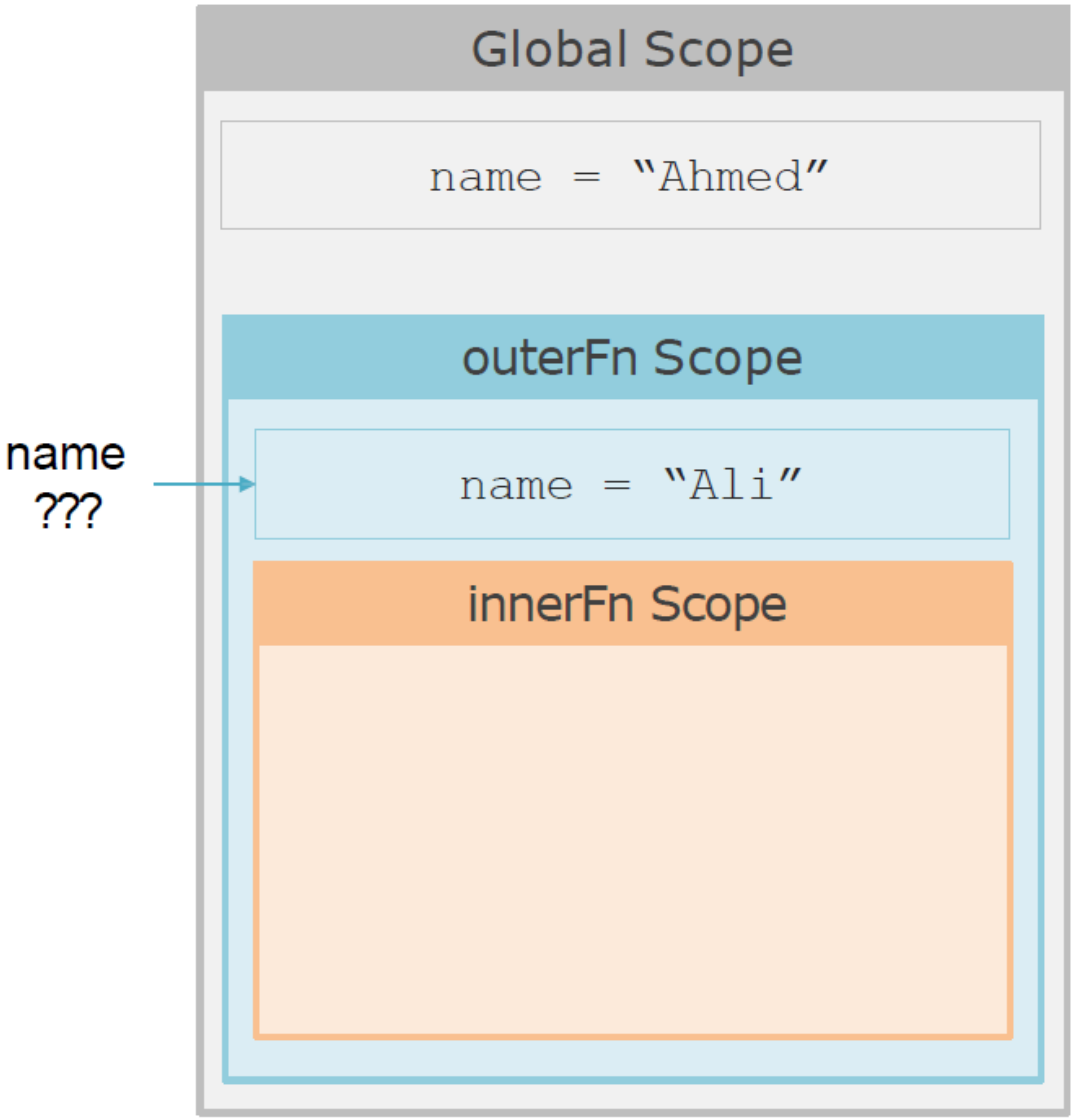
name
???



Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
→               print(name)
        innerFn()
outerFn()
```

Output:



Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
print(name)
```

