

IT5001 Software Development Fundamentals

5. Recursion Vs Iterations and Nested Functions

Sirigina Rajendra Prasad

Recursion vs Iteration

Reversing a String

- How about reversing a string? Of course, we can just use string slicing

```
>>> s = 'abcde12345'  
>>> s[::-1]  
'54321edcba'  
>>>
```

- How about we write a function for it?

```
>>> reverseStringI(s)  
'54321edcba'  
>>>
```

Reverse String (Iterative Version 1)

```
def reverseStringI(s):  
    output = ''  
    l = len(s)  
    for i in range(l):  
        output += s[l-i-1]  
    return output
```

```
>>> reverseStringI('abcde')  
'edcba'
```

i
0
1
2
3
4

Reverse String (Iterative Version 1)

```
def reverseStringI(s):  
    output = ''  
    l = len(s)  
    for i in range(l):  
        output += s[l-i-1]  
    return output
```

```
>>> reverseStringI('abcde')  
'edcba'
```

i	l-i-1
0	4
1	3
2	2
3	1
4	0

Reverse String (Iterative Version 1)

```
def reverseStringI(s):  
    output = ''  
    l = len(s)  
    for i in range(l):  
        output += s[l-i-1]  
    return output
```

```
>>> reverseStringI('abcde')  
'edcba'
```

i	l-i-1	s[l-i-1]
0	4	e
1	3	d
2	2	c
3	1	b
4	0	a

Reverse String (Iterative Version 1)

```
def reverseStringI(s):  
    output = ''  
    l = len(s)  
    for i in range(l):  
        output += s[l-i-1]  
    return output
```

```
>>> reverseStringI('abcde')  
'edcba'
```

i	l-i-1	s[l-i-1]	output
0	4	e	e
1	3	d	ed
2	2	c	edc
3	1	b	edcb
4	0	a	edcba

Reverse String (Iterative Version 2)

```
def reverseStringI(s):  
    output = ''  
    for c in s:  
        output = c + output  
    return output
```

```
>>> reverseStringI('abcde')  
'edcba'
```

c	output
a	a
b	ba
c	cba
d	dcba
e	edcba

Reversing String (Recursive Version)

```
def reverseStringR(s):  
    if not s:  
        return ''  
    return reverseStringR(s[1:])+s[0]
```

- reverseStringR('abcde')
- reverseStringR('bcde')+'a'
- reverseStringR('cde')+'b'+'a'
- reverseStringR('de')+'c'+'b'+'a'
- reverseStringR('e')+'d'+'c'+'b'+'a'
- reverseStringR('')+'e'+'d'+'c'+'b'+'a'
- ''+'e'+'d'+'c'+'b'+'a'
- 'edcba'

Taylor Series

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad \text{for all } x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots \quad \text{for all } x$$

$$\tan x = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1} = x + \frac{x^3}{3} + \frac{2x^5}{15} + \dots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\sec x = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n} = 1 + \frac{x^2}{2} + \frac{5x^4}{24} + \dots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} = x + \frac{x^3}{6} + \frac{3x^5}{40} + \dots \quad \text{for } |x| \leq 1$$

$$\begin{aligned} \arccos x &= \frac{\pi}{2} - \arcsin x \\ &= \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} = \frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3x^5}{40} - \dots \end{aligned} \quad \text{for } |x| \leq 1$$

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots \quad \text{for } |x| \leq 1, x \neq \pm i$$

Taylor Series

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad \text{for all } x$$

$n=0$ $n=1$ $n=2$

- We do not need the infinite precision
- We may just sum up to k terms

$$\sin x = \sum_{n=0}^k \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad \text{for all } x$$

Computing sine by Iteration

$$\sin x = \sum_{n=0}^k \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad \text{for all } x$$

- Using iteration

```
def sinI(x, k):  
    result = 0  
    for n in range(0, k):  
        result += ((-1)**n / fact(2*n+1)) * x**(2*n+1)  
    return result
```

```
>>> print(sinI(PI/6, 10))  
0.50000000000592083
```

```
>>> from math import sin
```

```
>>> sin(PI/6)
```

