Python

The Easy Way

Ahmed Moawad

Course Objectives

Learn about Python, its uses and really understand it.

Python Inventor

guido van rossum

Why Python

Easy To learn

Rapid Development

General Purpose Language

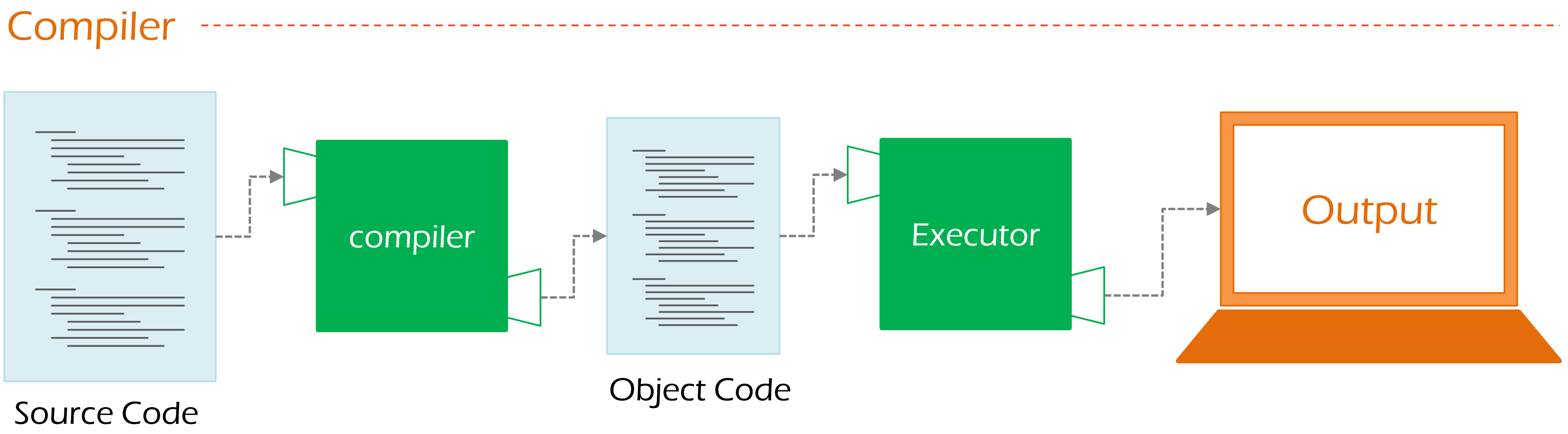
Python 2 or 3

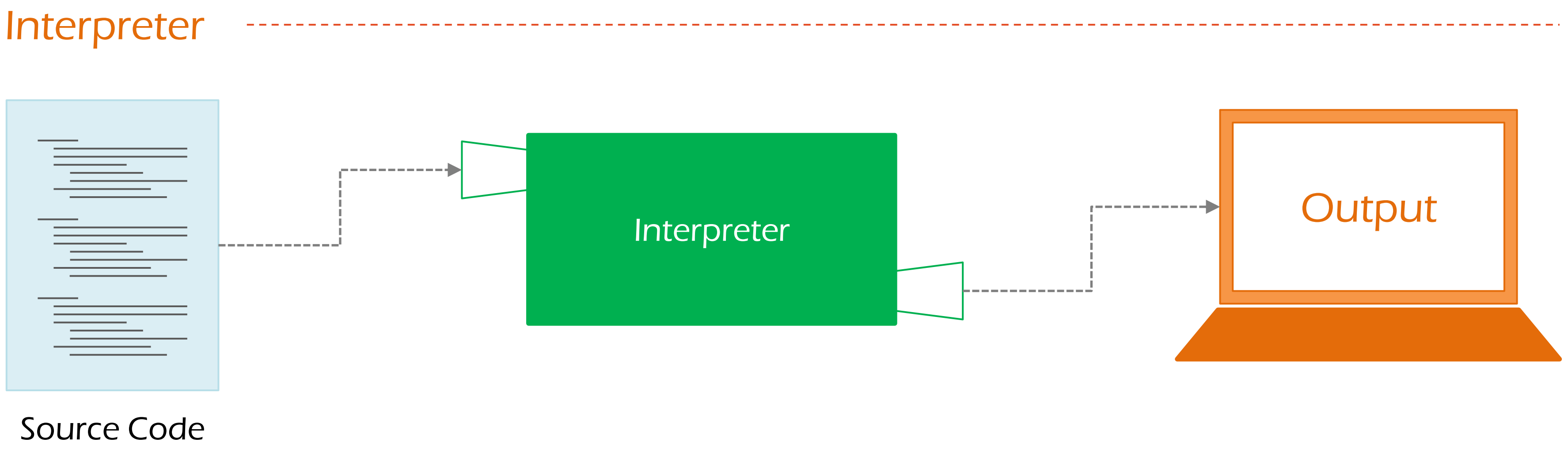
Python 2 is the legacy, Python 3 is the future of the language

How does python work?

HPW

Compiler vs Interpreter





Python is Interpreted Language

Hello World Program

```
print(“Hello, World”)
```

Syntax

Python Syntax Rules

Identifiers Rules

A Python **identifier** is a name used to identify a variable, function, class, module or other object

Starts only with:	a → z	A → Z	_	
Can't Contain :	Punctuations Characters			
Can Contain:	digits	a → z	A → Z	_

Python is Case Sensitive Language

Reserved Words

A Python **identifier** doesn’t be one of these words

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Line Indentations

Level 1

Level 2

```
if True:
    print(“Hello, World”)
else:
    print(“Bye, World”)
```

No ;

Just Line Indentation

Quotes … 1.. 2 … 3

' ' , " " , ''' ''' & """ """

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""
```

Comments

```
# this is a comment
```

Variables & Data Types

Python is loosely typed language

Declare a Variable

```
Variable Identifier = Variable Value
```

```
name = “Ahmed”

age = 17

isStudent = True

age = “seventeen”
```

Data Types

- Strings
- Numbers
- Boolean
- Tuples
- Lists
- Dictionaries

```
type(variable_name)
```

Type Conversion

```
age = 17.5
int(age)                # 17
float(age)              # 17.5
str(age)                # “17.5”
```

Operators

Arithmetic

`+`	addition Op	`2 + 3`	`#output: 5`
`-`	Subtraction Op	`4 - 2`	`#output: 2`
`*`	Multiplication Op	`4 * 5`	`#output: 20`
`/`	Division Op	`16 / 5`	`#output: 3.2`
`%`	Modulus Op	`16 % 5`	`#output: 1`
`//`	Division without Fractions	`16 // 5`	`#output: 3`
`**`	Exponent Op	`2 ** 4`	`#output: 16`

Assignment

`=`	assign	`x = 4`	`#output: 4`
`+=`	add and assign	`x += 3`	`#output: 7`
`-=`	subtract and assign	`x -= 2`	`#output: 5`
`*=`	multiply and assign	`x *= 6`	`#output: 30`
`/=`	divide and assign	`x /= 2`	`#output: 15`
`%=`	get modulus and assign	`x %= 8`	`#output: 7`
`//=`	floor divide and assign	`x //= 3`	`#output: 2`
`**=`	get exponent and assign	`x **= 4`	`#output: 16`

Comparison

```
a <op> b
```

==	return True if a equals b
>=	return True if a equals or greater than b
<=	return True if a equals or lesser than b
!=	return True if a not equals b
<>	return True if a not equals b
>	return True if a greater than b
<	return True if a lesser than b

== Examples

When using == Python assume that:

```
True = 1, False = 0
```

```
2 == "2"                     #output: False

True == "True"               #output: False

False == 0                   #output: True

True == 1                    #output: True

True == 2                    #output: False
```

Boolean Operators

Expression (Logic Gate) Expression

Logic Gates

`and`	AND Logic Gate
`or`	OR Logic Gate
`not`	Not Logic Gate

```
True and False                          #output: False

True or False                           #output: True

not False                               #output: True

not (True == 2)                         #output: True

(False == 0) and (True == 1)            #output: True
```

Falsy Values

None, False, 0 , Empty collections: “ ”, ( ), [ ], { }

More Examples

```
2 and 1                                 #output: 1

2 or 1                                  #output: 2

not 4                                   #output: False

not 0                                   #output: True

2 and 0                                 #output: 0

0 and 2                                 #output: 0

“Google” and 1                          #output: 1

“” and “Go”                             #output: “”

False or 0                              #output: 0
```

Strings

Play with Strings

How To

```
name = “Ahmed”
```

—or—

```
name = ‘Ali’
```

Play !

```
name = “Ahmed ”

print(name) # Ahmed

fullName = “Mohamed ” + name * 3 + “ Ali”;

print(fullName) # Mohamed Ahmed Ahmed Ahmed Ali

nameIntro = ( “I’m ” fullName );

print(nameIntro) # I’m Mohamed Ahmed Ahmed Ahmed Ali

print(name[4])   # d

print(name[1:3]) # hm

print(name[:4])  # Ahme

print(name[6])   # Index Error
```

Methods

```
name = “information technology institute”

name.capitalize() # Information Technology Institute

len(name) #32

order = “Go read info about his work info in ” + name

order.replace(“info”, “”,2)

# Go read about his work in information technology institute

digits,containDigits = “0102002932”, “Tel0102002932”

digits.isDigit() # True

containDigits.isDigit() # False
```

String Formatting

```
str.format(*args,**kwargs)
```

Example

```
intro = “My Name is {0}”
intro.format(‘Ahmed’)
# My Name is Ahmed

intro = “My Name is {1}, I work at {0}”
intro.format(‘ITI’,‘Ali’)
# My Name is Ali, I work at ITI

intro = “My Name is {name}, I work at {place}”
intro.format(name=‘Ahmed’, place=‘ITI’)
# My Name is Ahmed, I work at ITI
```

Numbers

Play with Numbers

Types

int	18
long *	503340343L
float	18.5
complex	19+4j

* Available in Python 2 only

Type Conversion

int	`int("18")`
long	`long(18.5)`
float	`float(15)`
complex	`complex(4,5)`

Play !

```
w, x, y, z = 4, 4.4, 4.6, 15

round(x)                              #output: 4

round(y)                              #output: 5

min(x,y,z)                            #output: 4.4

max(x,y,z)                            #output: 15
```

Data Structures

lists

Intro

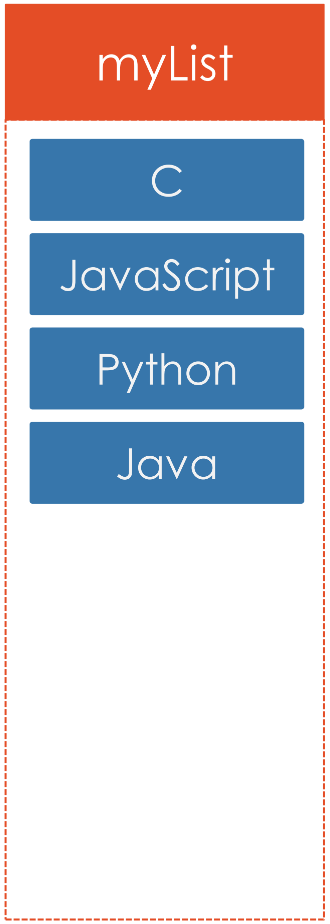
A collection of various data types

```
newList = []
```

```
newList = [1, “hi”, True]
newList[0]   #1
newList[1]   #”Hi”
newList[2]   #True
newList[3]   #Index Error
```

Methods

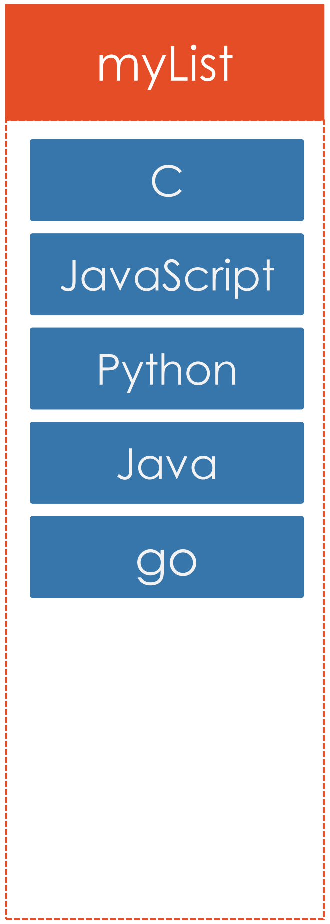
```
myList = ["C", "JavaScript", "Python", "Java", "php"];
```



```
myList.pop(4)
```

Methods

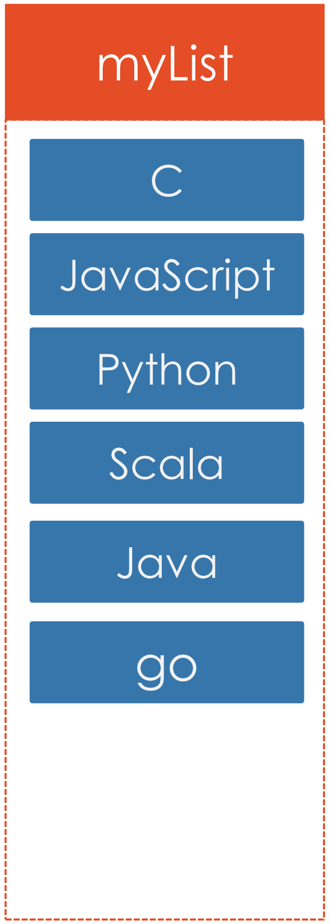
```
myList = [“C”, “JavaScript”, “Python”, “Java”, “php”];
```



```
myList.pop(4)

myList.append(“go”)
```

Methods

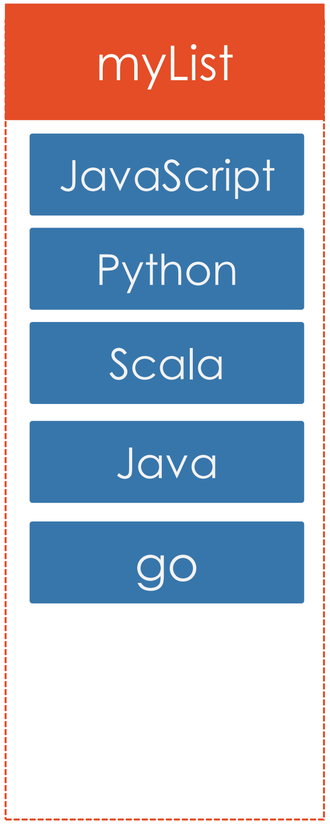
```
myList = ["C", "JavaScript", "Python", "Java", "php"];
```



```
myList.pop(4)

myList.append("go")

myList.insert(3, 'Scala')
```

Methods

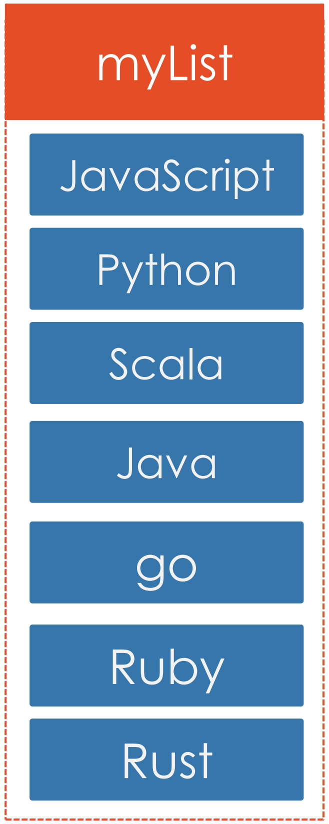
```
myList = ["C", "JavaScript", "Python", "Java", "php"];
```