Output:

```
Ali
```

Global Scope

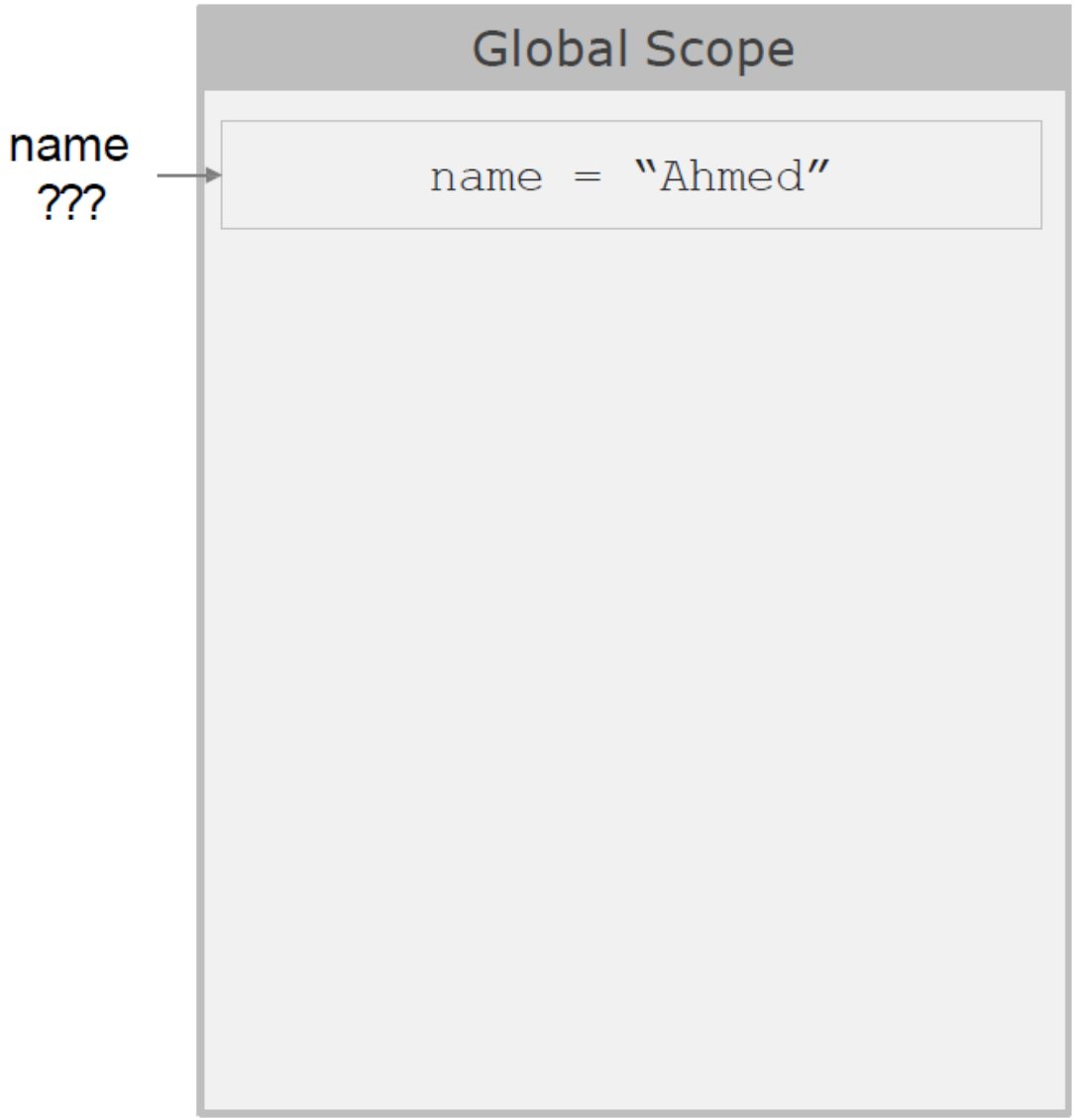
```
name = "Ahmed"
```

Lexical Scope

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
print(name)
```

Output:

```
Ali
Ahmed
```



global Keyword

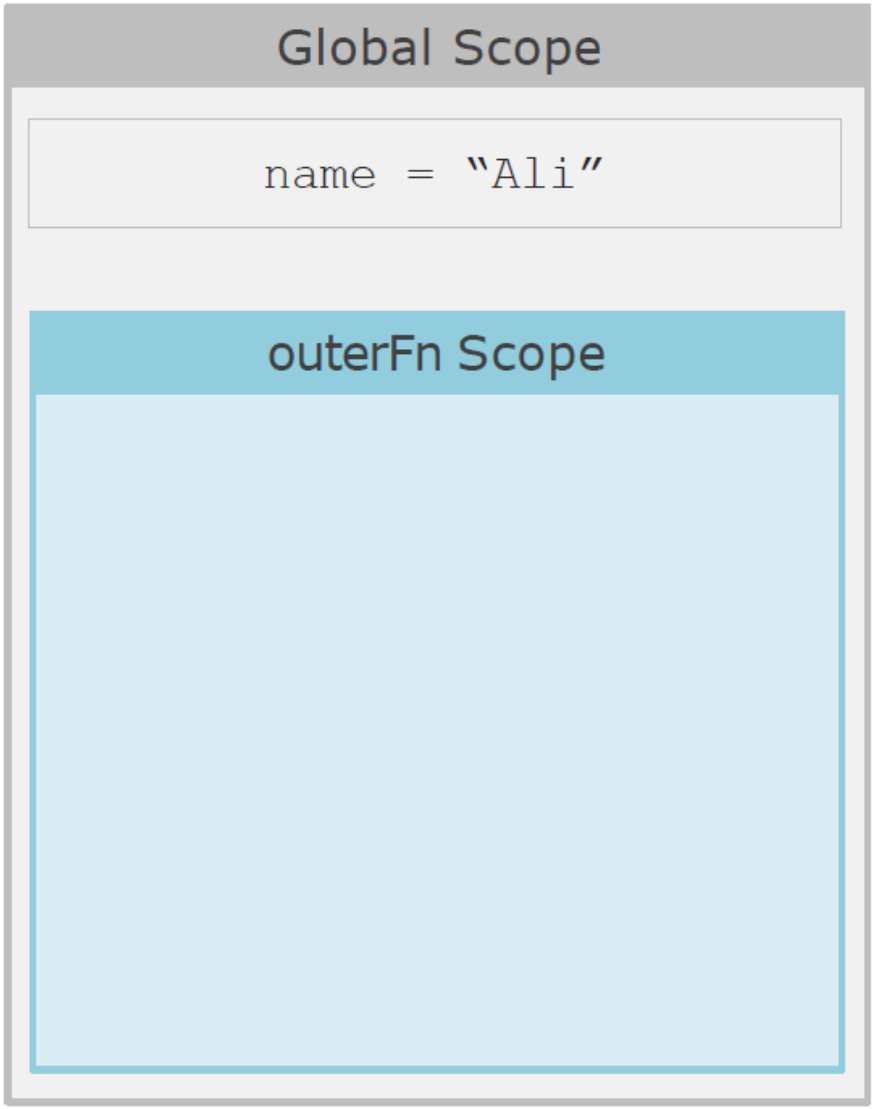
```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

Output:

global Keyword

```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

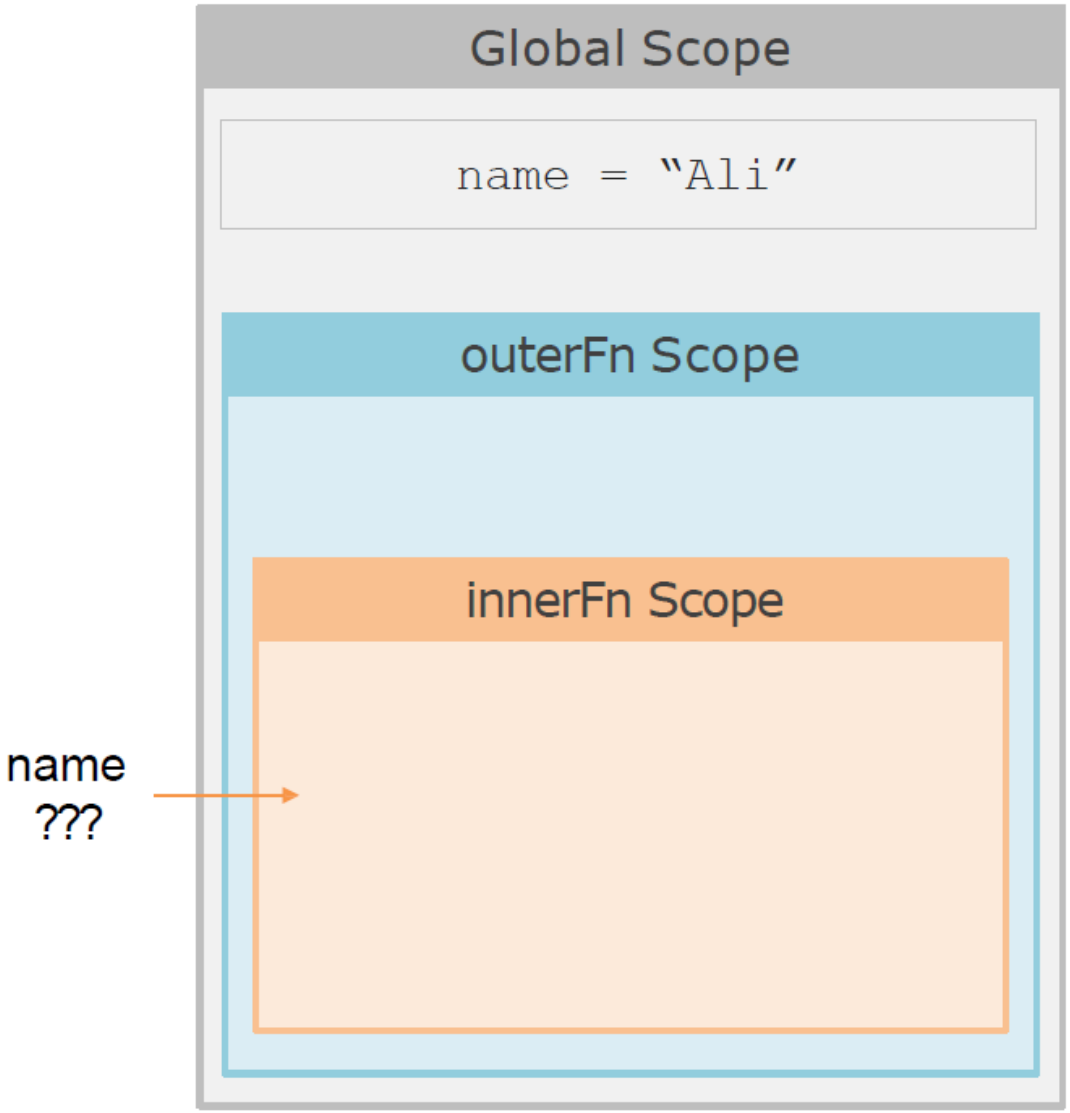
Output:



global Keyword

```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

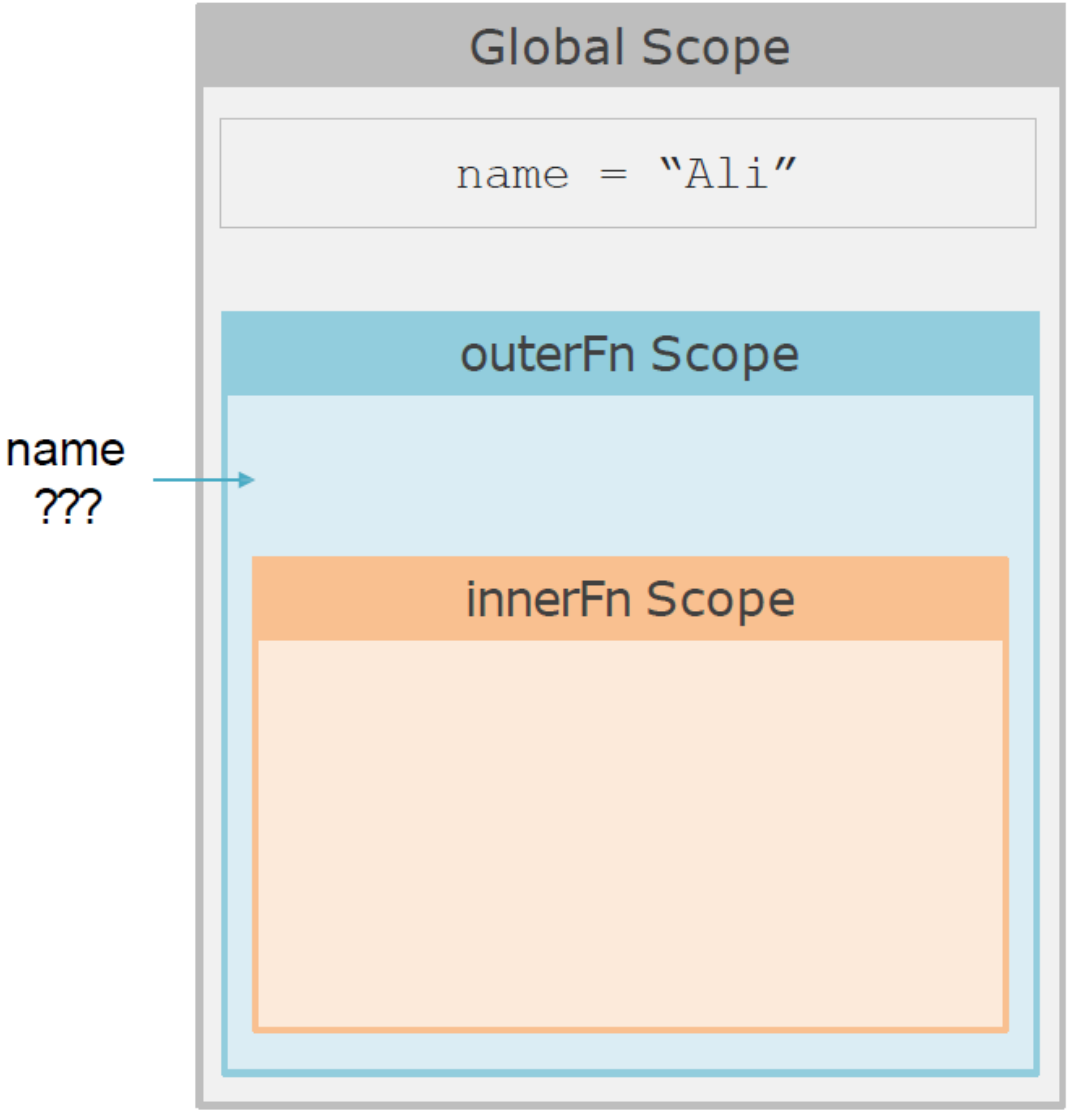
Output:



global Keyword

```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

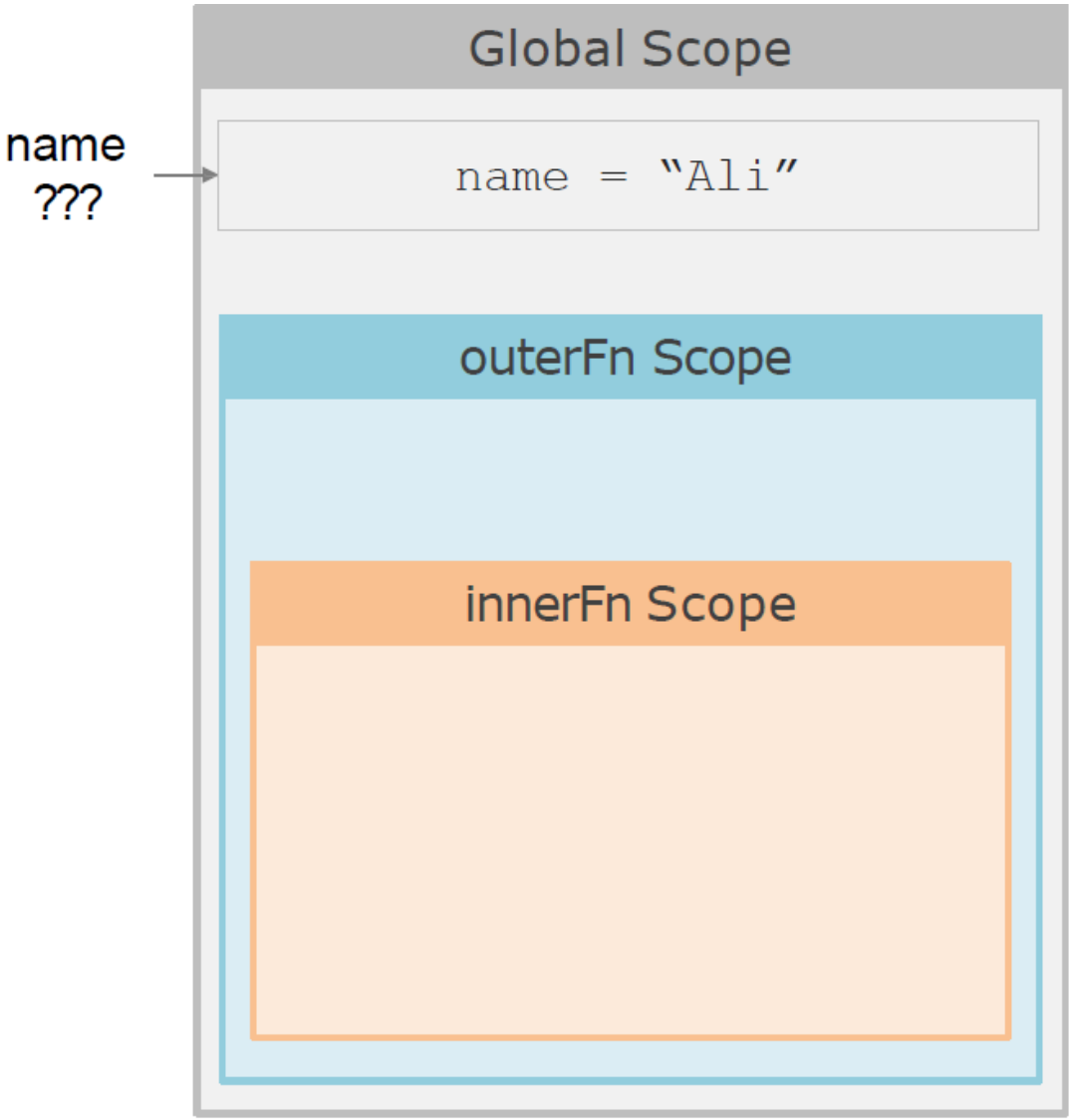
Output:



global Keyword

```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
```

Output:

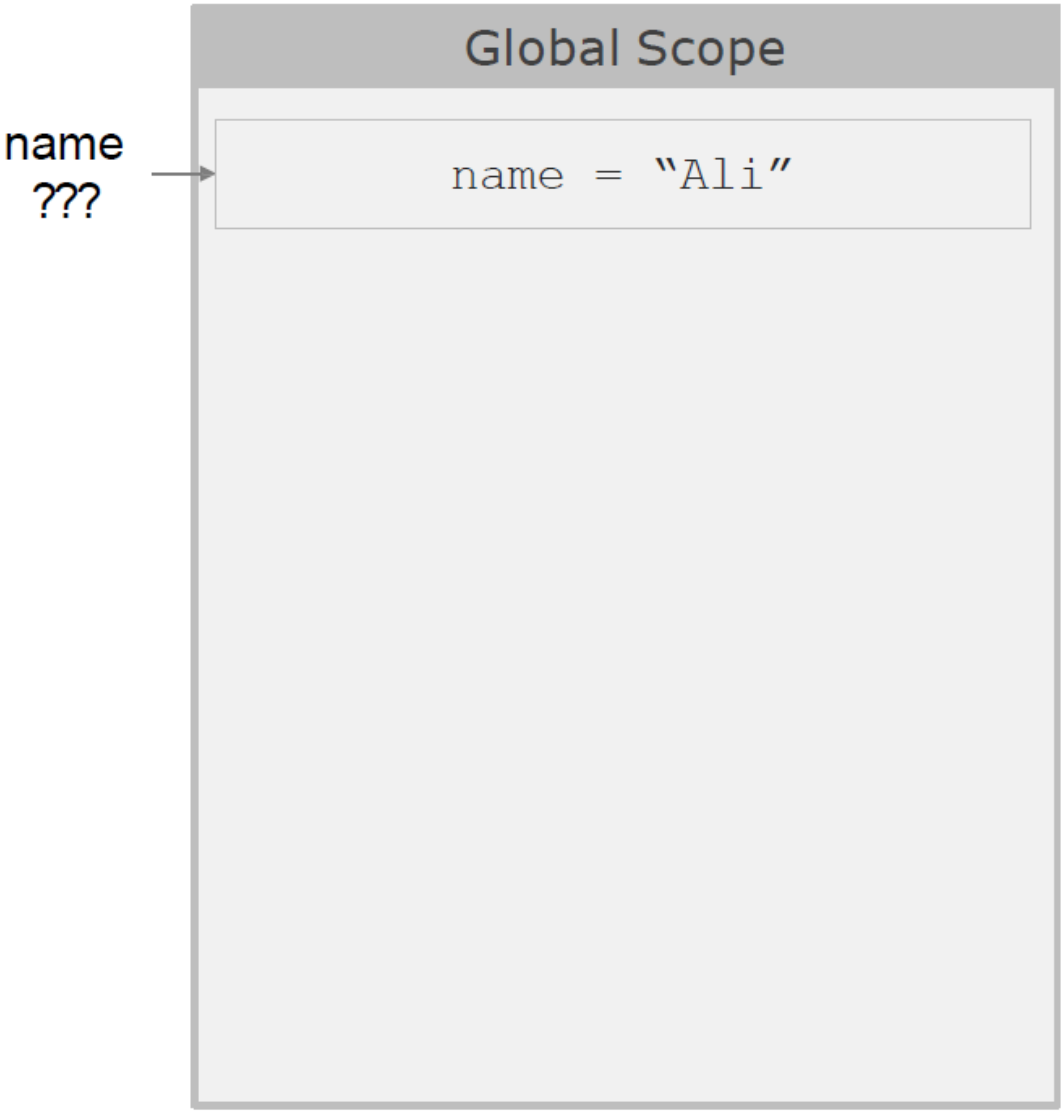


global Keyword