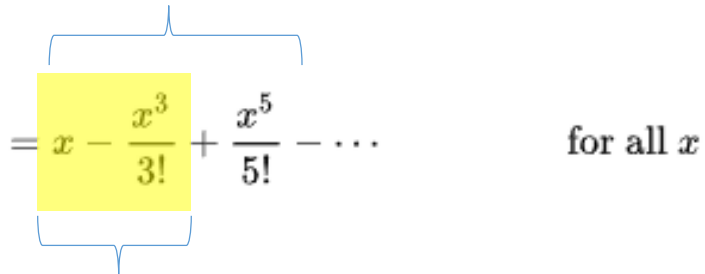
```
0.50000000000592083
```

← Python Library version of “sin()”

Computing sine by Recursion

Sum up to $n = 2$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$



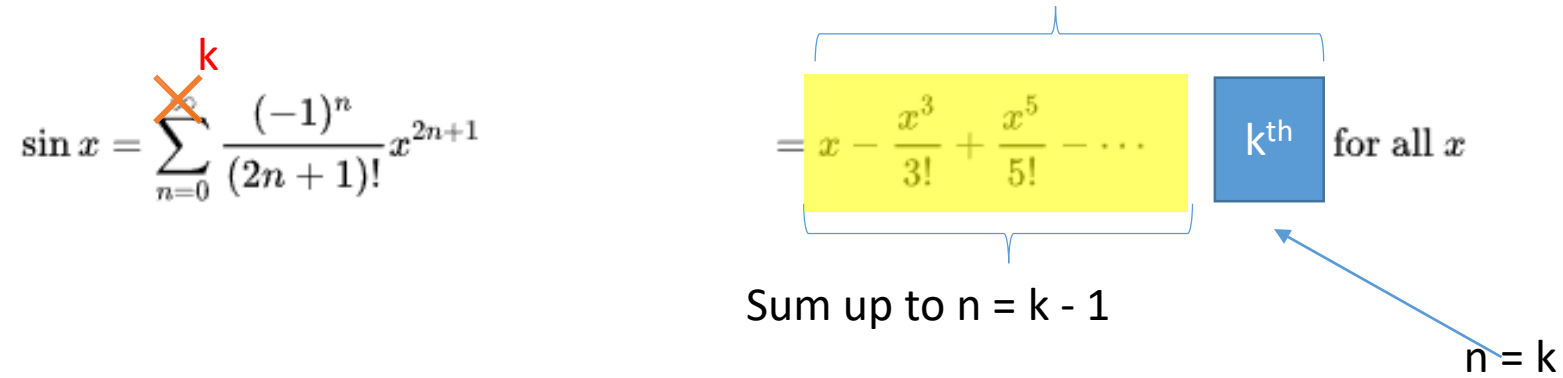
for all x

Sum up to $n = 1$

- In general, if we want to sum up to the k terms

Sum up to $n = k$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$



for all x

Sum up to $n = k - 1$

$n = k$

Computing sine by Recursion

- Assuming that if the function $\text{sinR}(x, k)$ sums until $n = k$, then

$$\text{sinR}(x, k) = \text{sinR}(x, k-1) + \text{the } k^{\text{th}} \text{ term}$$

- In general, if we want to sum up to the k terms

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Sum up to $n = k$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots$$

Sum up to $n = k - 1$

k^{th} for all x

$n = k$

Computing sine by Recursion

- Assuming that if the function $\text{sinR}(x, k)$ sums until $n = k$, then

$$\text{sinR}(x, k) = \text{sinR}(x, k-1) + \text{the } k\text{th term}$$

```
def sinR(x, k):  
    if k < 0:  
        return 0  
    return sinR(x, k-1) + ((-1)**k / fact(2*k+1)) * x**(2*k+1)
```

```
>>> sinR(PI/6, 6)  
0.50000000000592083  
>>> from math import sin  
>>> sin(PI/6)  
0.50000000000592083
```