```
myList.pop(4)

myList.append("go")

myList.insert(3, 'Scala')

myList.remove("C")
```

Methods

```
myList = ["C", "JavaScript", "Python", "Java", "php"];
```



```
myList.pop(4)

myList.append("go")

myList.insert(3, 'Scala')

myList.remove("C")

yourList = ["Ruby", "Rust"];

myList.extend(yourList)
```

Tuples

Immutable Lists

Intro

Same as Lists but Tuples are immutable

```
newTuple = ()
```

```
t = (1, "hi", True)
t[1]
# hi
t[1] = 4
TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Key/value Pairs

Intro

A key: value **comma** seperated elements Data Structure

```
newDict = {}
```

```
d = {"name": "Ahmed", "track": "OS"}
d["name"]
# Ahmed
d["name"] = "Ali"
# {name: "Ali", track: "OS"}
```

Methods

```
infoDict = {'track': 'OS', 'name': 'Ahmed', 'age': 17}

infoDict.keys() # dict_keys(['track', 'name', 'age'])

'name' in infoDict # True

infoDict.items()

# dict_items([('track', 'OS'), ('name', 'Ahmed'), ('age', 17)])

addInfoDict = {'track': 'SD', 'branch': "Smart"}

infoDict.update(addInfoDict)

#{'track': 'SD', 'name': 'Ahmed', 'age': 17 , 'branch': "Smart"}
```

Control Flow

Conditions & Loops

If statement

```
if (x == 2):
    print(“Two”)
elif (x == 3):
    print(“Three”)
else:
    print(“others”)
```

for … in

```
languages = ['JavaScript', 'Python', 'Java']
for l in languages:
      print(l)
```

```
Output:
JavaScript
Python
Java
```

Range Function

```
range([start,] end[, step])
```

Examples

```
range(5)                [0,1,2,3,4]

range(0,5,1)            [0,1,2,3,4]

range(1,10,2)           [1,3,5,7,9]
```

```
for i in range(10):
    print(i)
```

```
0  1  2  3  4  5  6  7  8  9
```

while

```
dayCount = 0
while dayCount < 4:
        print("We are learning Python")
        dayCount += 1
```

```
Output:
We are learning Python
We are learning Python
We are learning Python
We are learning Python
```

DayCount
1
2
3
4

Break Statement

```
for i in range(10):
    if (i == 5):
        break
    print(i)
```

```
0    1    2    3    4
```

Continue Statement

```
for i in range(10):
    if (i == 5):
        continue
    print(i)
```

```
0    1    2    3    4         6    7    8    9
```

Else Statement

```
for i in range(10):
    if (i == 5):
        continue
    print(i)
else:
    print(10)
```

```
0  1  2  3  4     6  7  8  9  10
```

Pass Statement

```
for i in range(10):
    if (i == 5):
        pass
    print(i)
```

```
0  1  2  3  4  5  6  7  8  9
```

input Function

```
input(prompt_message)
```

Example

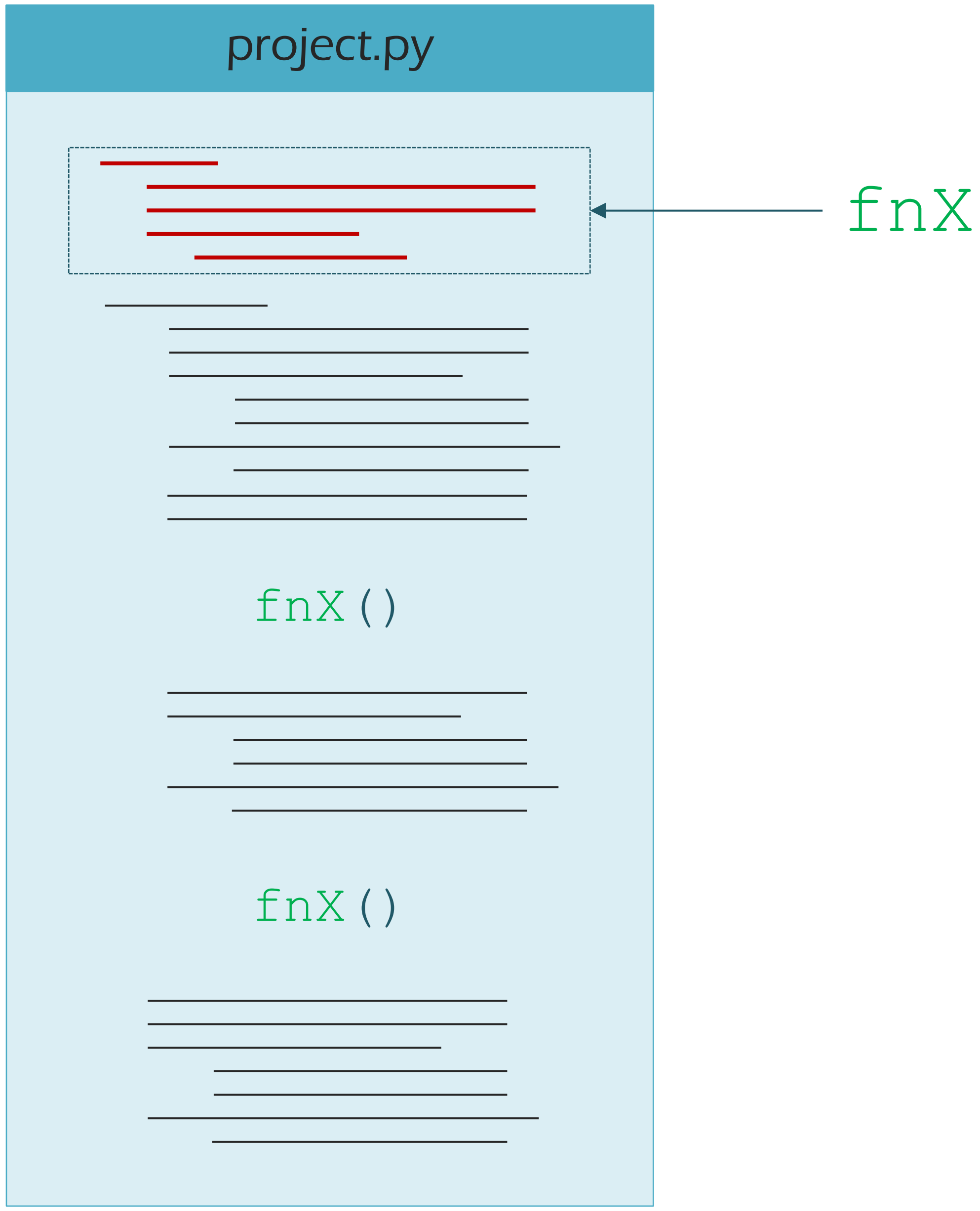
```
name = input("What's your Name? ");
print(name);
```

```
Output:

What's your name?  Mahmoud
Mahmoud
```

Functions

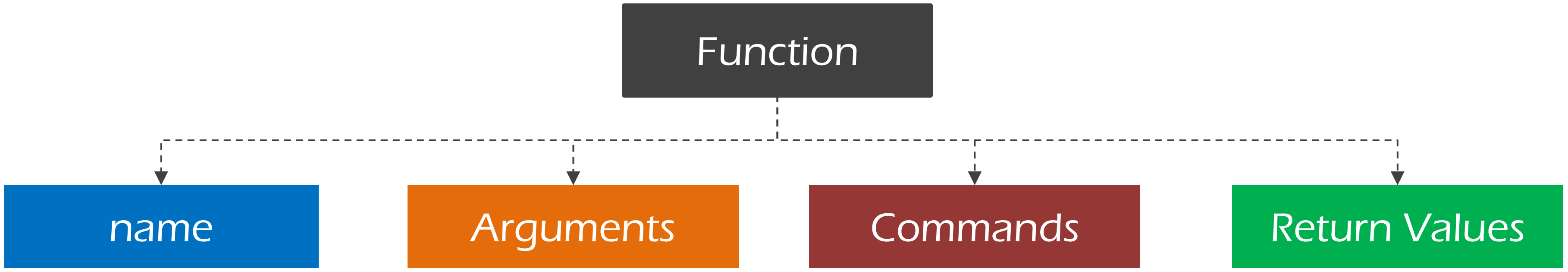
Make your code more generic

Intro



Defining

```python
def fnName:
        pass
```



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

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

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
```

*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
```

**keywords

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

Calling It

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

Scope

To know your limits

Lexical Scope

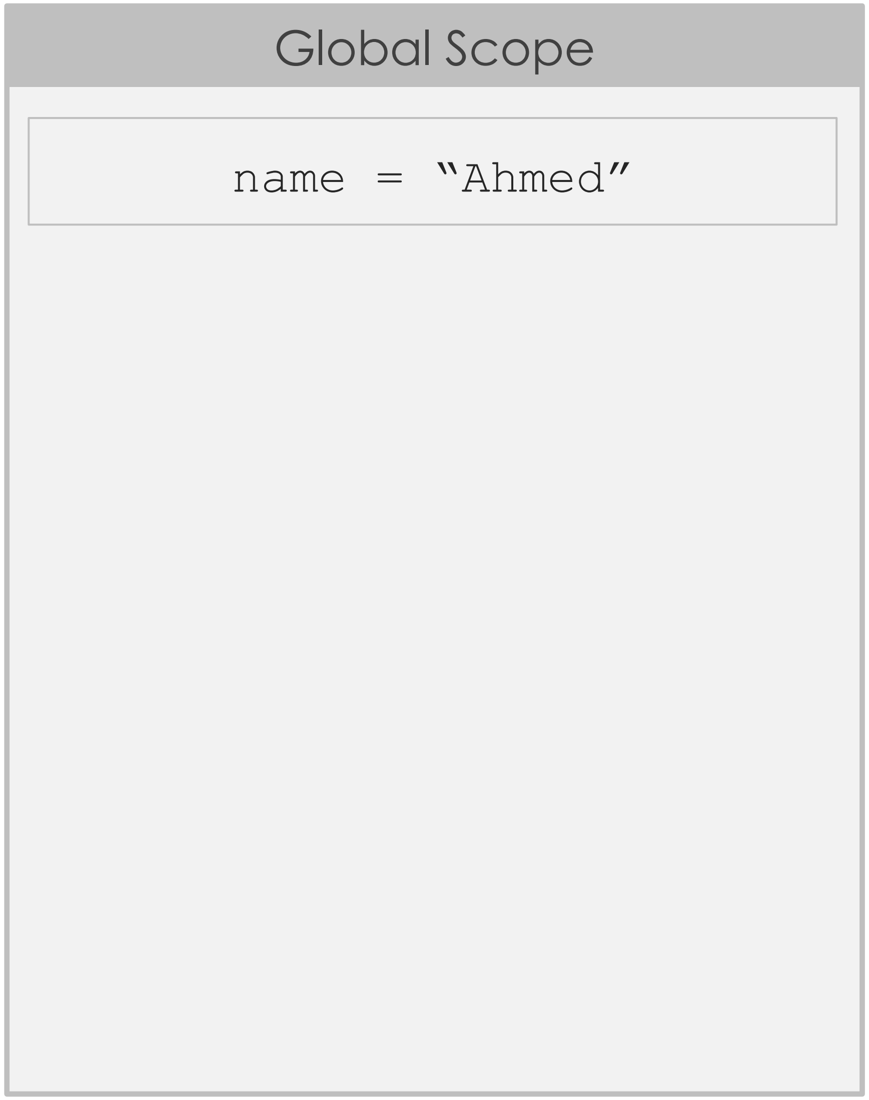
Global Scope

Output:

Lexical Scope

```
name = "Ahmed"
```

Output:



Lexical Scope

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

Output:

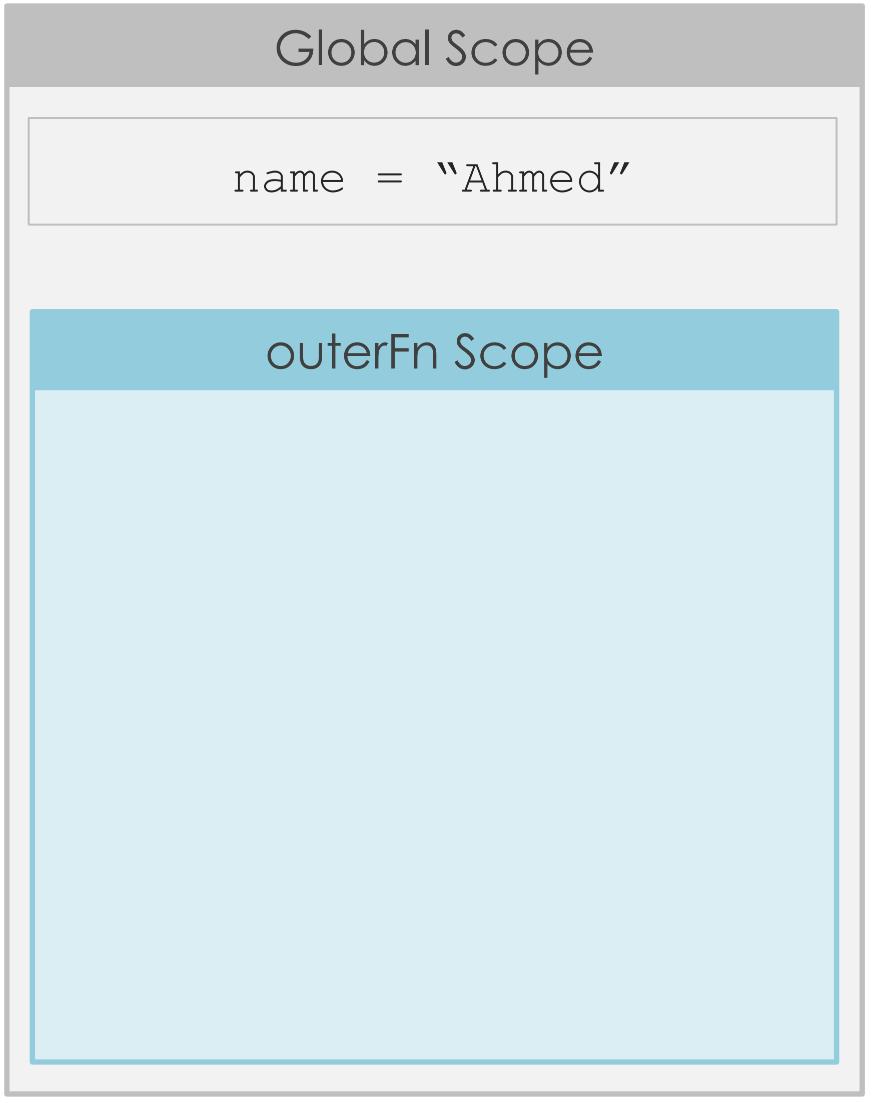
Global Scope

```
name = “Ahmed”
```

Lexical Scope

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

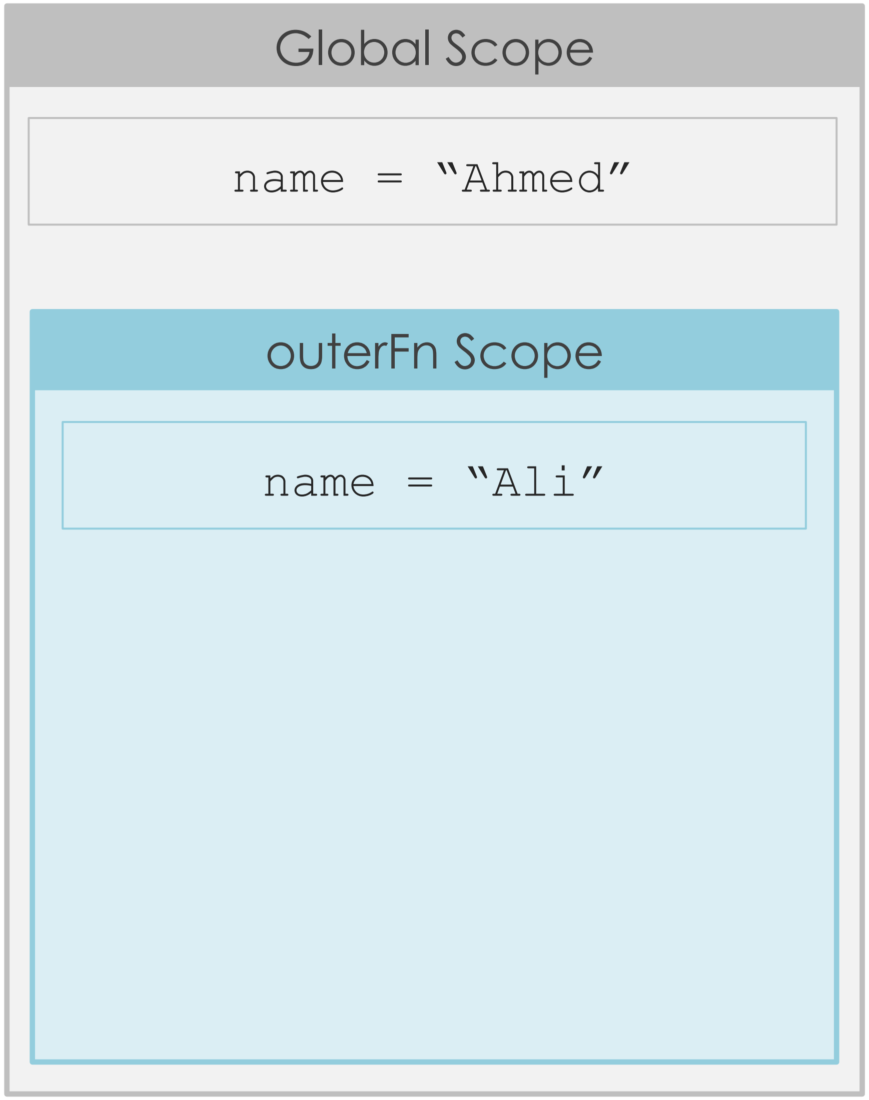
Output:



Lexical Scope

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

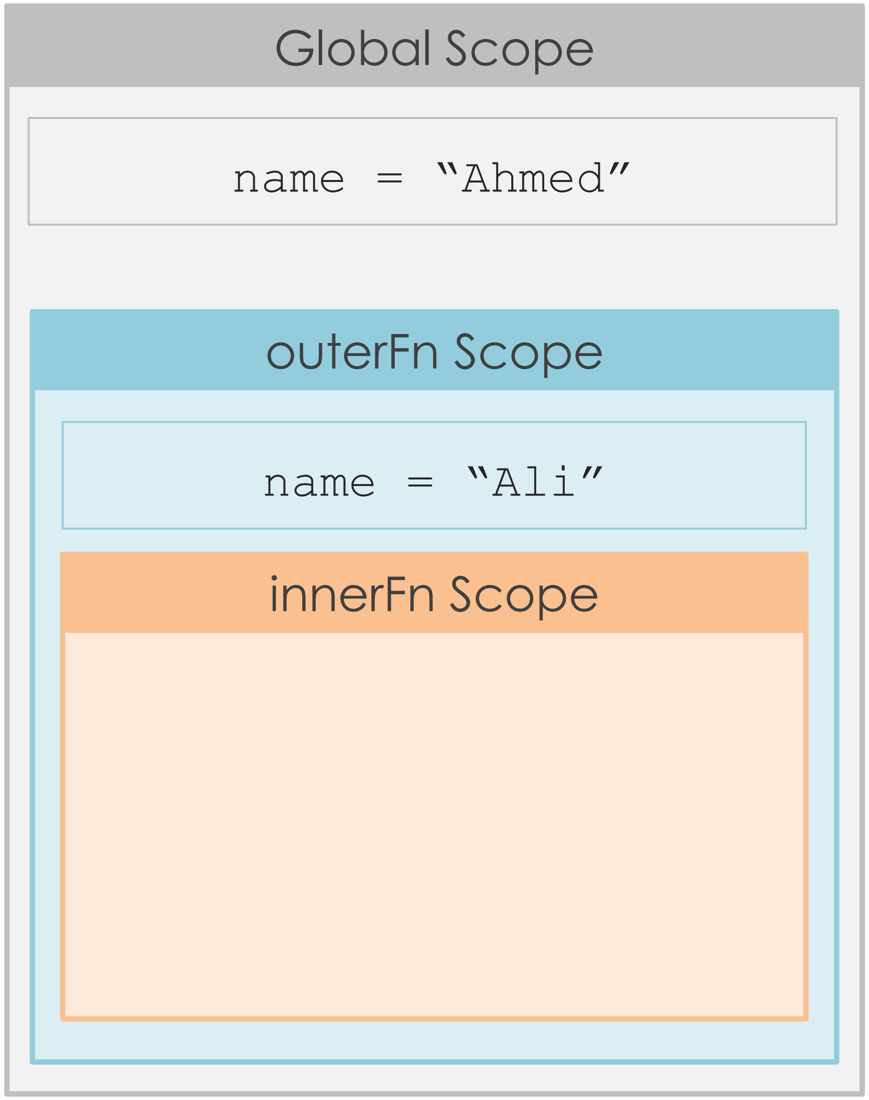
Output:



Lexical Scope

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

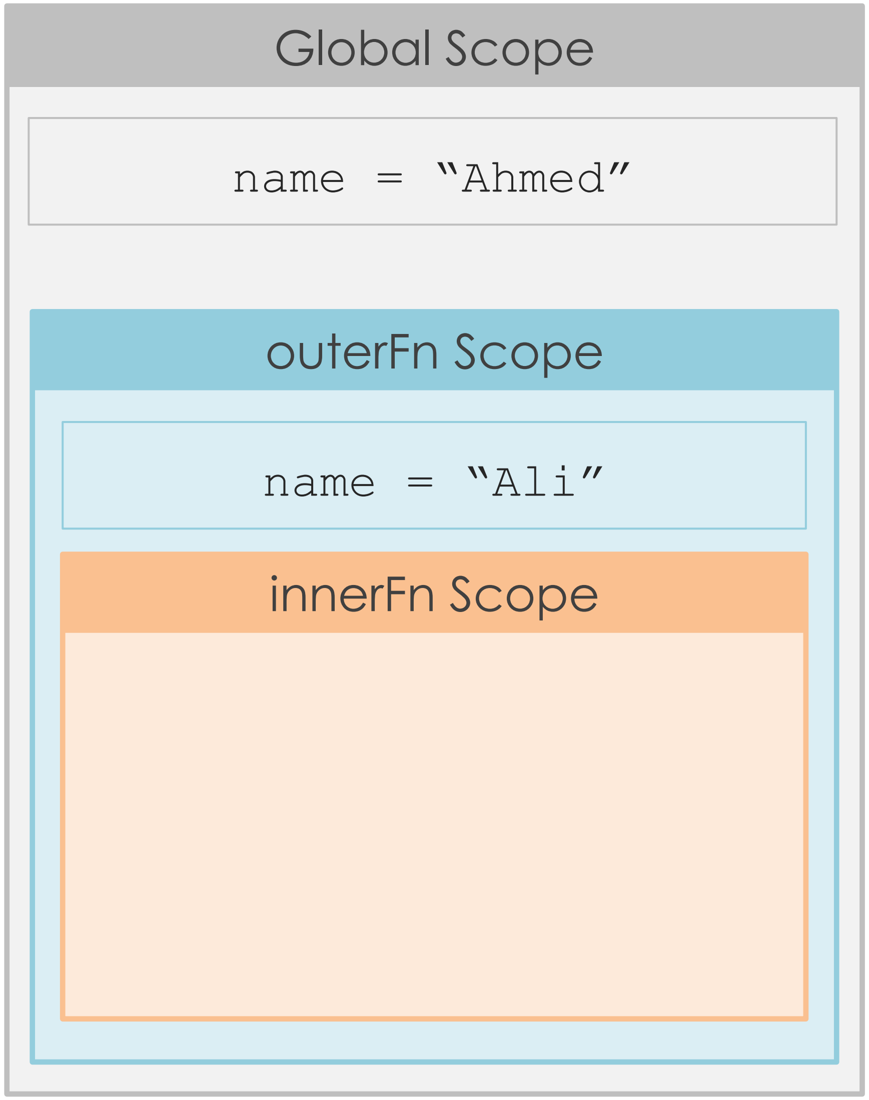
```
Output:
```



Lexical Scope

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

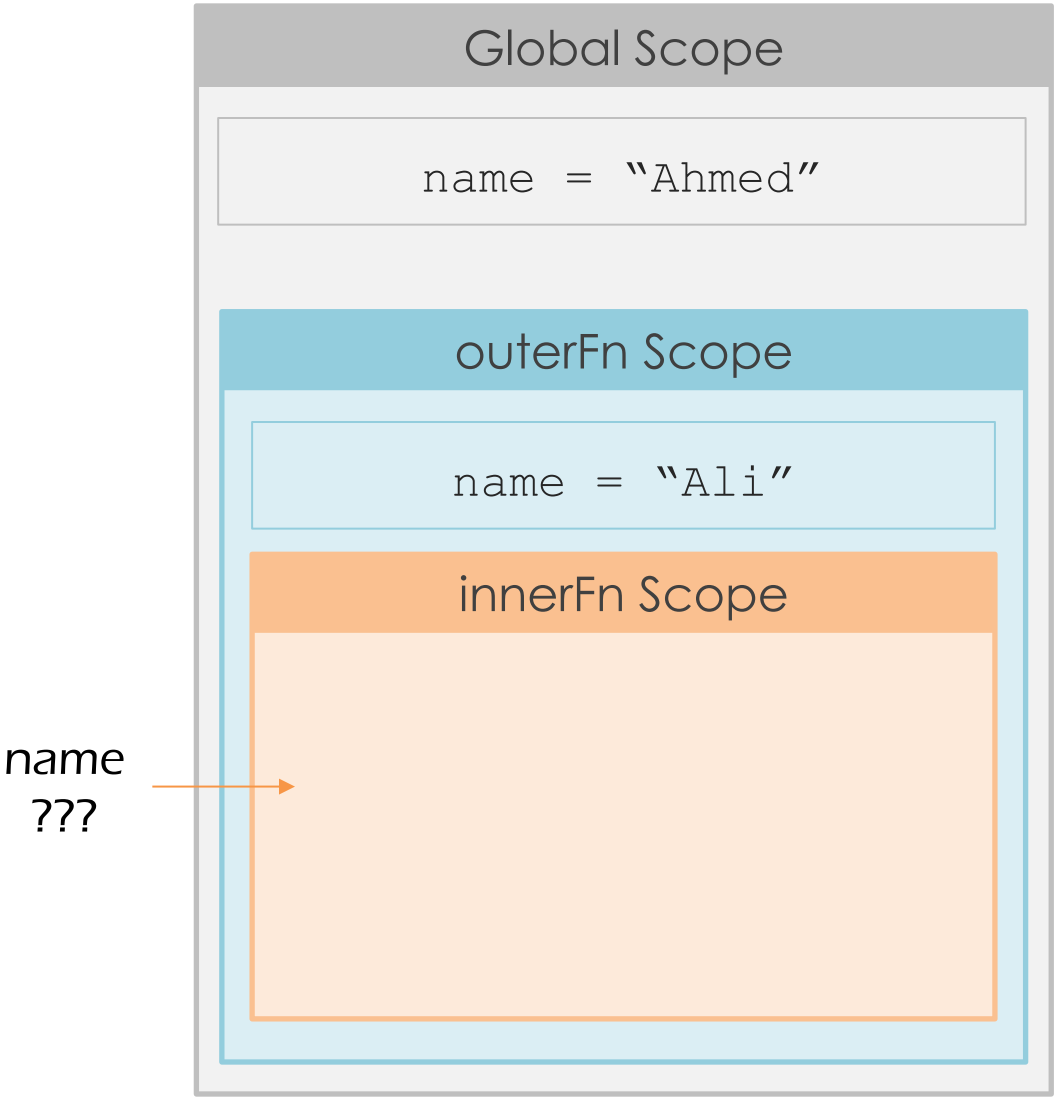
Output:



Lexical Scope

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

Output:

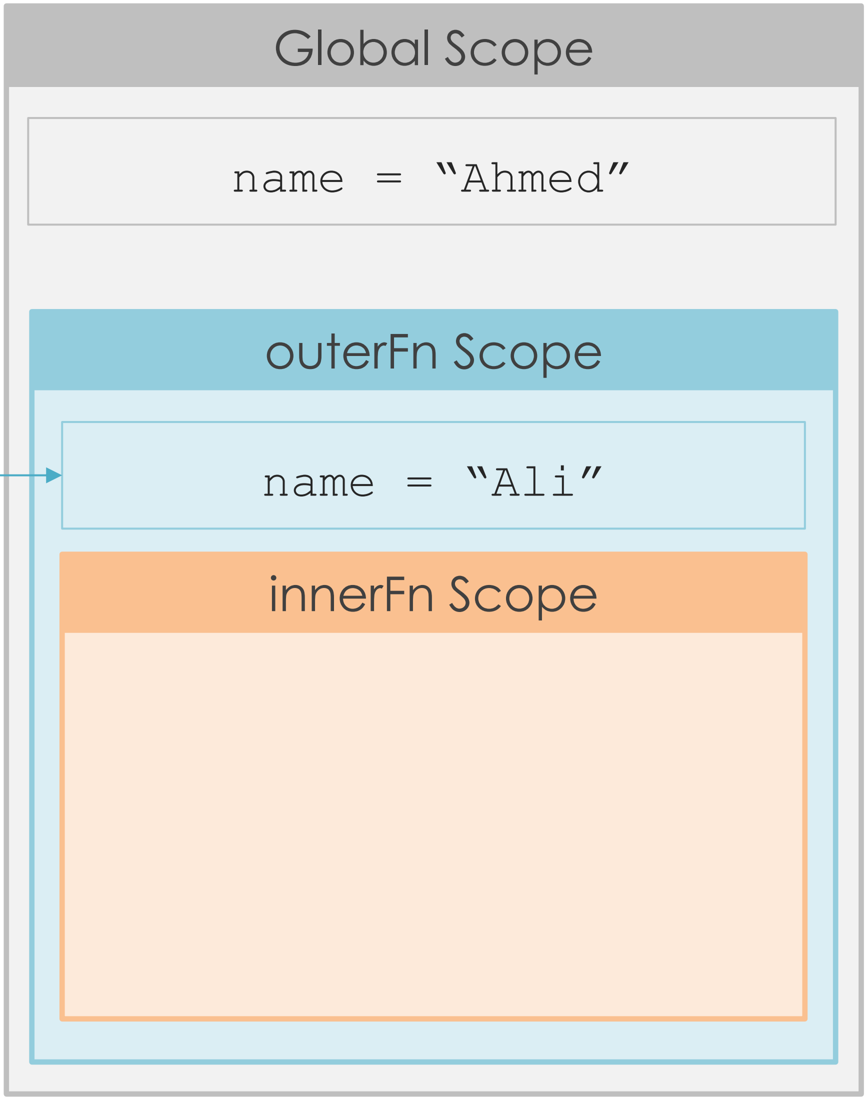


Lexical Scope

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

Output:

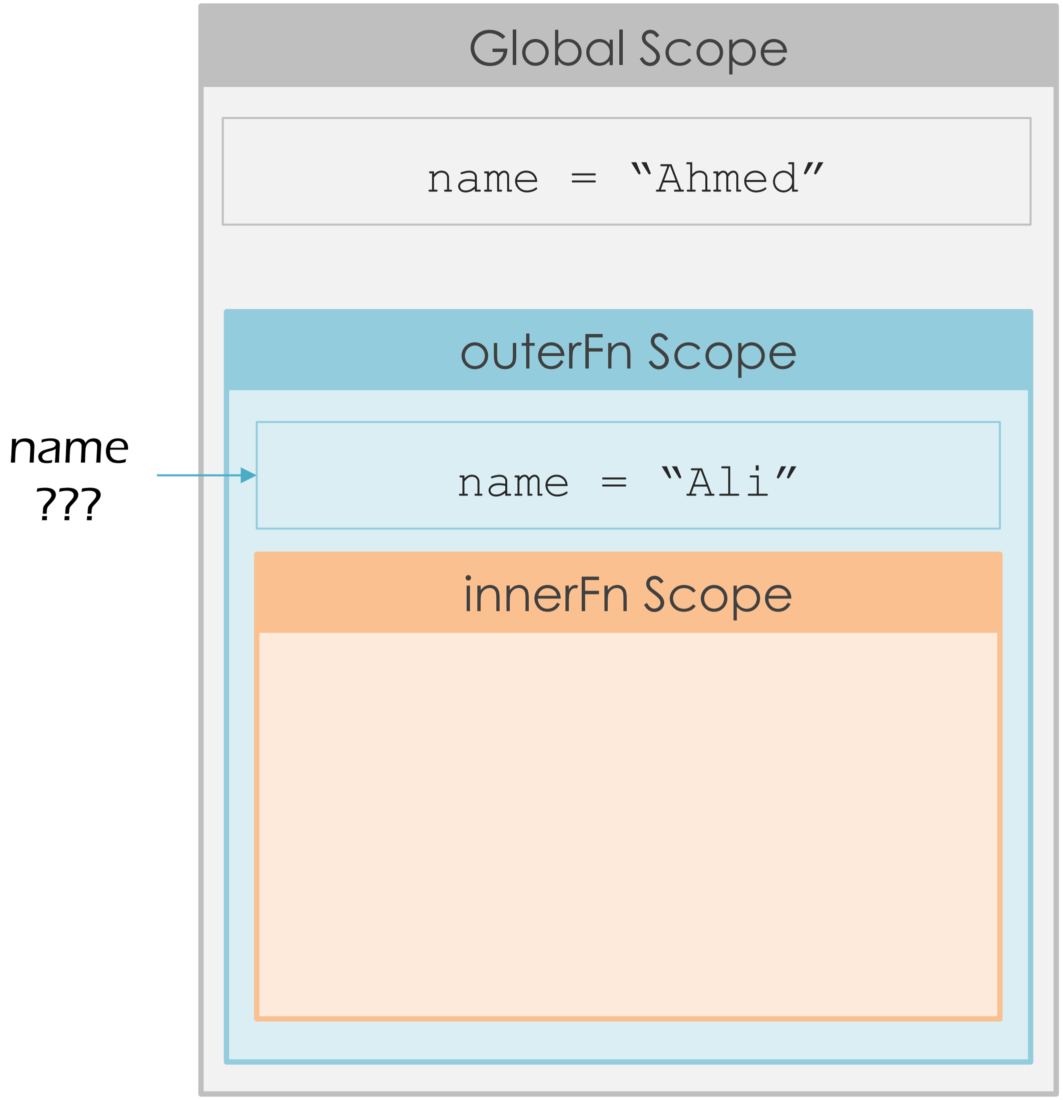
name
???



Lexical Scope

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

```
Output:
Ali
```



Lexical Scope

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

Output:

```
Ali
```

Global Scope

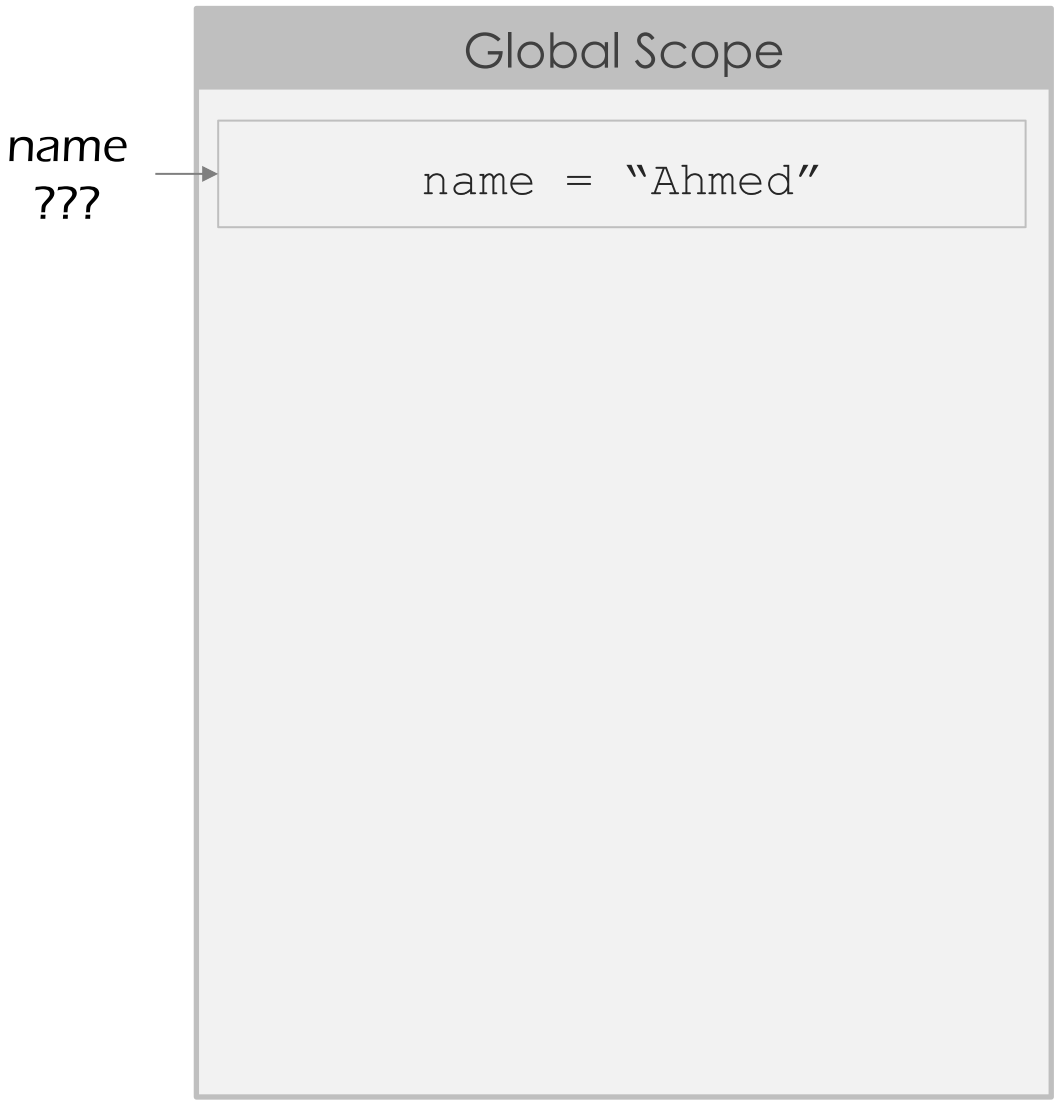
```
name = "Ahmed"
```

Lexical Scope

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

Output:

```
Ali
```

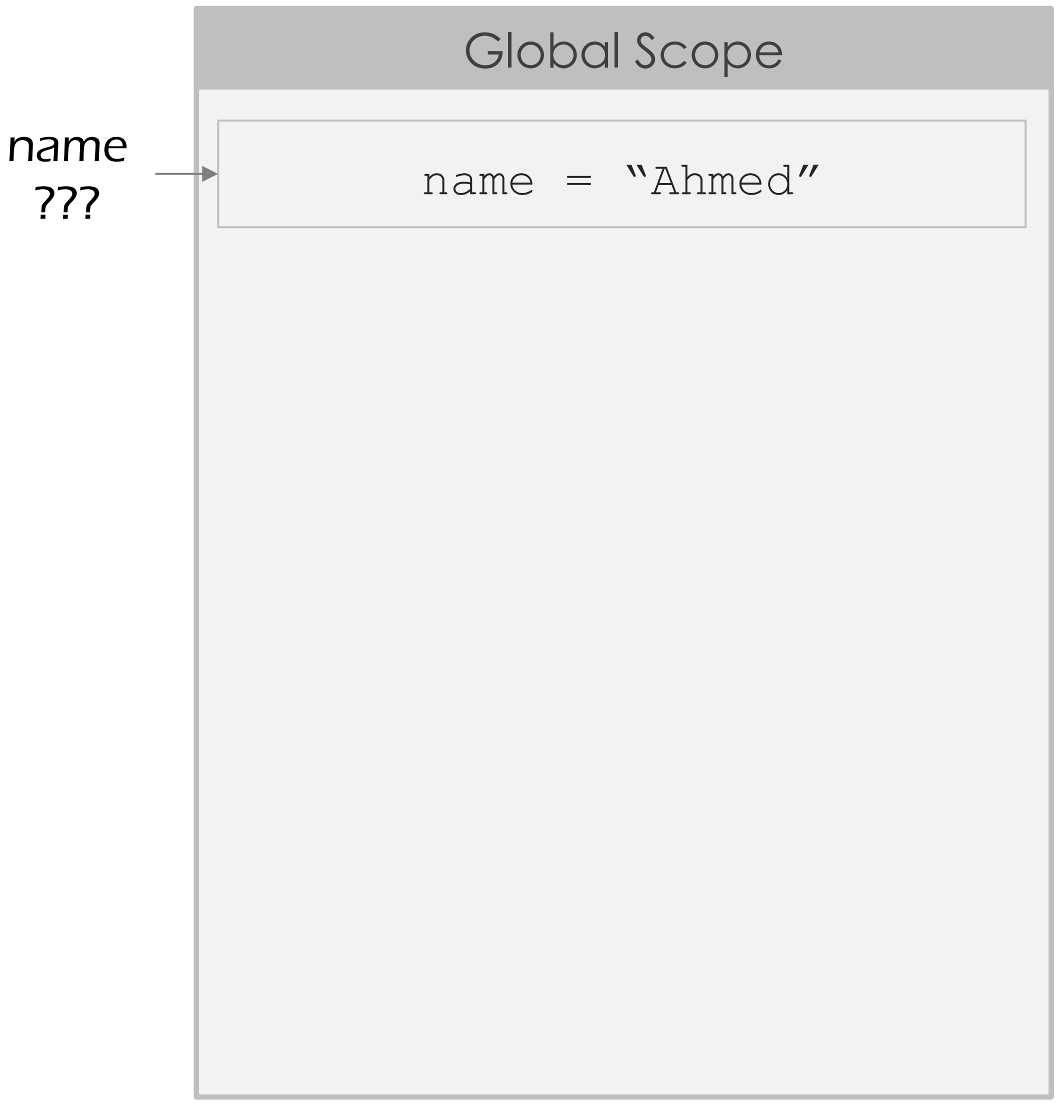


Lexical Scope

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

Output:

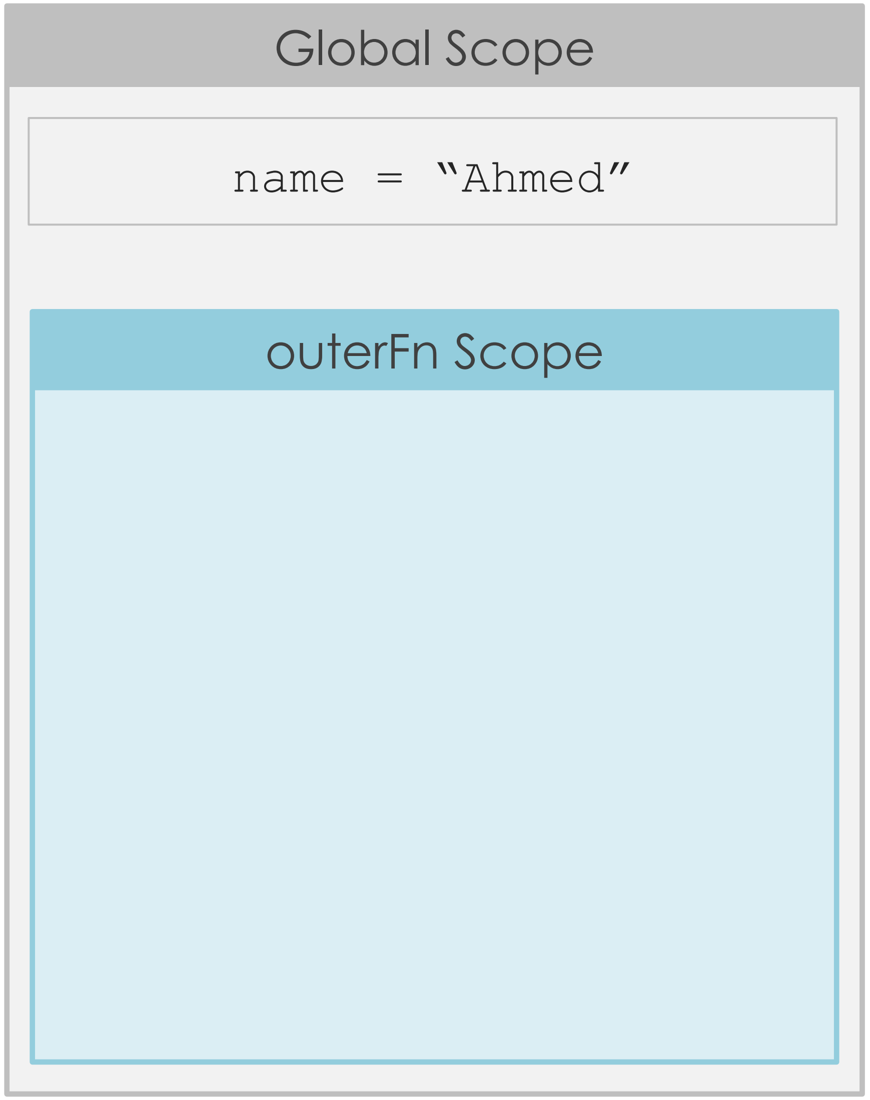
```
Ali
Ahmed
```



global Keyword

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

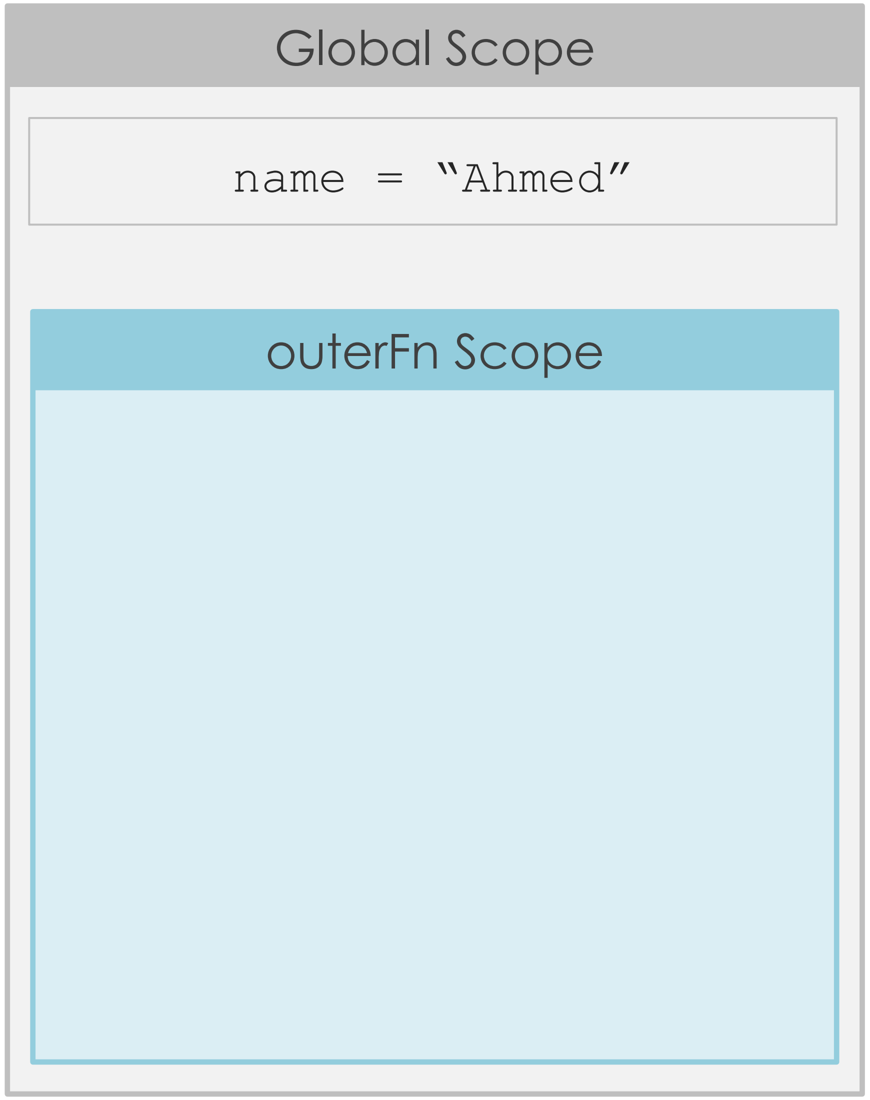
Output:



global Keyword

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

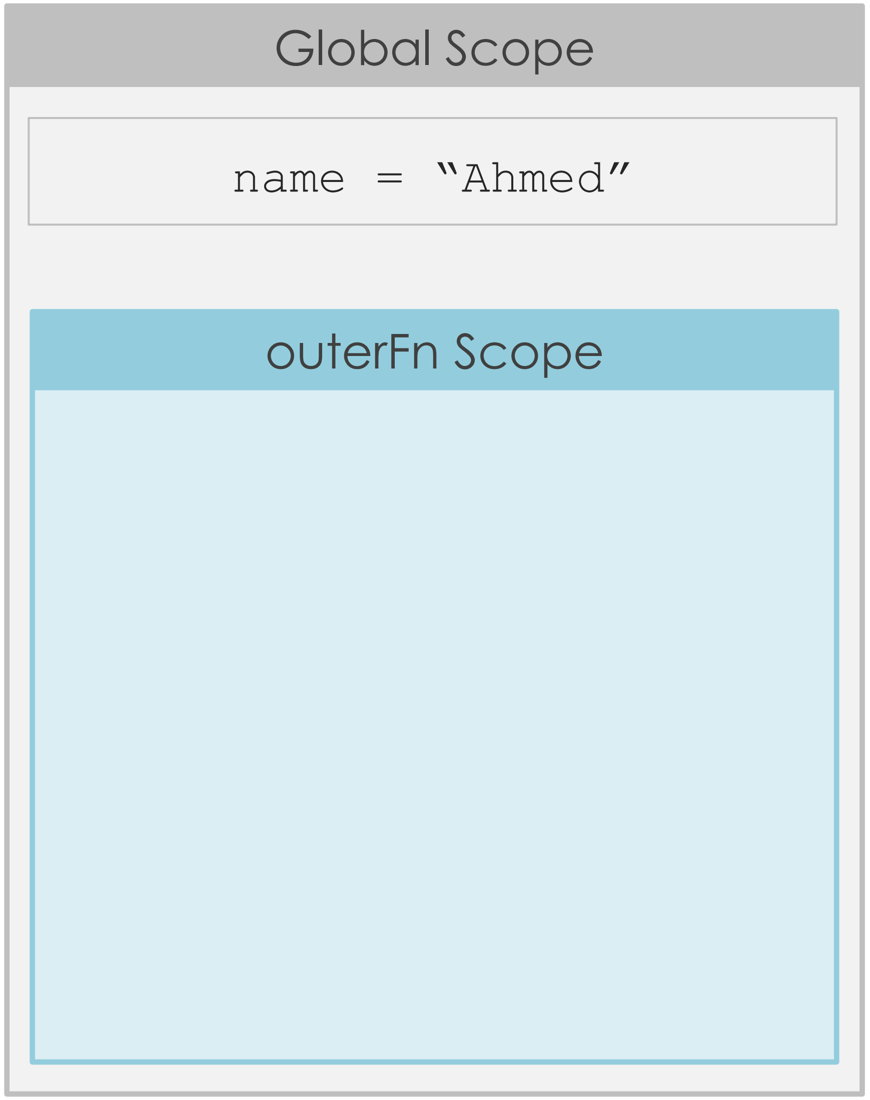
Output:



global Keyword

```
name = “Ahmed”
def outerFn():
      global name
   →  name = “Ali”
      def innerFn():
            print(name)
      innerFn()
outerFn()
```

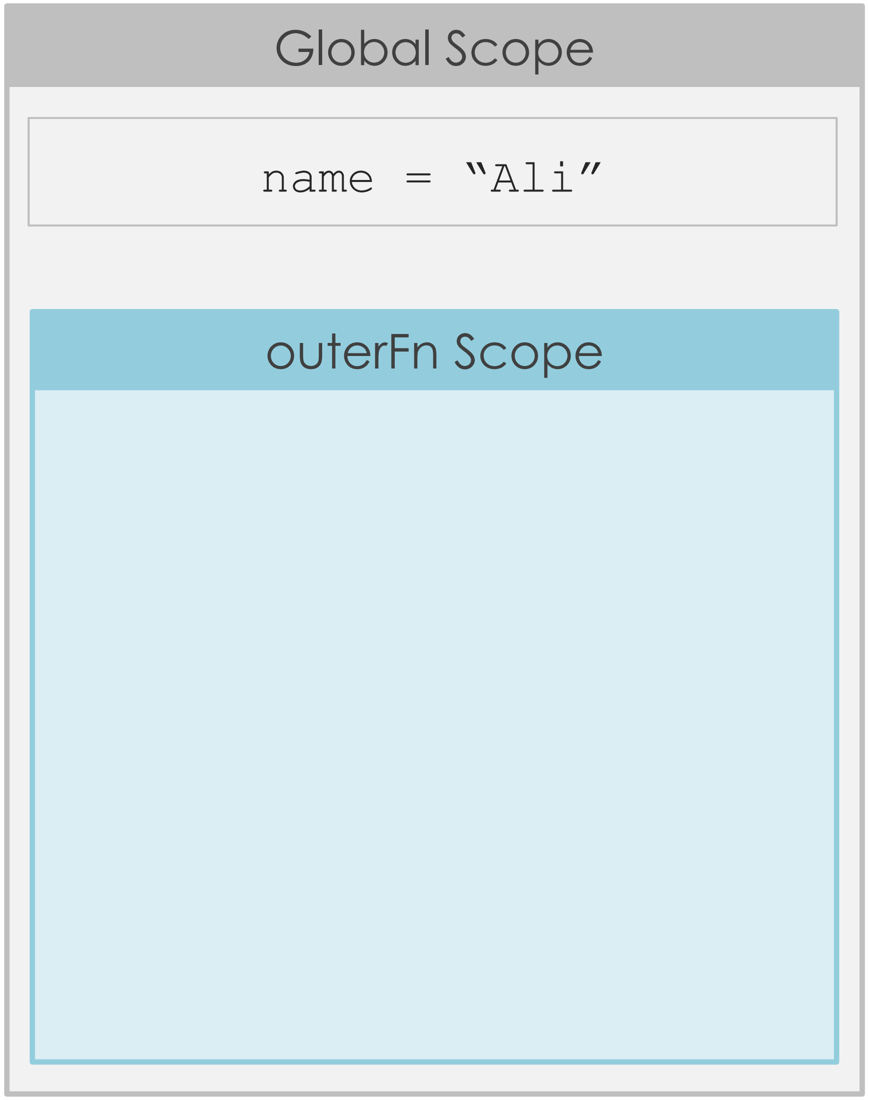
Output:



global Keyword

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

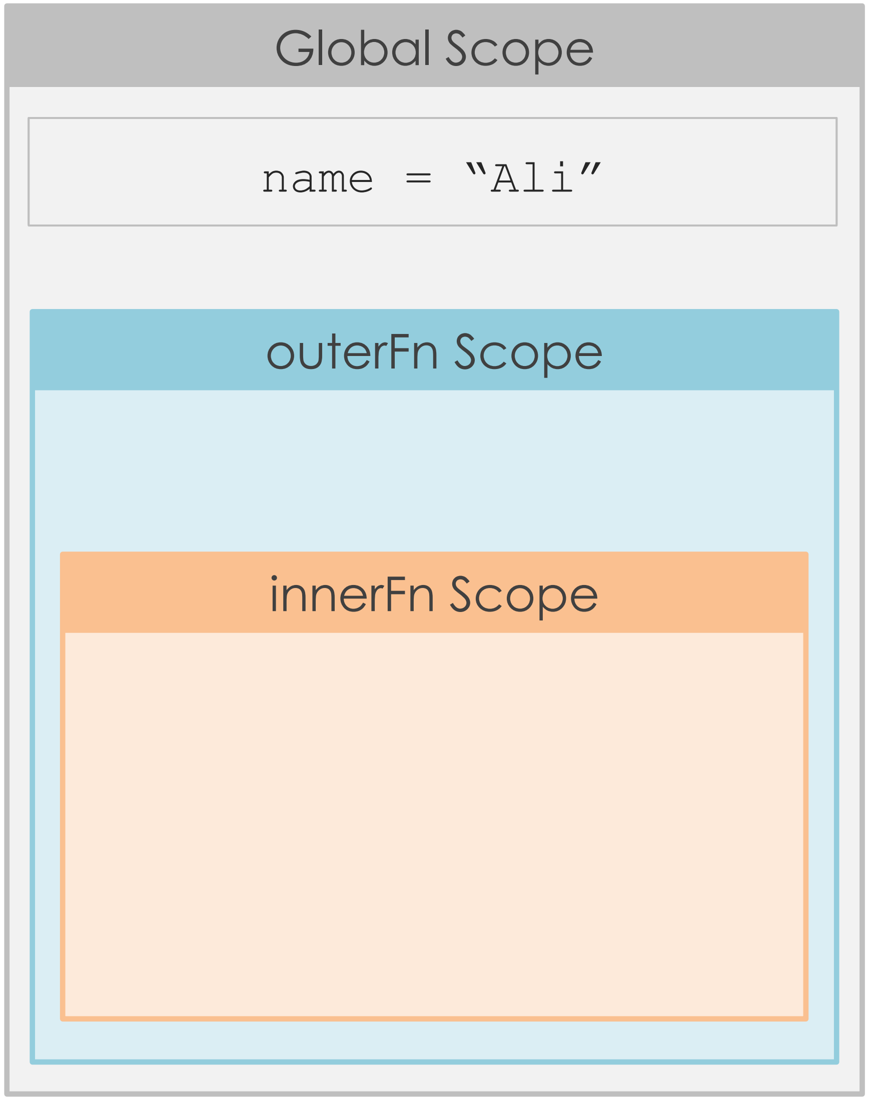
Output:



global Keyword

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

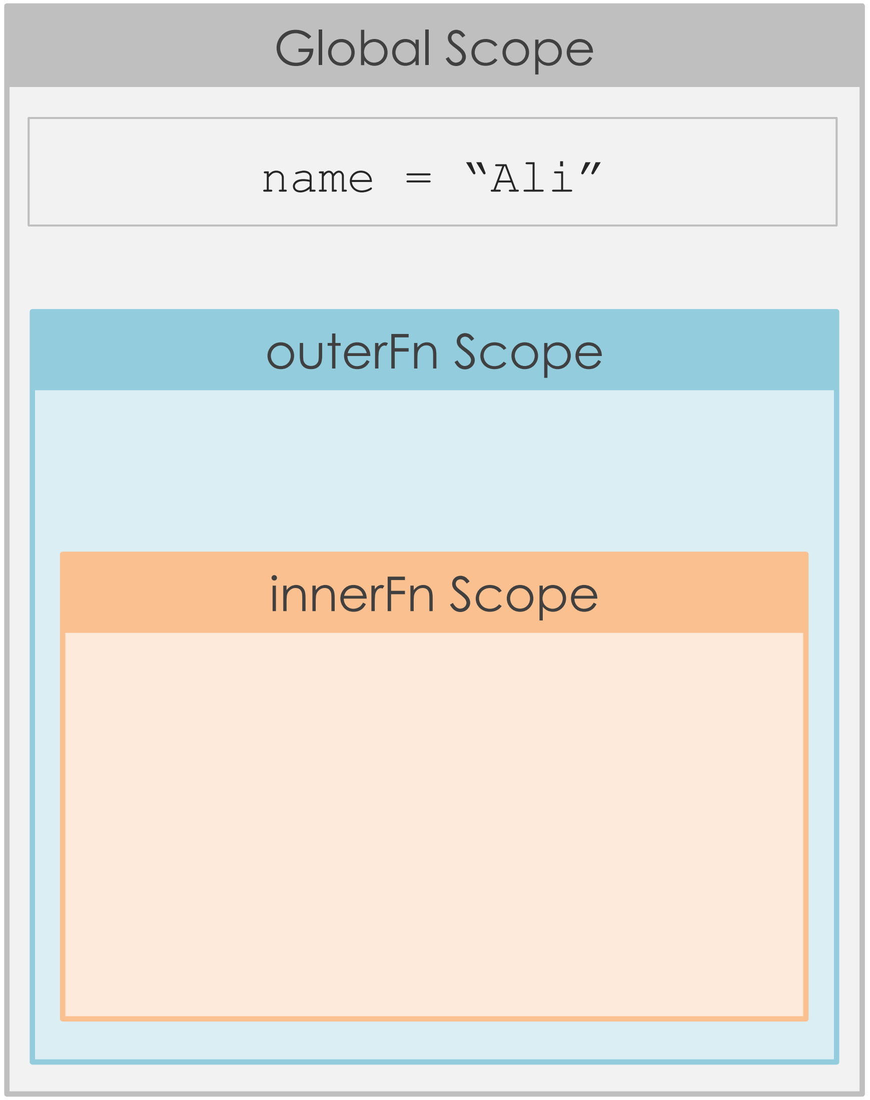
Output:



global Keyword

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

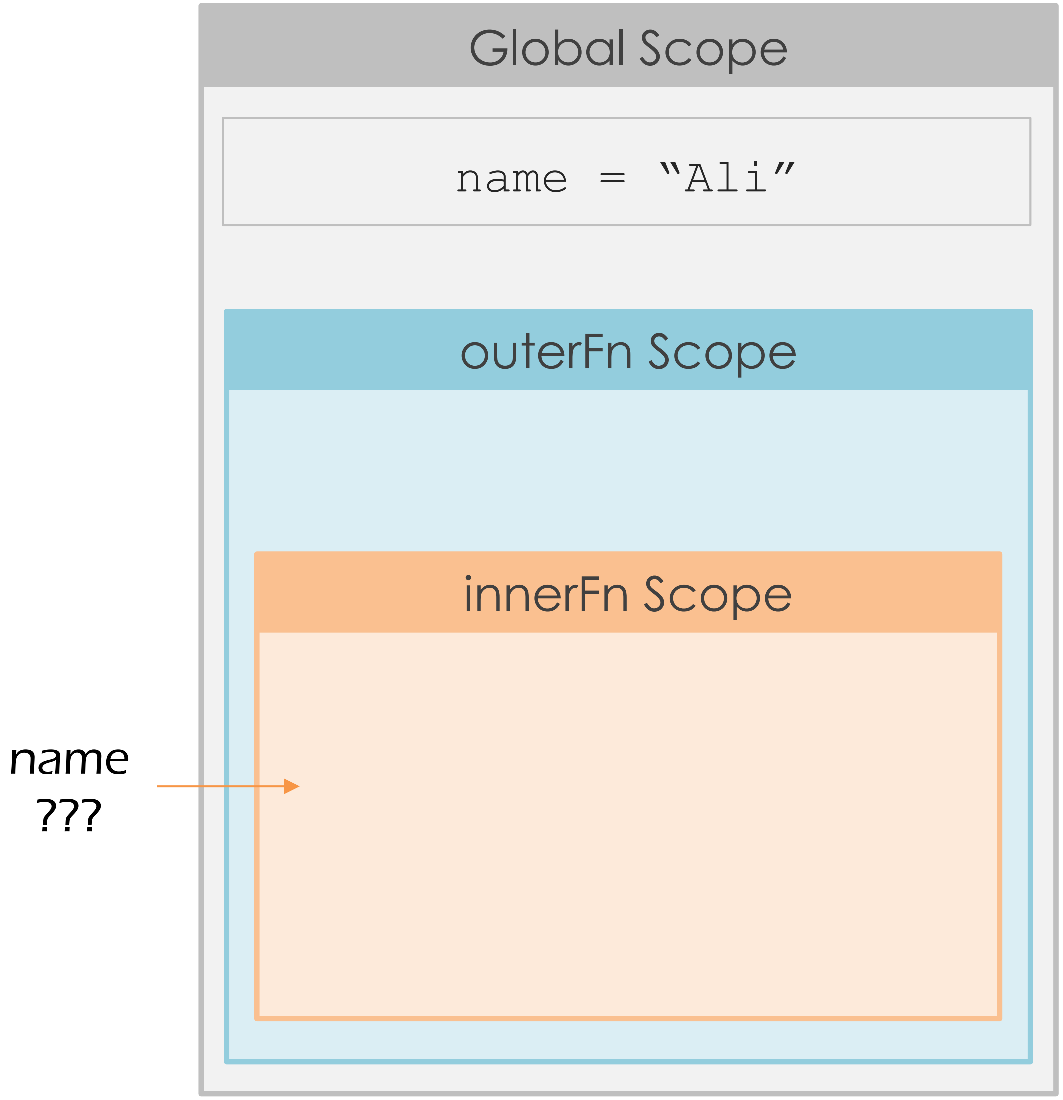
Output:



global Keyword

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

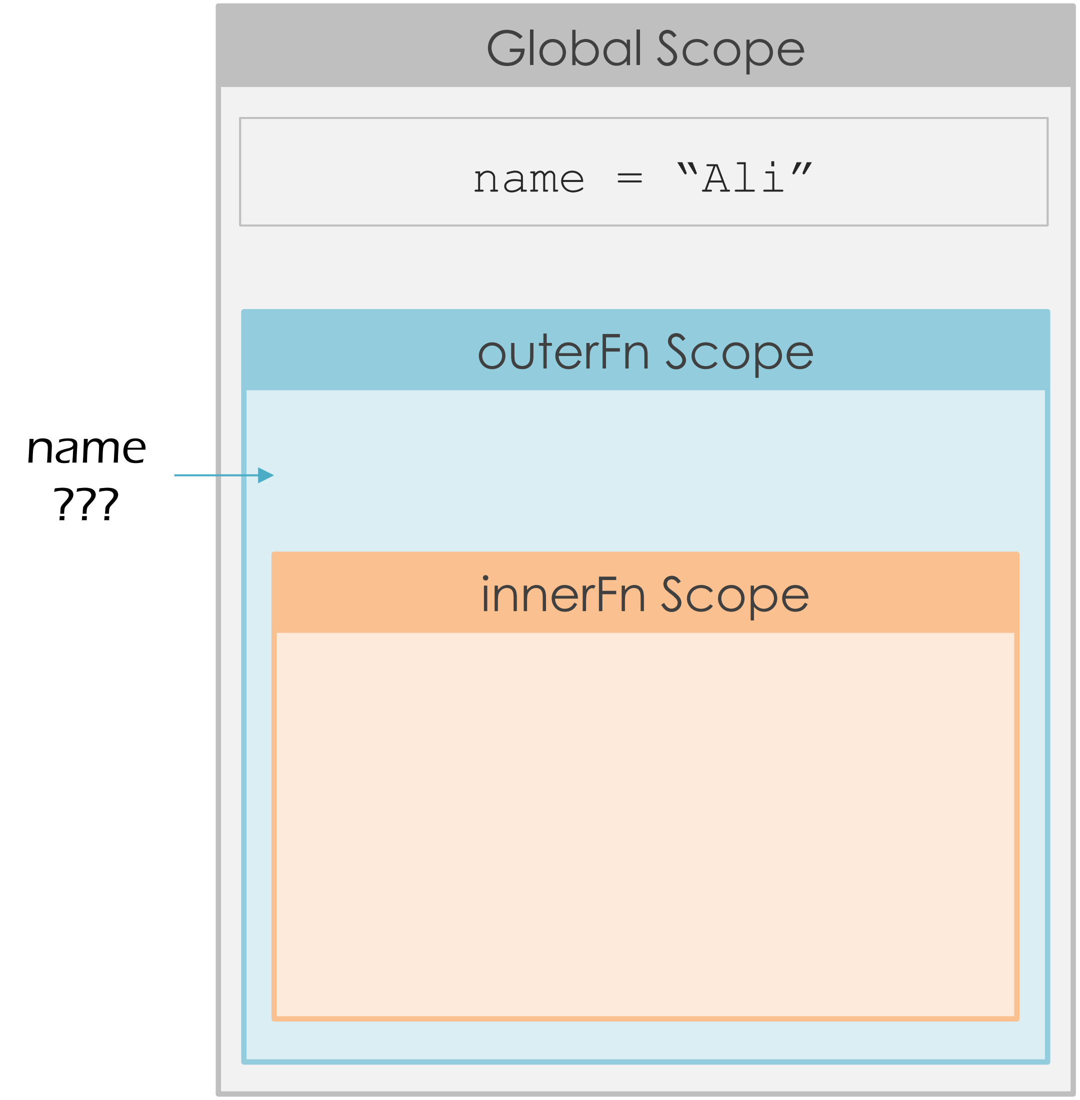
Output:



global Keyword

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

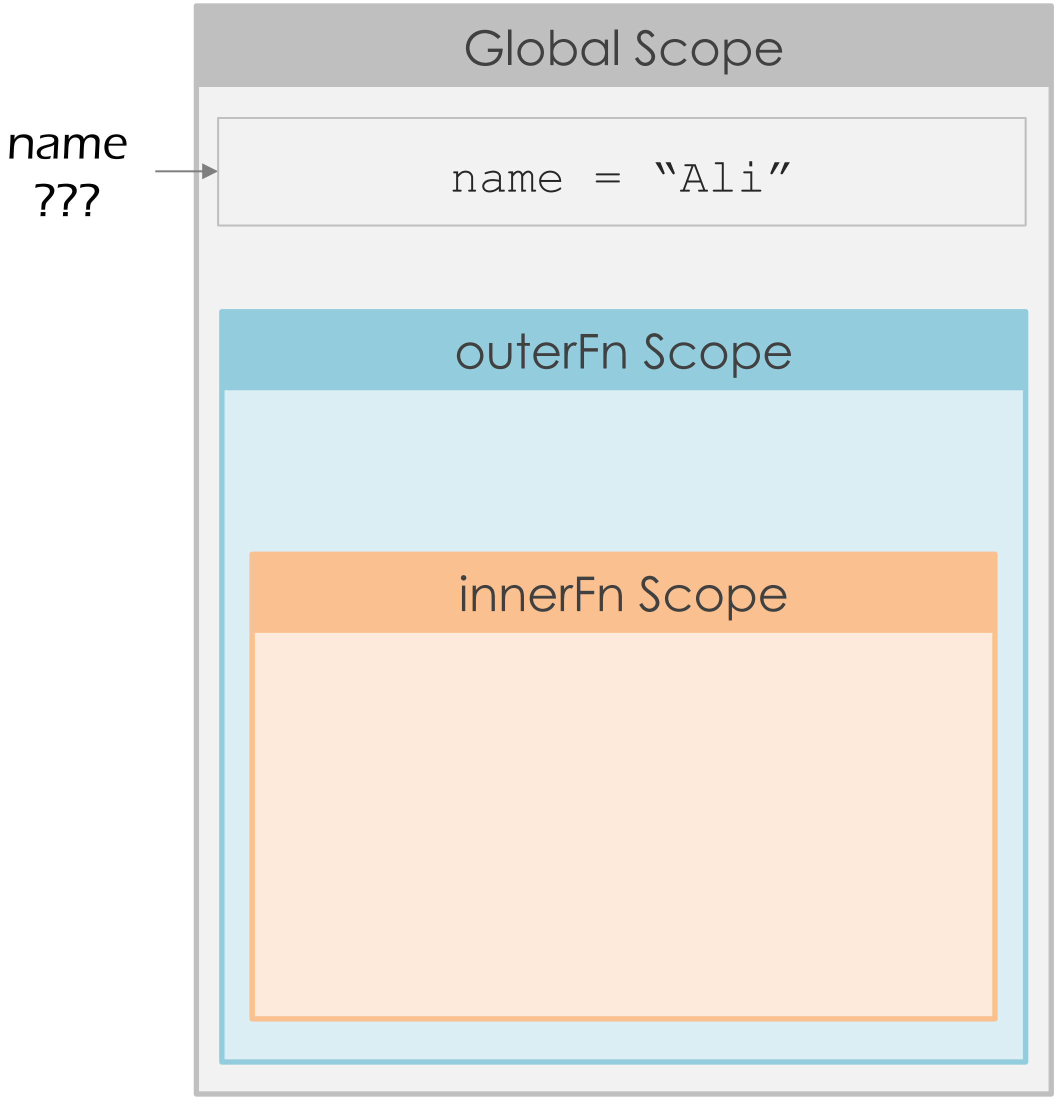
Output:



global Keyword

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

Output:

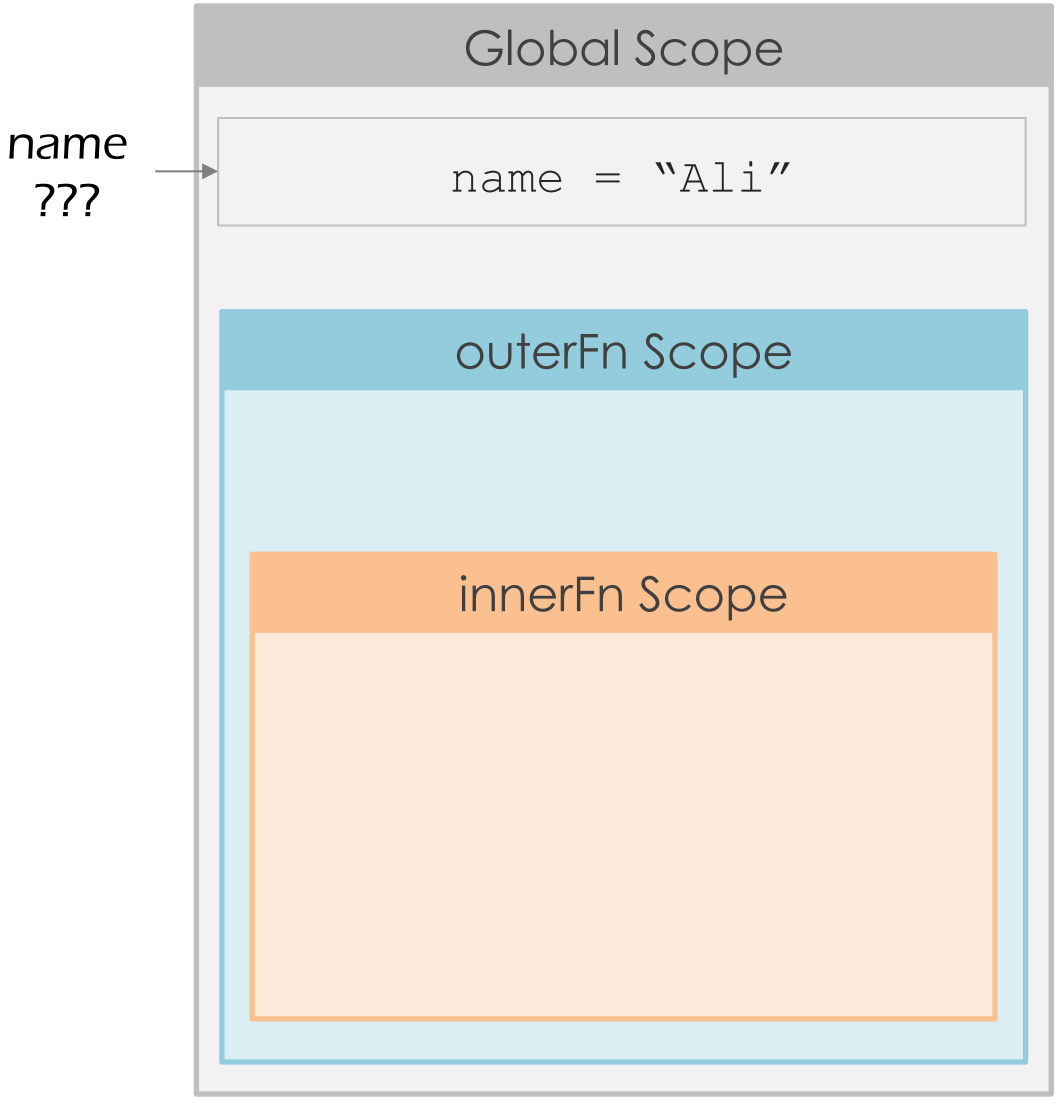


global Keyword

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

Output:

```
Ali
```

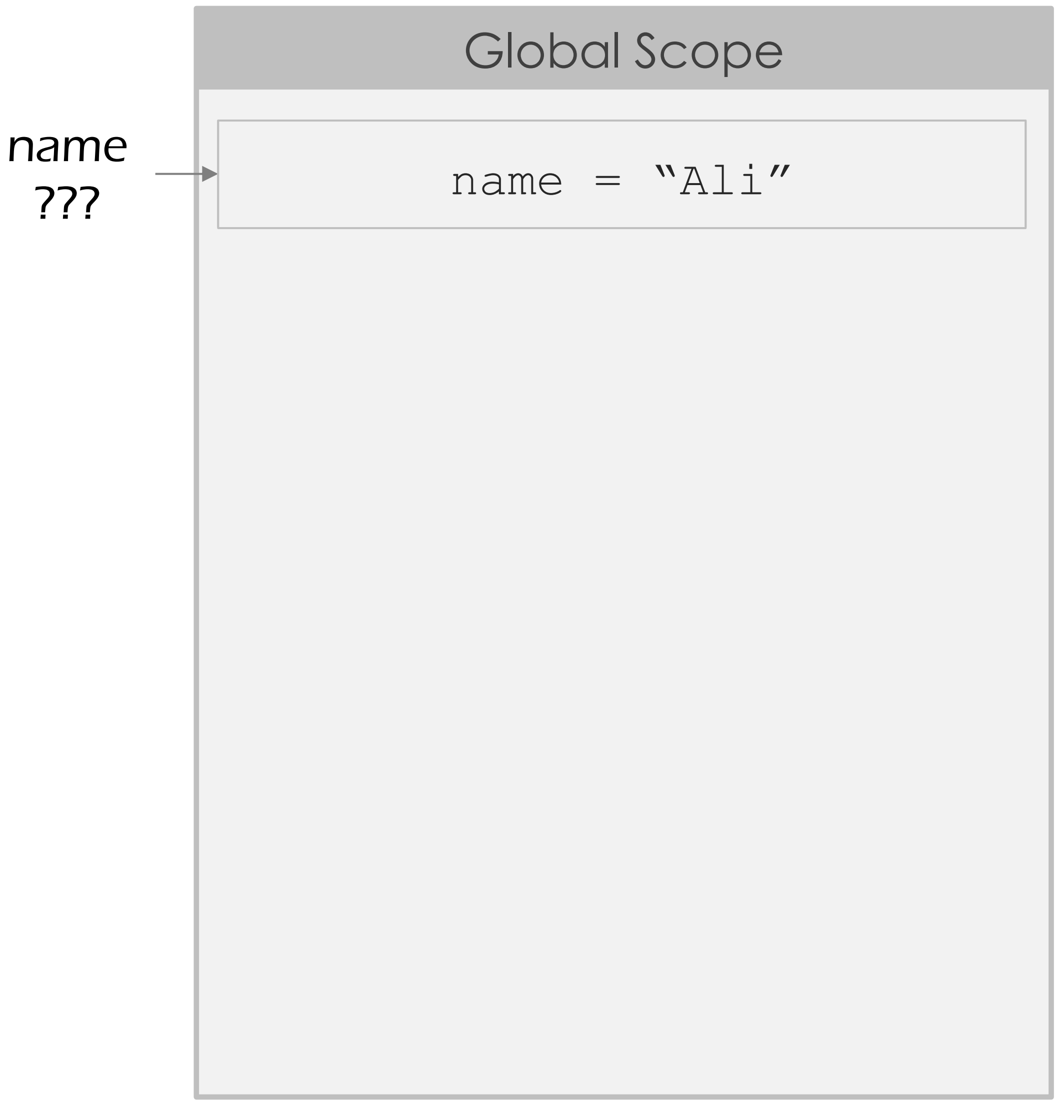


global Keyword

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

Output:

```
Ali
```

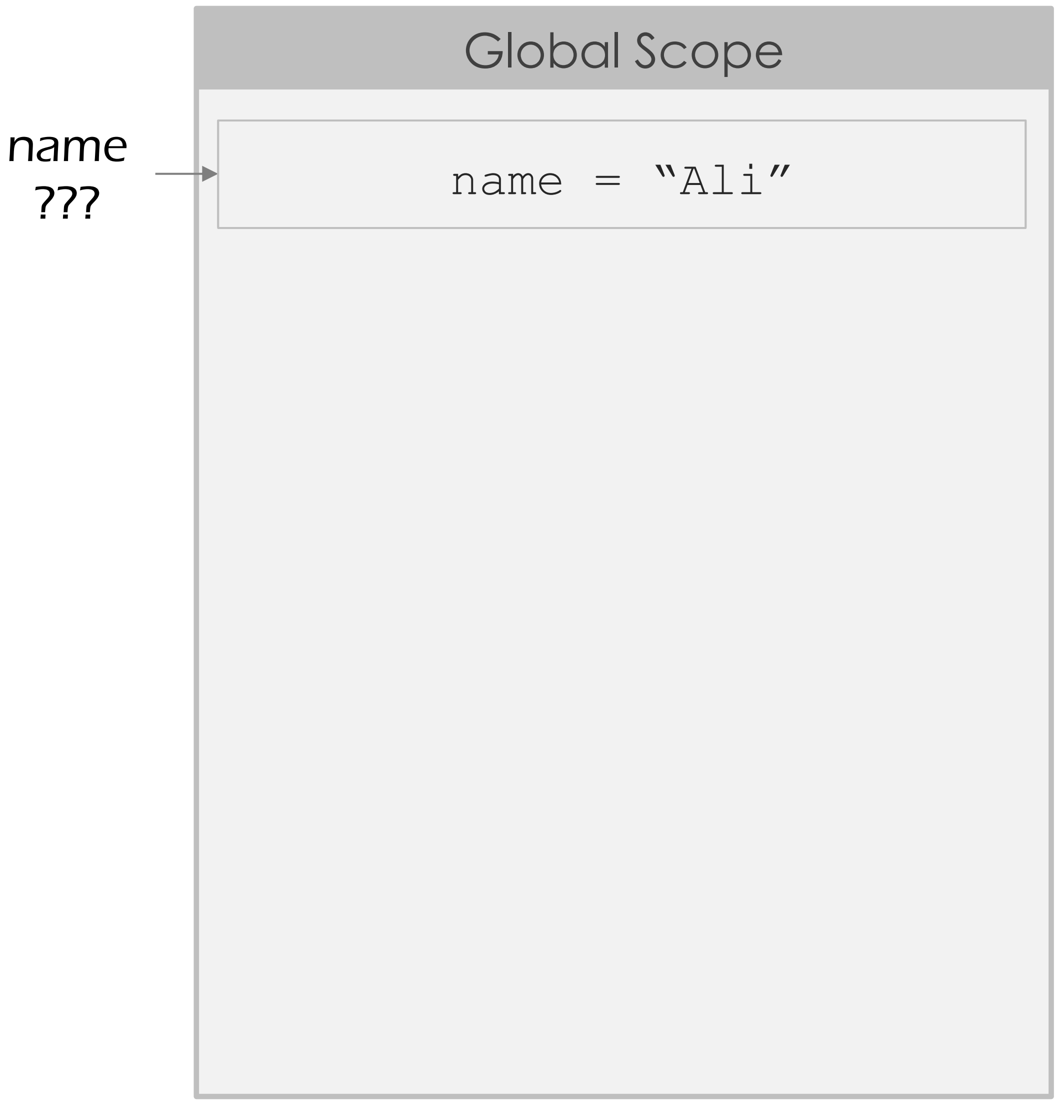


global Keyword

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

Output:

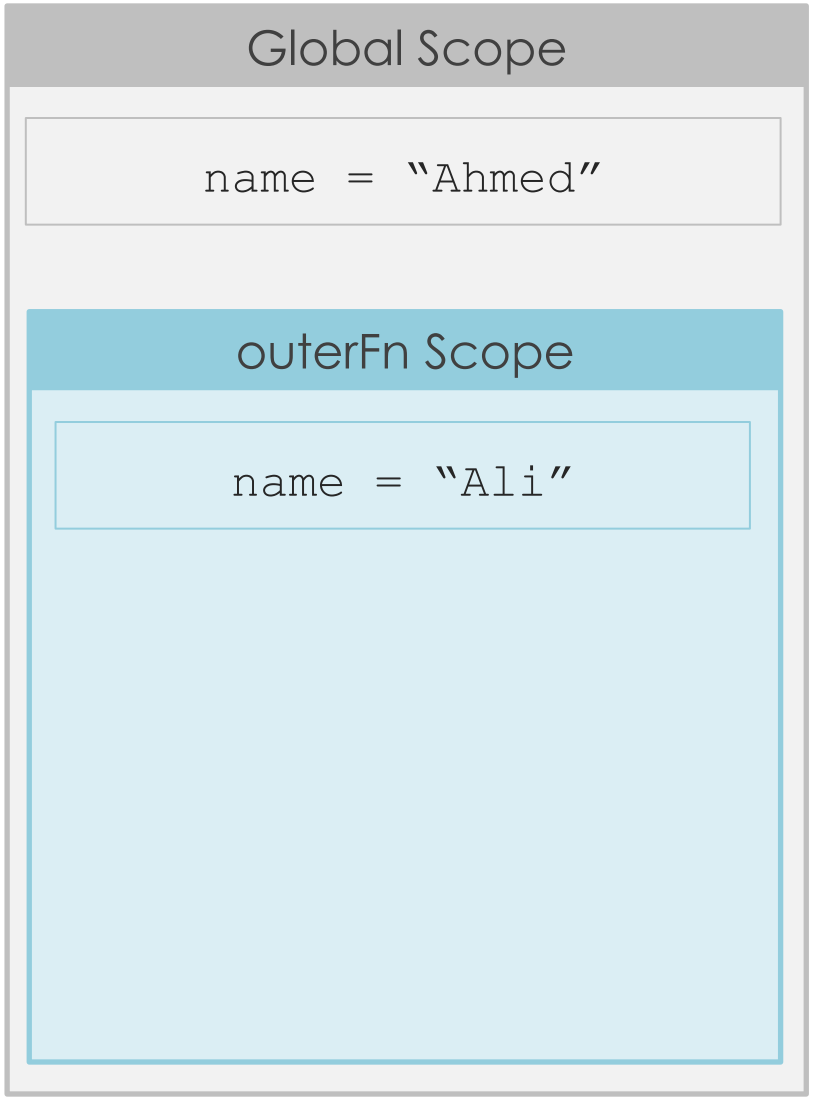
```
Ali
Ali
```



nonlocal Keyword

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

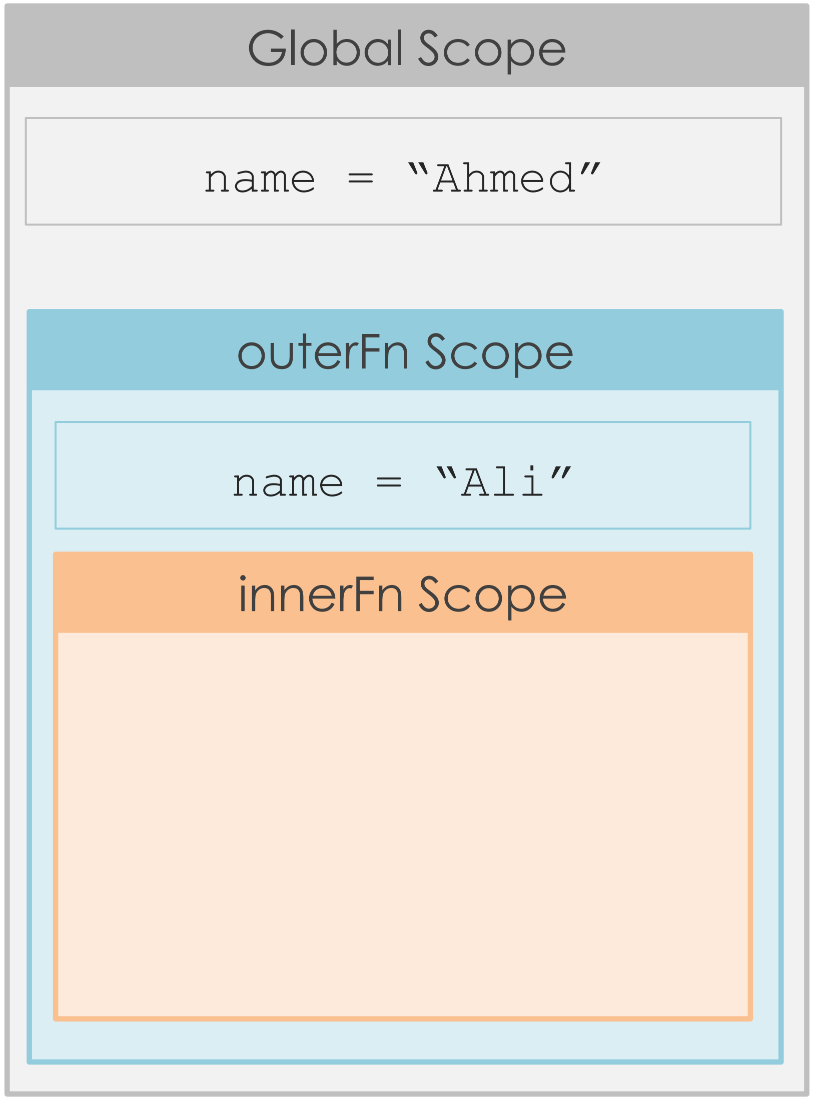
Output:



nonlocal Keyword

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

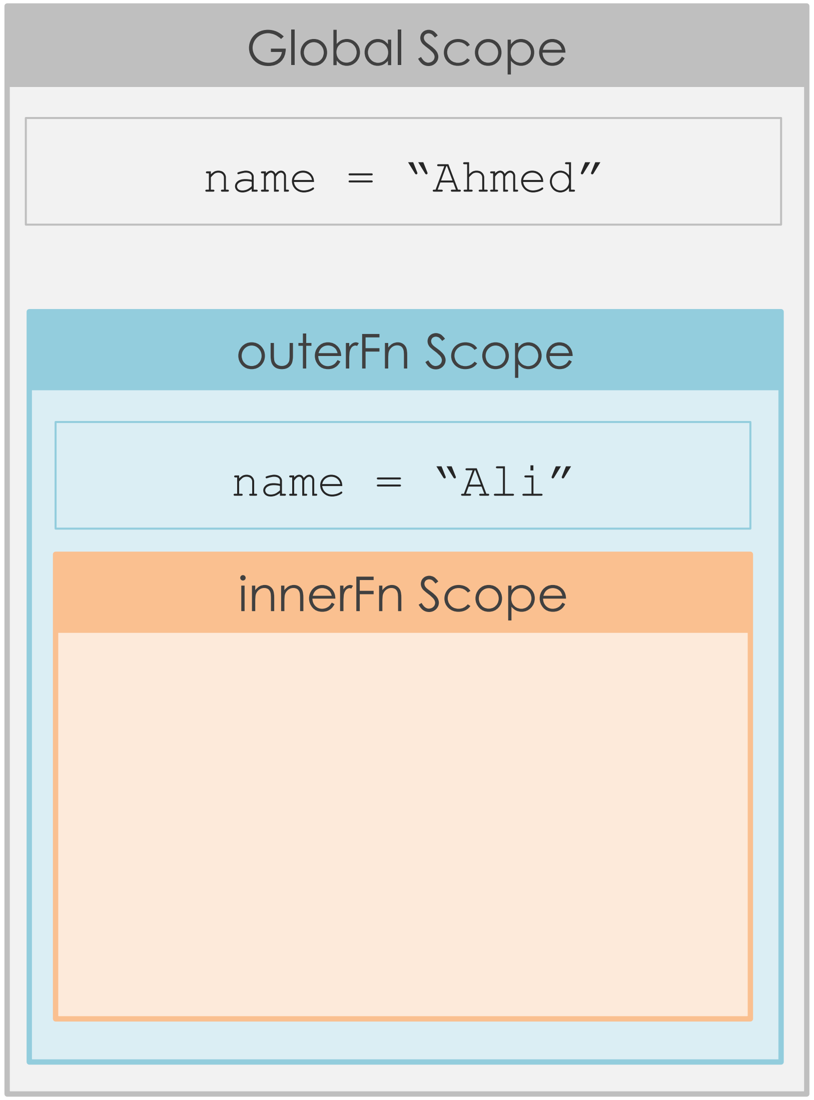
Output:



nonlocal Keyword

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

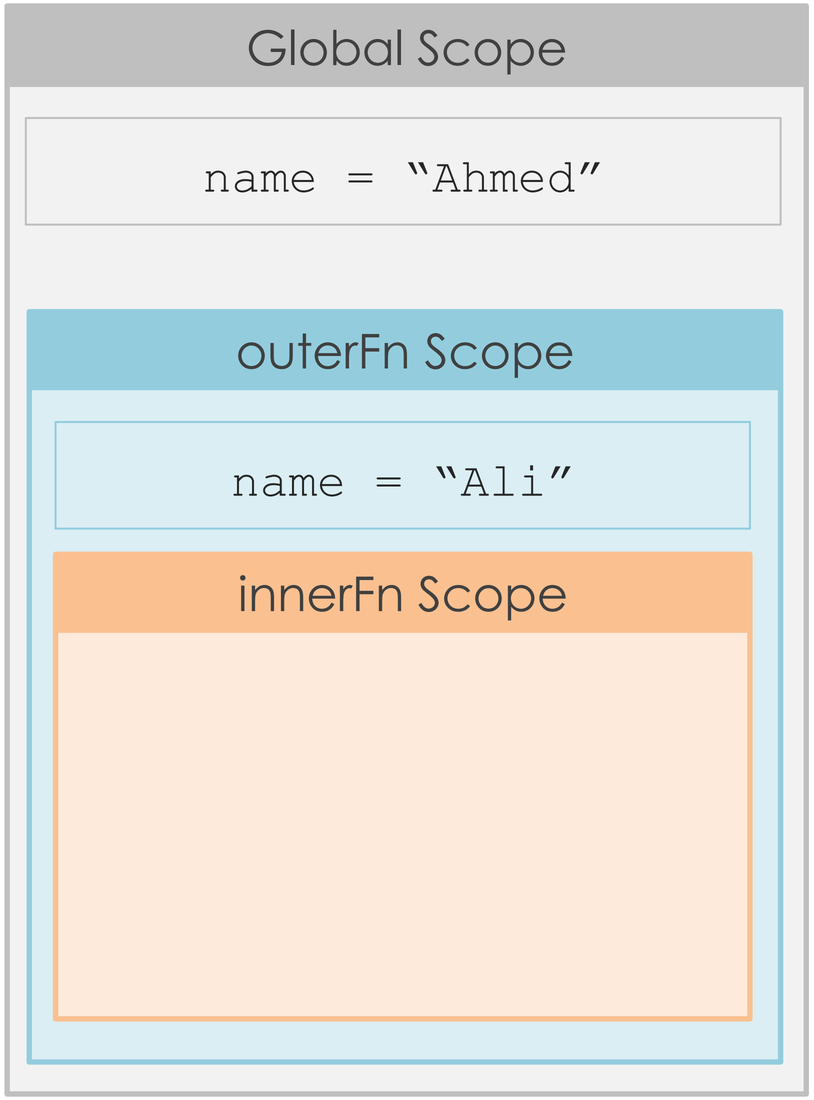
Output:



nonlocal Keyword

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

```
Output:
Ali
```

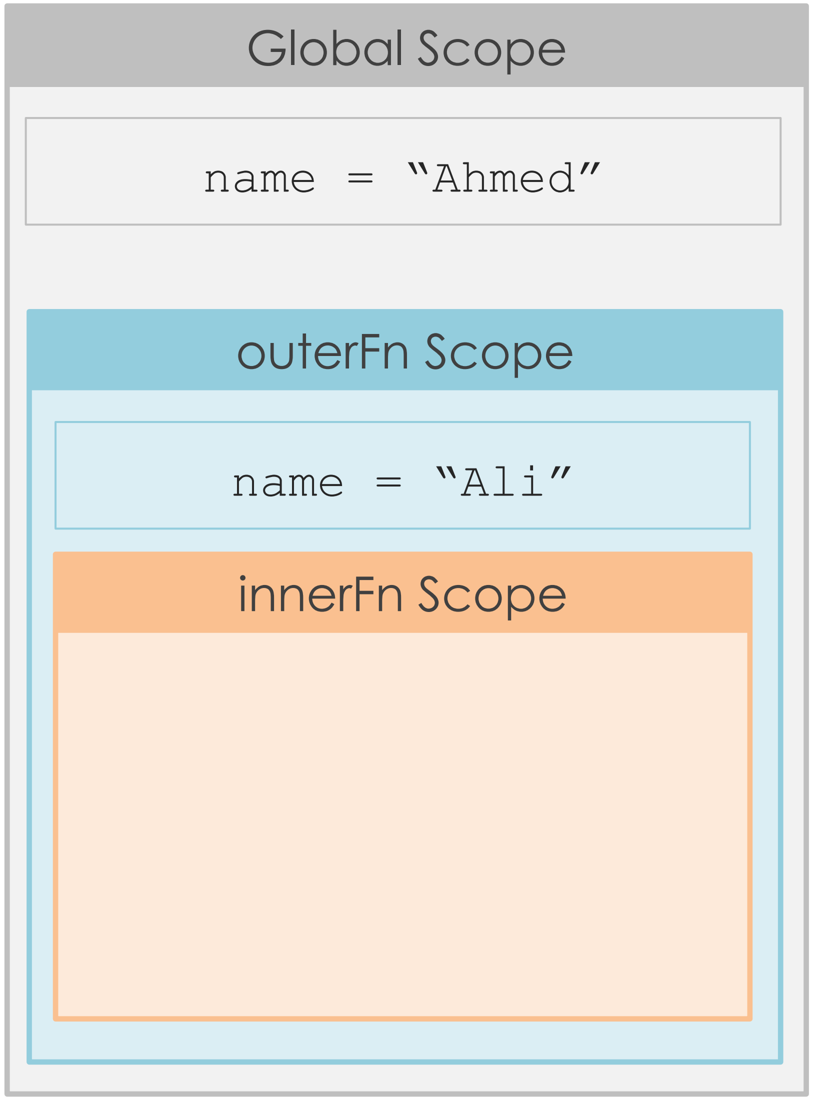


nonlocal Keyword

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

Output:

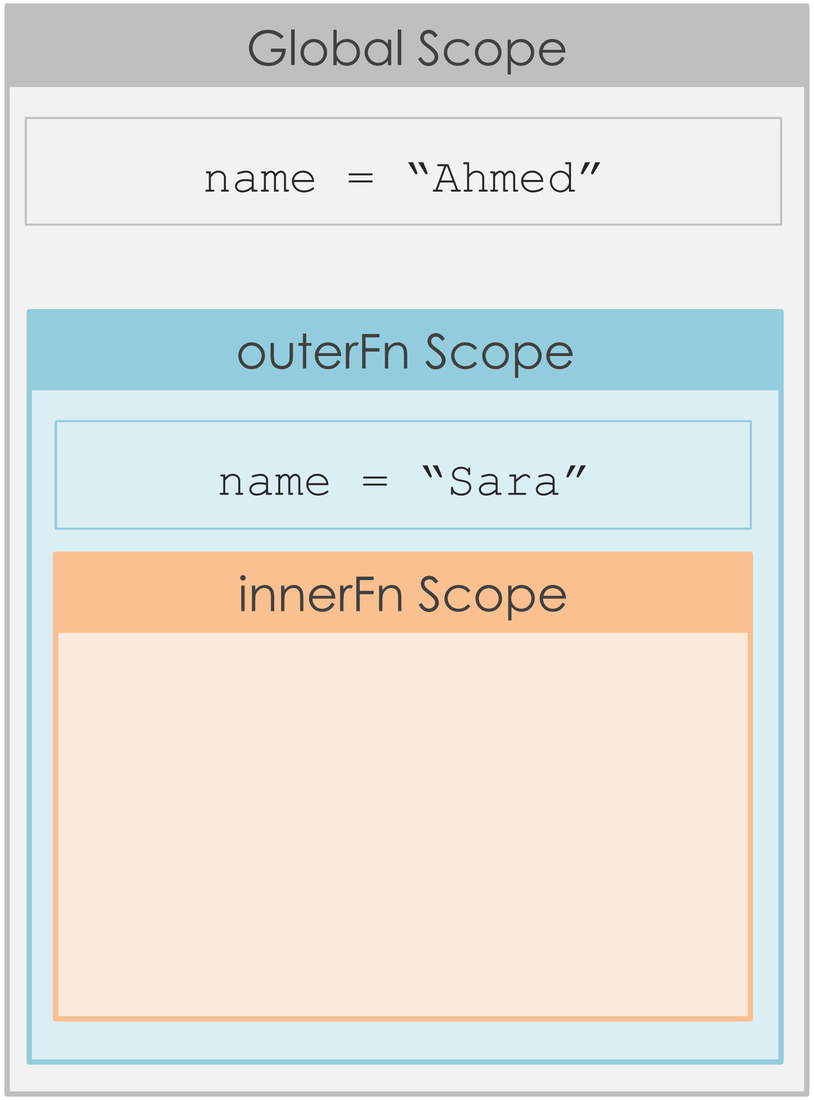
```
Ali
```



nonlocal Keyword

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

```
Output:
Ali
```

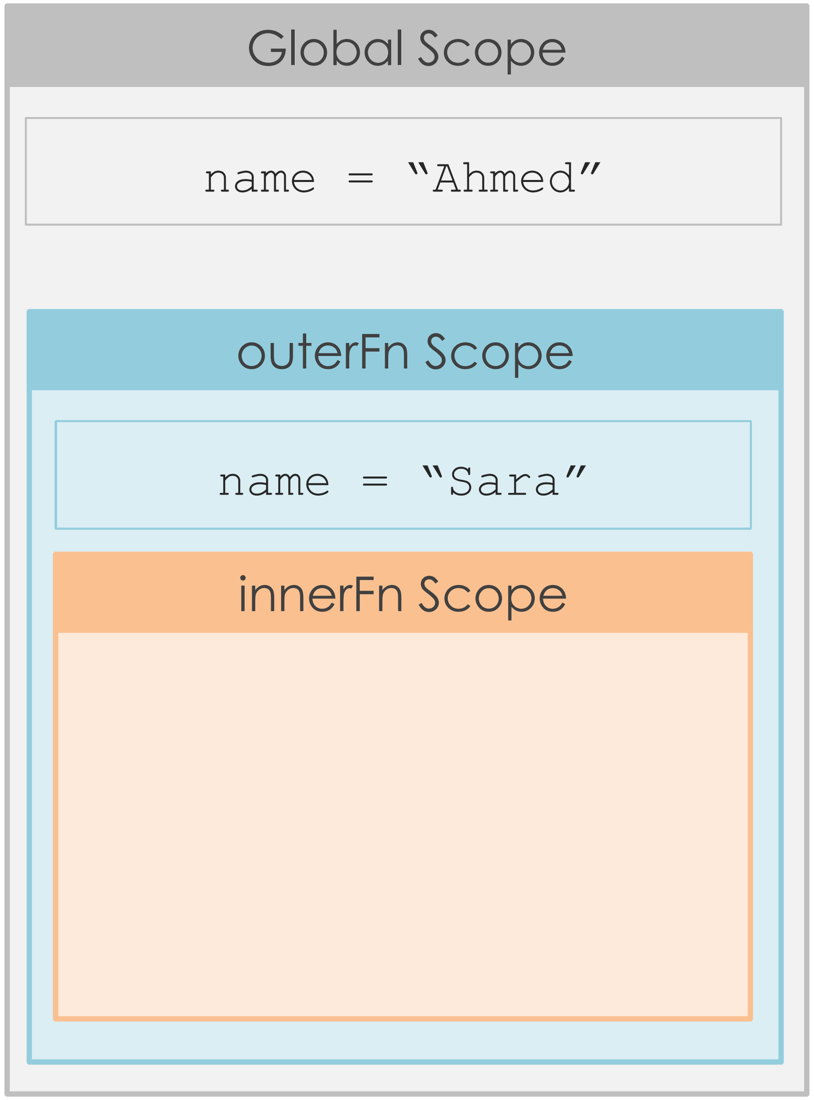


nonlocal Keyword

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

Output:

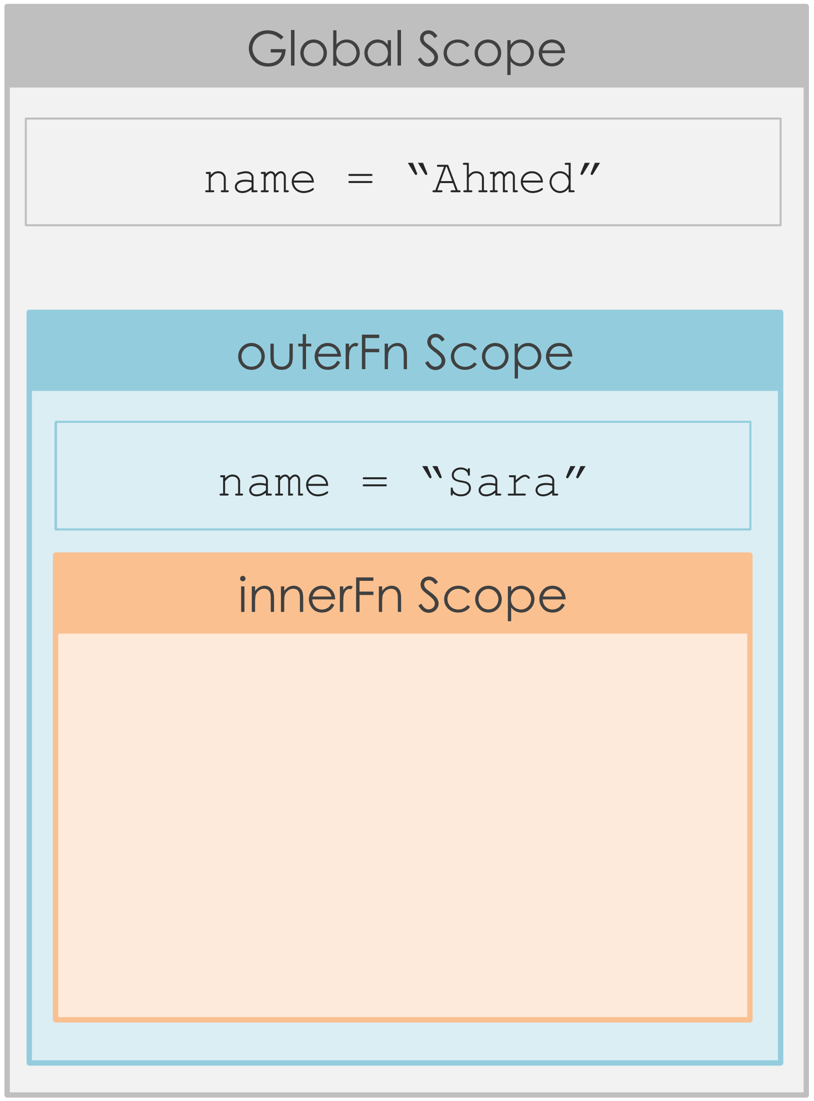
```
Ali
```



nonlocal Keyword

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

```
Output:
Ali
Sara
```



Tips and Tricks

Shorthand If

```
Do a command if this condition is true else do other command
```

Example

```
canFly = True
bird = “Dove” if canFly else “Penguin”
# bird = “Dove”
```

Swap Variables

Traditional Way

```
x = 4
y = 5
temp = x
x = y
y = temp
```

Python Way

```
x,y = 4,5
x,y = y,x
```

Enhanced Print

```
print(“I’m”, end=“ ”)
print(“Ahmed”, end=“. ”)
print(“I”, “love”, “python”)
```

```
Output:
I’m Ahmed. I Love Python
```

join and split

```
":".join(["1","Ali","grp"])     # colon is the separator
# '1:Ali:grp'
" ".join("ITI")                 # space is the separator
# 'I T I'
"Sara Mohamed".split(" ")       # space is the delimiter
# ["Sara" , "Mohamed"]
"django:flask".split(":")       # colon is the delimiter
# ["django" , "flask"]
```

is vs ==

```
True == 1                     # True

True is 1                     # False

list1 = [1,2,3]

list2 = [1,2,3]

list1 == list2                # True

list1 is list2                # False
```

Sequence Unpacking

```
l = [1,13,3,7]

a,b,c,d = l

# a=1,b=13,c=3,d=7

a,*b,c = l

# a=1,b=[13,3],c=7
```

enumerate Function

```
languages = [“JavaScript”, “Python”, “Java”]
for i , l in enumerate(languages):
      print(“Element Value: ” , l, end=“, ”)
      print(“Element Index: ” , i)
```

```
Output:
Element Value: JavaScript, Element index: 0
Element Value: Python, Element index: 1
Element Value: Java, Element index: 2
```

all & any

all check if all items in an iterable are truthy value.
any check if one item at least in an iterable is truthy value.

```
L = [0,5,9,7,8]
all(L)                              #False
any(L)                              #True
```

Thank You