```
name = "Ahmed"
def outerFn():
        global name
        name = "Ali"
        def innerFn():
                print(name)
        innerFn()
outerFn()
print(name)
```

Output:

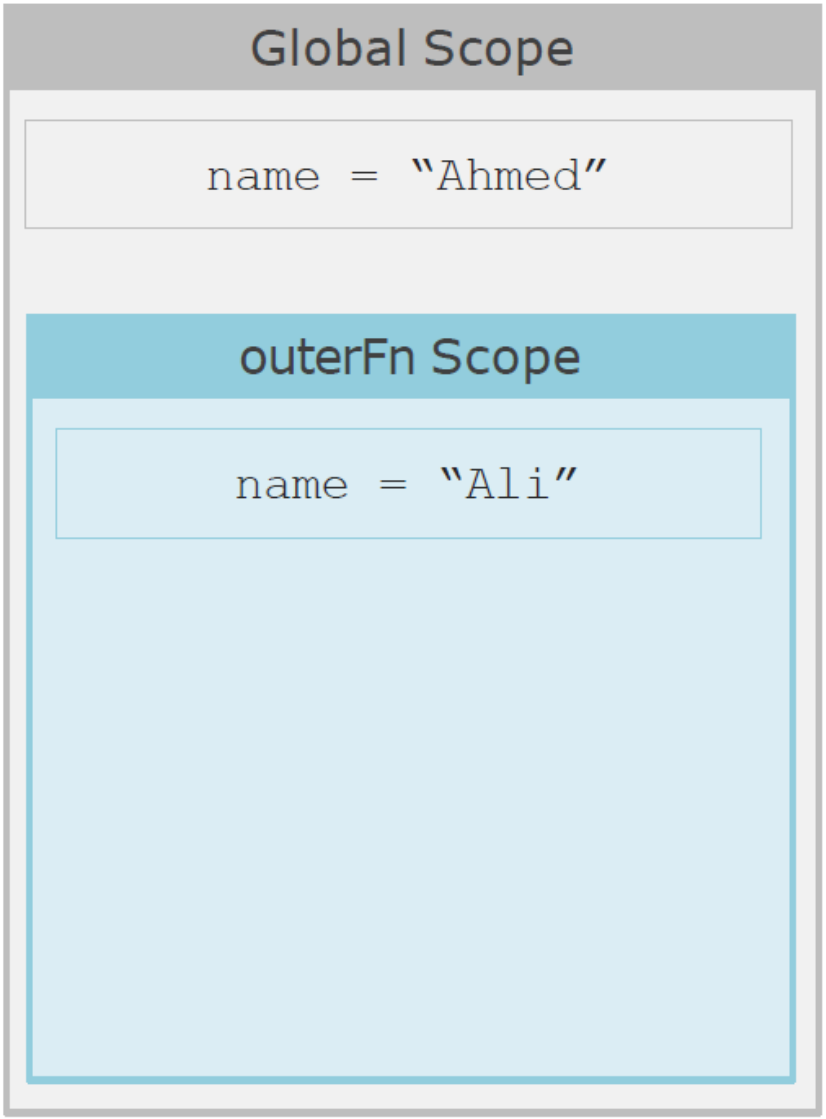
```
Ali
Ali
```



nonlocal Keyword