More Taylor Series

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots \quad \text{for all } x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots \quad \text{for all } x$$

$$\tan x = \sum_{n=1}^{\infty} \frac{B_{2n}(-4)^n(1-4^n)}{(2n)!} x^{2n-1} = x + \frac{x^3}{3} + \frac{2x^5}{15} + \dots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\sec x = \sum_{n=0}^{\infty} \frac{(-1)^n E_{2n}}{(2n)!} x^{2n} = 1 + \frac{x^2}{2} + \frac{5x^4}{24} + \dots \quad \text{for } |x| < \frac{\pi}{2}$$

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} = x + \frac{x^3}{6} + \frac{3x^5}{40} + \dots \quad \text{for } |x| \leq 1$$

$$\begin{aligned} \arccos x &= \frac{\pi}{2} - \arcsin x \\ &= \frac{\pi}{2} - \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} = \frac{\pi}{2} - x - \frac{x^3}{6} - \frac{3x^5}{40} - \dots \end{aligned} \quad \text{for } |x| \leq 1$$

$$\arctan x = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots \quad \text{for } |x| \leq 1, x \neq \pm i$$

Recursion Common Patterns

```
def reverseStringR(s):
```

```
    if not s:  
        return ''
```

```
    return reverseStringR(s[1:]) + s[0]
```

```
def sinR(x, k):
```

```
    if k < 0:
```

```
        return 0
```

```
    return sinR(x, k-1) + ((-1)**k / fact(2*k+1)) * x**(2*k+1)
```

→ Base cases

Recursion step to reduce the problem one-by-one ←

Iteration Common Patterns

```
def reverseStringI(s):
```

```
    output = ''
```

```
    l = len(s)
```

```
    for i in range(l):
```

```
        output += s[l-i-1]
```

```
    return output
```

```
def sinI(x,k):
```

```
    result = 0
```

```
    for n in range(0,k):
```

```
        result += ((-1)**n / fact(2*n+1)) * x**(2*n+1)
```

```
    return result
```

Accumulate element one-by-one

Initial the final answer to “nothing” at the beginning.
Accumulate and return the final answer

Iteration/Recursion Conversion

```
def sinR(x,k):  
    if k < 0:  
        return 0  
    return sinR(x,k-1) + ((-1)**k / fact(2*k+1)) * x**(2*k+1)
```

Base case

The answer for previous $k - 1$ terms

The k th term

```
def sinI(x,k):  
    result = 0  
    for n in range(0,k):  
        result += ((-1)**n / fact(2*n+1)) * x**(2*n+1)  
    return result
```

Iteration/Recursion Conversion

```
def reverseStringR(s):  
    if not s:  
        return ''  
    return reverseStringR(s[1:]) + s[0]
```

Base case

The answer for previous $k - 1$ terms

The k th term

```
def reverseStringI(s):  
    output = ''  
    l = len(s)  
    for i in range(l):  
        output += s[l-i-1]  
    return output
```

“Homework”

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1}$$

$$= x + \frac{x^3}{6} + \frac{3x^5}{40} + \dots \quad \text{for } |x| \leq 1$$

- The answer for all k-1 terms?
- Base case?
- Kth term?

Another Example

Recursion vs Iteration

- SumDigits

- Given a positive number n , the sum of all digits is obtained by adding the digit one-by-one
 - For example, the sum of 52634 = $5 + 2 + 6 + 3 + 4 = 20$
- Write a function `sum(n)` to compute the sum of all the digits in n

- Factorial

- Factorial is defined (recursively) as $n! = n * (n - 1)!$ such that $0! = 1$
- Write a function `fact(n)` to compute the value of $n!$

- Can you do it in both recursion and iteration?

SumDigits

Iteration

```
def sum(n):  
    res = 0  
    while n > 0:  
        res = res + n%10  
        n = n//10  
    return res
```

base/initial value

computation

continuation/next value

Recursion

```
def sum(n):  
    if n == 0:  
        return 0  
    else:  
        return n%10 + sum(n//10)
```

stop/base case (*they are related, how?*)

temporary result variables
not needed in recursion (*why?*)

Factorial

Iteration

```
def fact(n):  
    res = 1  
    while n > 0:  
        res = res * n  
        n = n-1  
    return res
```

base/initial value

computation

continuation/next value

Recursion

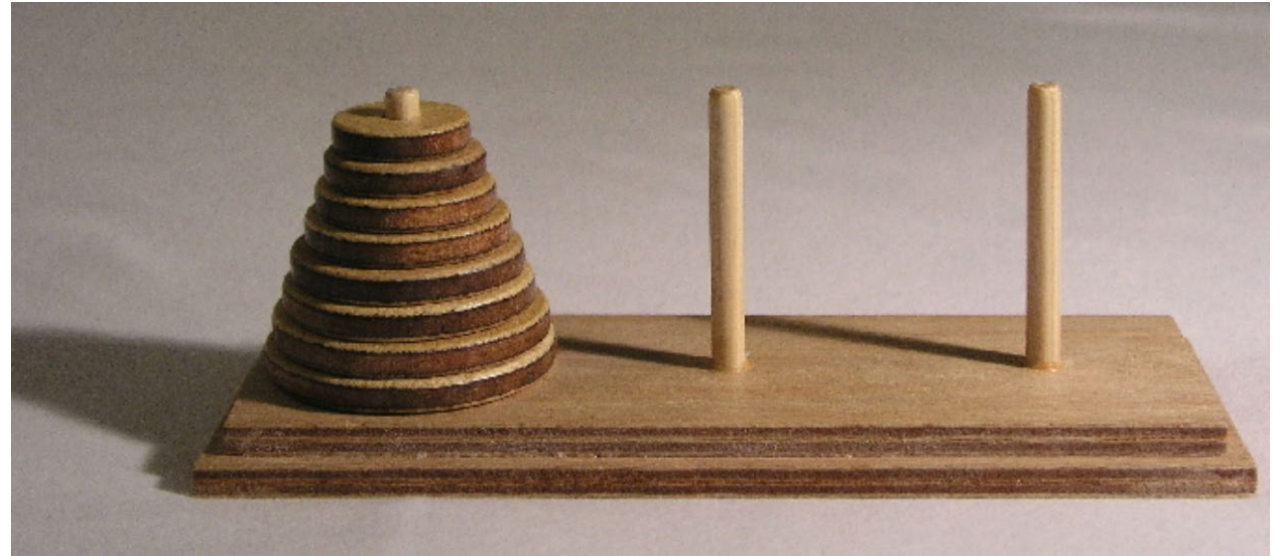
```
def fact(n):  
    if n == 0:  
        return 1  
    else:  
        return n * fact(n-1)
```

stop/base case (*they are related, how?*)

temporary result variables
not needed in recursion (*why?*)

Towers of Hanoi Problem

- Move entire stack to the last tower while obeying the following rules
 - Only one disc may be moved at a time
 - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
 - No disk may be placed on top of a disk that is smaller than it.



https://en.wikipedia.org/wiki/Tower_of_Hanoi

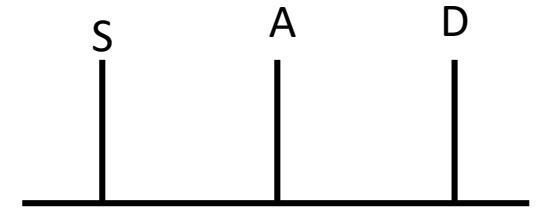
Question: How many moves for n discs?

$$2^n - 1$$

Towers of Hanoi

```
def towers_Hanoi(n, S, D, A):  
    #Base case  
    if n == 1:  
        return print(f'Move Disc {n} from Tower {S} to Tower {D}')  
  
    #Move top n-1 discs to auxiliary tower  
    towers_Hanoi(n - 1, S, A, D)  
    print(f'Move Disc {n} from Tower {S} to Tower {D}')  
    #Move top n-1 discs to destination tower  
    towers_Hanoi(n - 1, A, D, S)
```