```
name = "Ahmed"
def outerFn():
    name = "Ali"
    def innerFn():
        nonlocal name
        print(name)
        name = "Sara"
    innerFn()
    print(name)
outerFn()
```

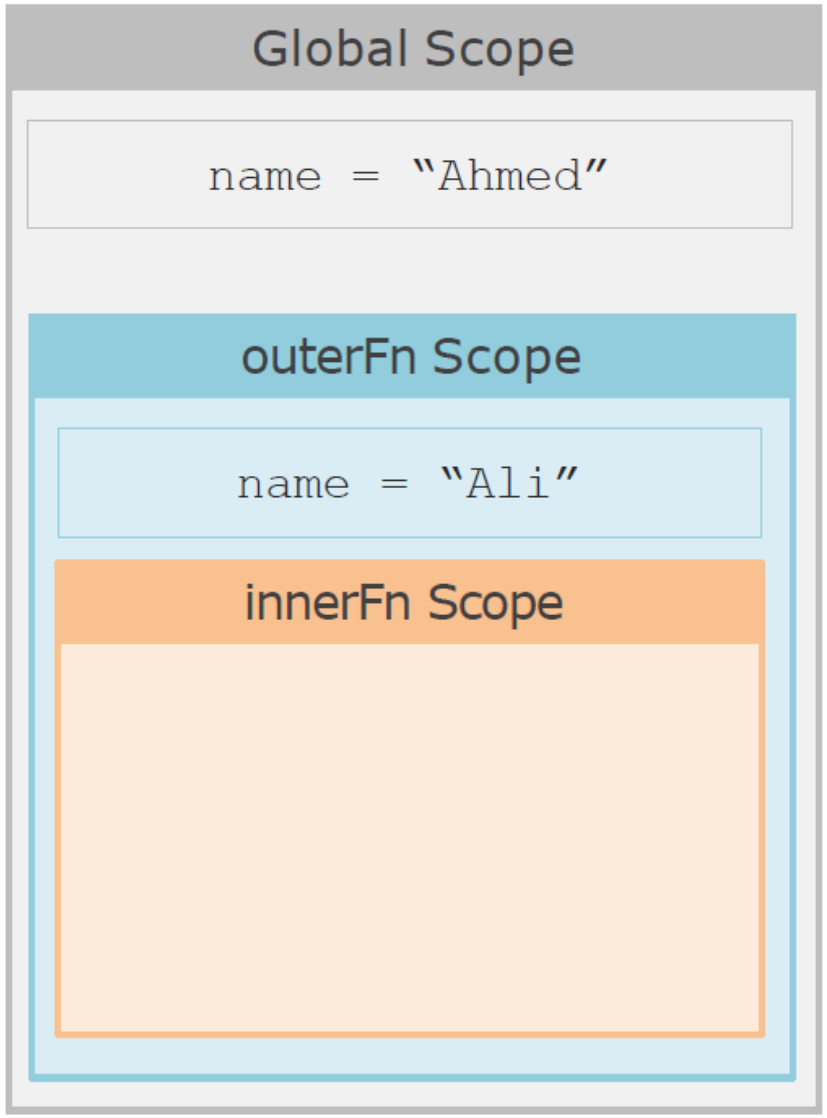
Output:



nonlocal Keyword

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
            nonlocal name
            print(name)
            name = "Sara"
    →   innerFn()
        print(name)
outerFn()
```

Output:

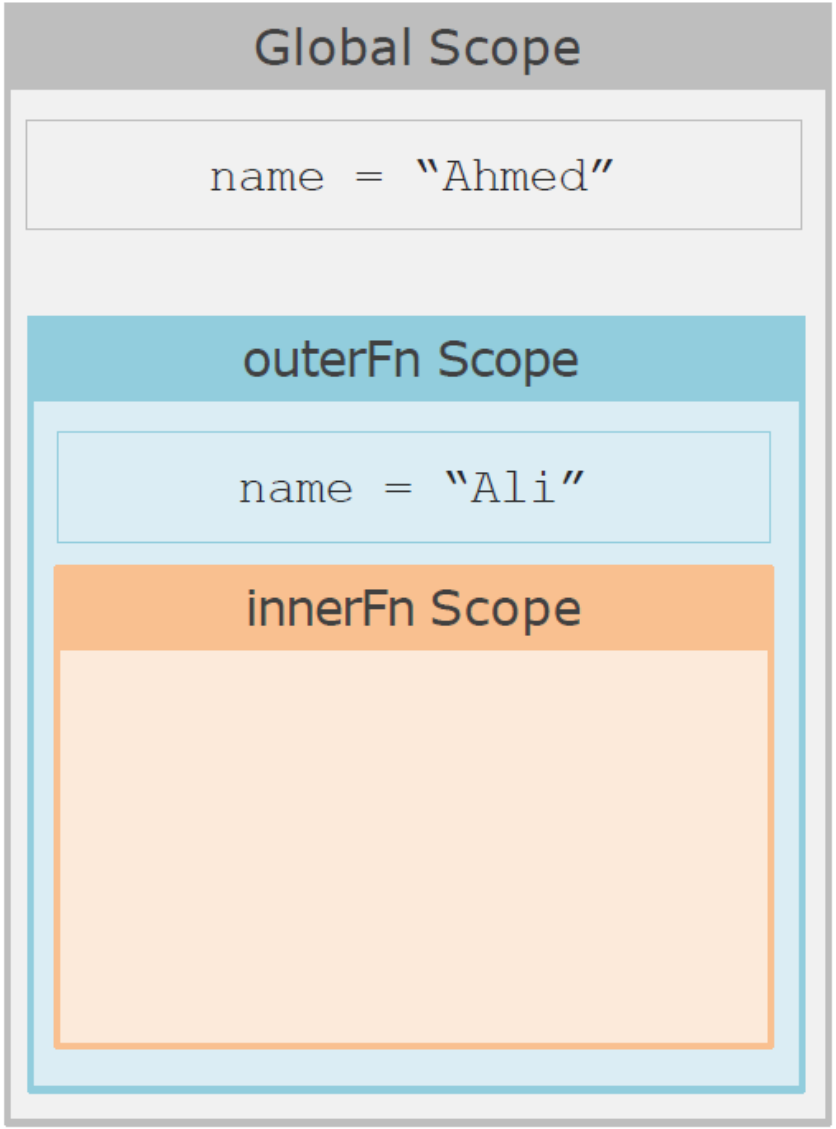


nonlocal Keyword

```
name = "Ahmed"
def outerFn():
    name = "Ali"
    def innerFn():
        nonlocal name
        print(name)
→       name = "Sara"
    innerFn()
    print(name)
outerFn()
```