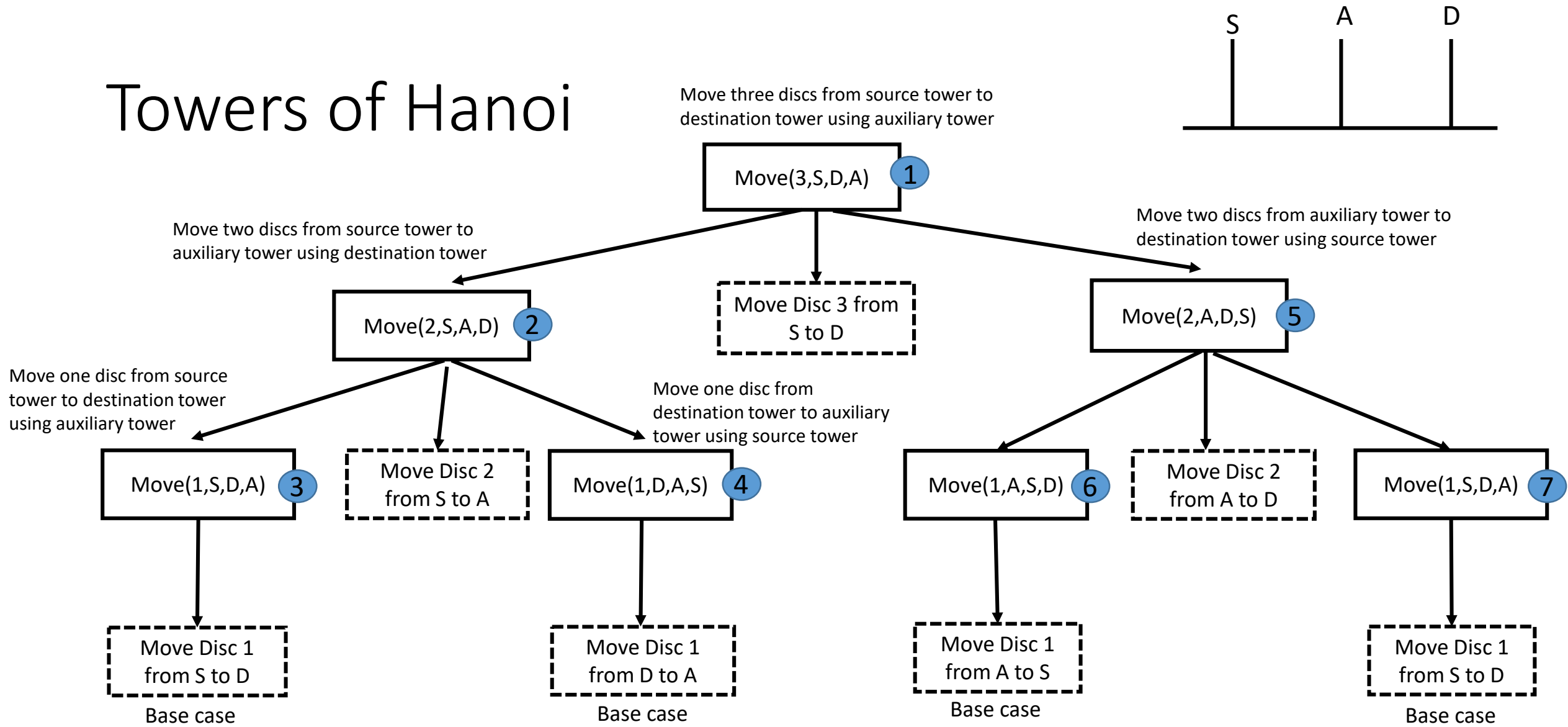
```
towers_Hanoi(3, 'S', 'D', 'A')
```



Solution:

```
Move Disc 1 from Tower S to Tower D  
Move Disc 2 from Tower S to Tower A  
Move Disc 1 from Tower D to Tower A  
Move Disc 3 from Tower S to Tower D  
Move Disc 1 from Tower A to Tower S  
Move Disc 2 from Tower A to Tower D  
Move Disc 1 from Tower S to Tower D
```