Output:

```
Ali
```

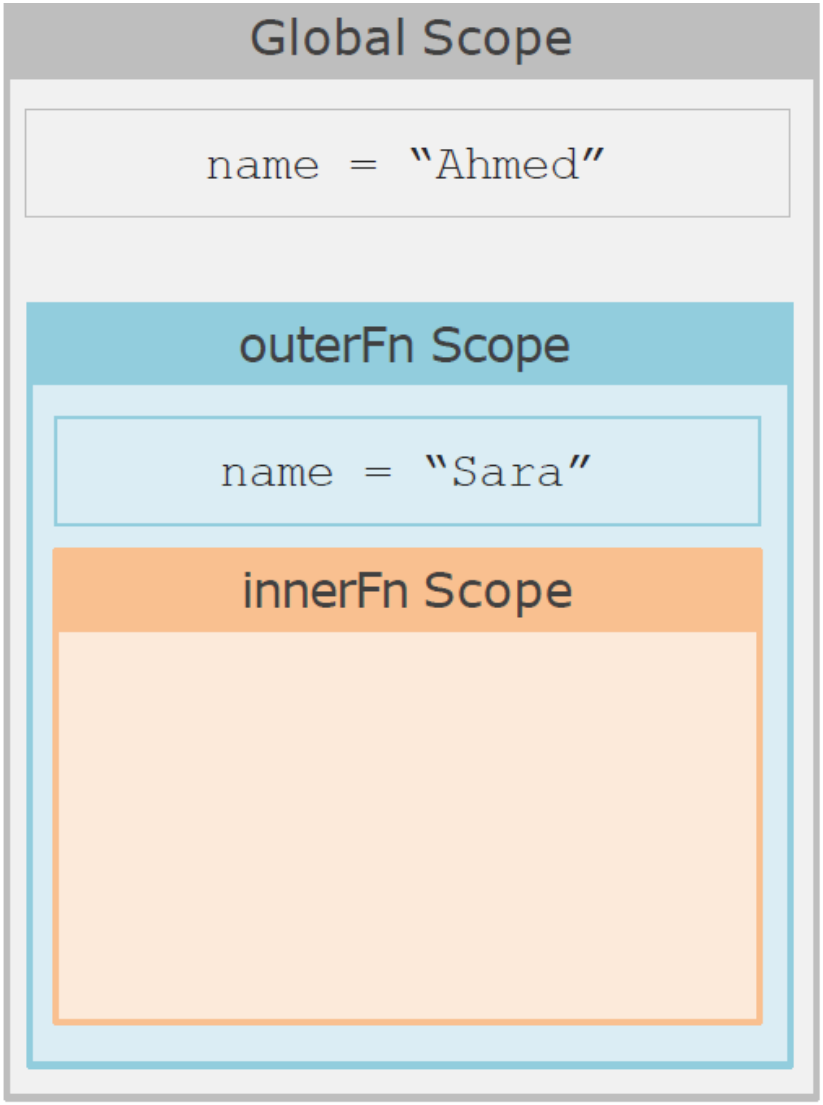


nonlocal Keyword

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
            nonlocal name
            print(name)
    →       name = "Sara"
        innerFn()
        print(name)
outerFn()
```

Output:

```
Ali
```

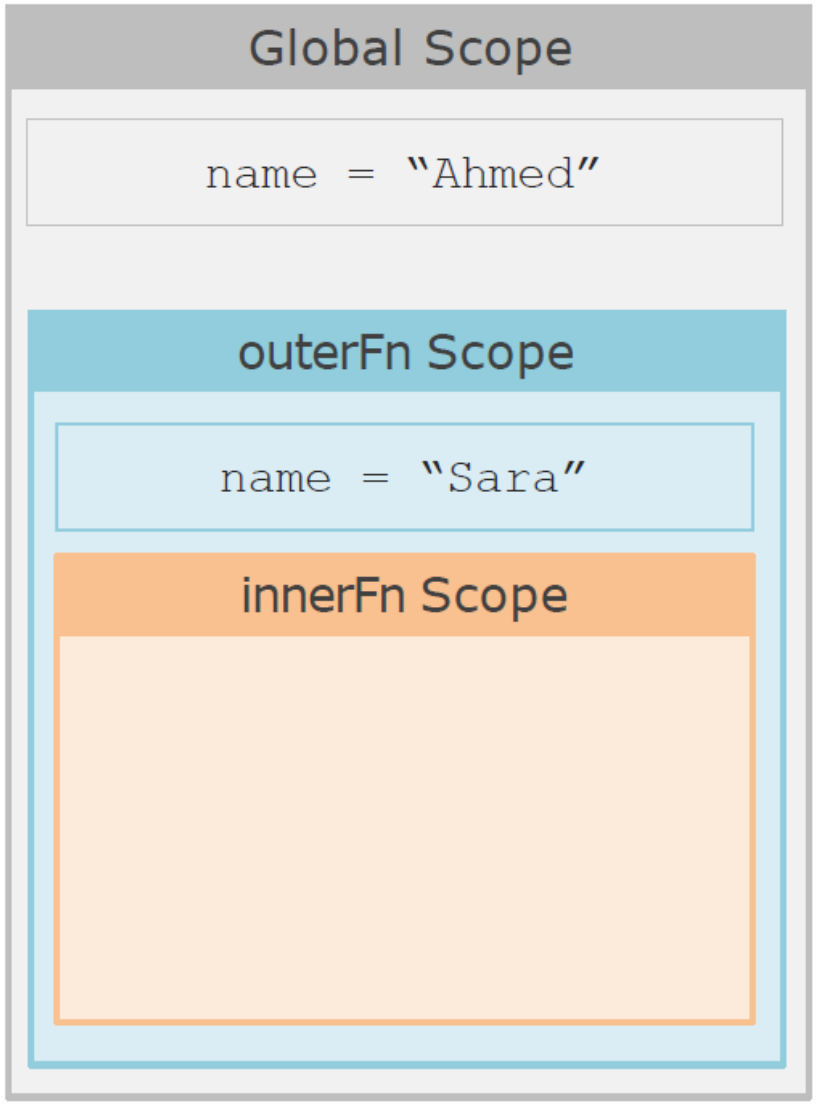


nonlocal Keyword

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
            nonlocal name
            print(name)
            name = "Sara"
        innerFn()
    →   print(name)
outerFn()
```

Output:

```
Ali
```

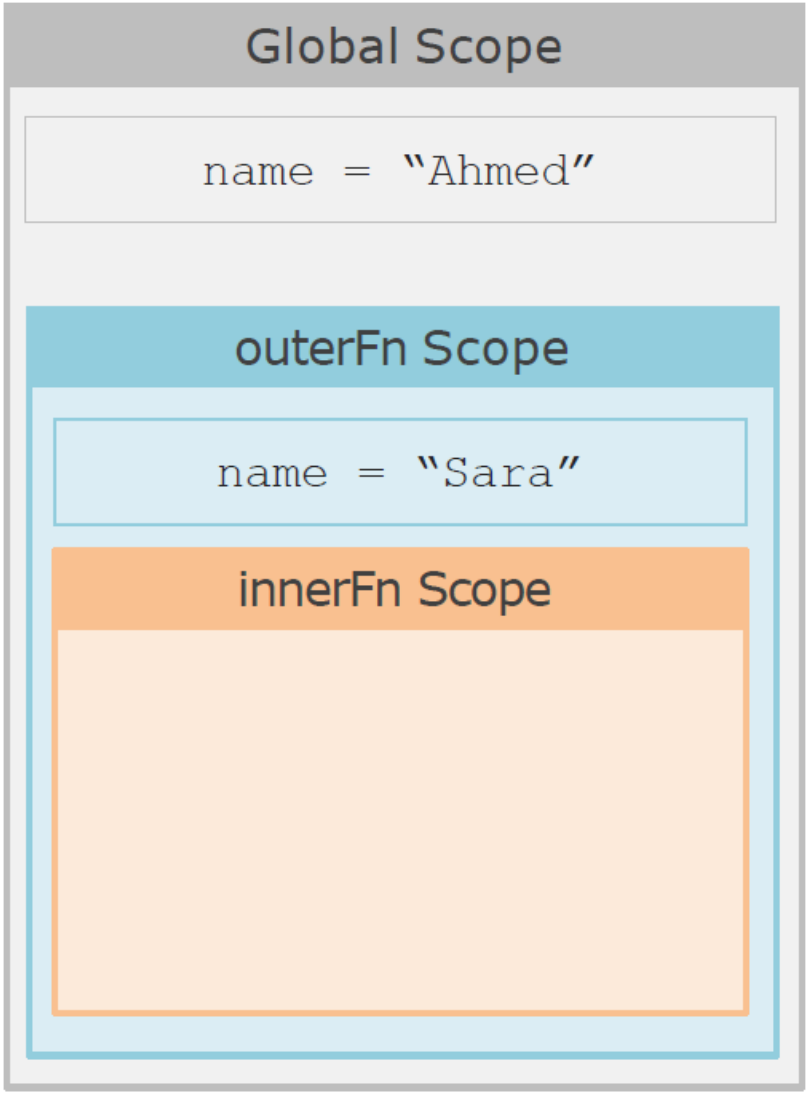


nonlocal Keyword

```
name = "Ahmed"
def outerFn():
        name = "Ali"
        def innerFn():
            nonlocal name
            print(name)
            name = "Sara"
        innerFn()
    →   print(name)
outerFn()
```

Output:

```
Ali
Sara
```



Lambda

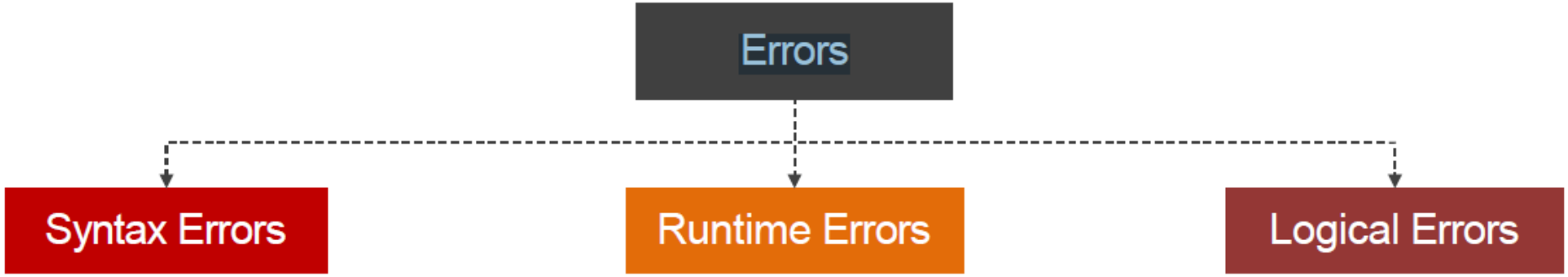
- small **anonymous** function that can have any number of arguments, but can only have one expression.
- used for simple tasks

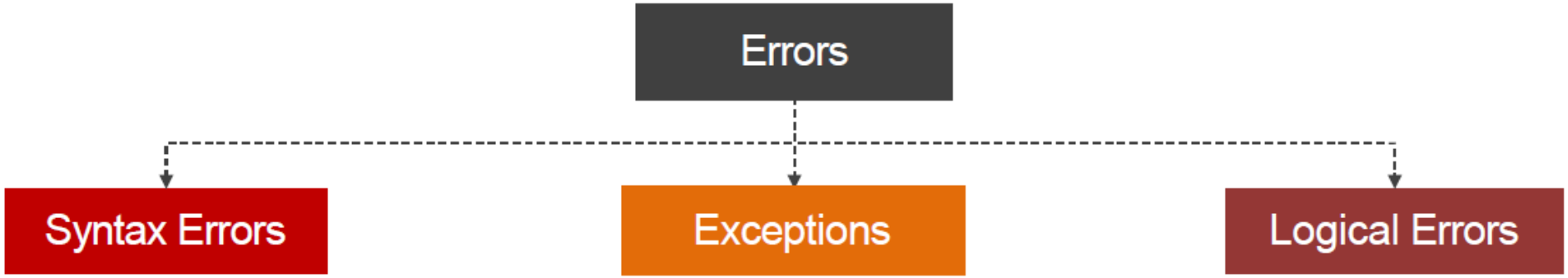
lambda arguments: expression

```
add = lambda x, y: x + y
print(add( x: 5,  y: 3))  # Output: 8

square = lambda x: x ** 2
print(square(4))  # Output: 16
```

Error handling





Error handling

Syntax Errors

Errors that will show up if you doesn't follow Python Syntax Rules

```
print("You missed the closing round braces "
```

```
print("You missed the closing round braces "
                                            ^
SyntaxError: invalid syntax
```

Error handling

Exceptions

Errors detected during execution are called **Exceptions**

```
print(firstname);
```

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

Error handling

Handling Exceptions

```
try:
    doTry()
except:
    doExcept()
finally:
    doFinally()
```

Put the code that you want to handle its exceptions

Handle the exception if it raised in the try clause

Put the code that you want to run always if there is an exception or not.

Error handling

Handling Exceptions

```
try:
        for i in range(3):
            print(3/i)
except  ValueError:
        print(“Value Error")
except  ZeroDivisionError:
        print("you divided by 0")
finally:
        print("this will print no matter what")
```

Exercises