Towers of Hanoi

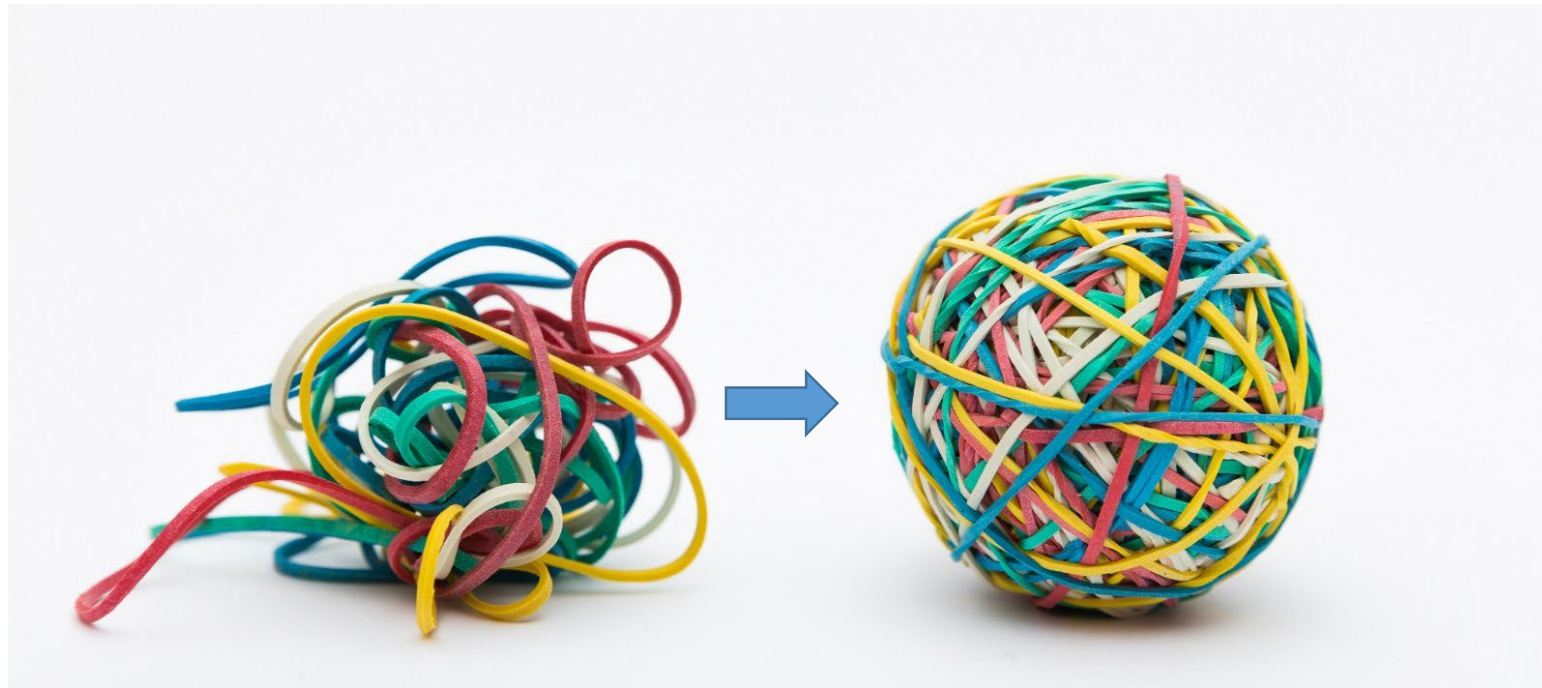


Depth-first
In-order traversal

“Homework”

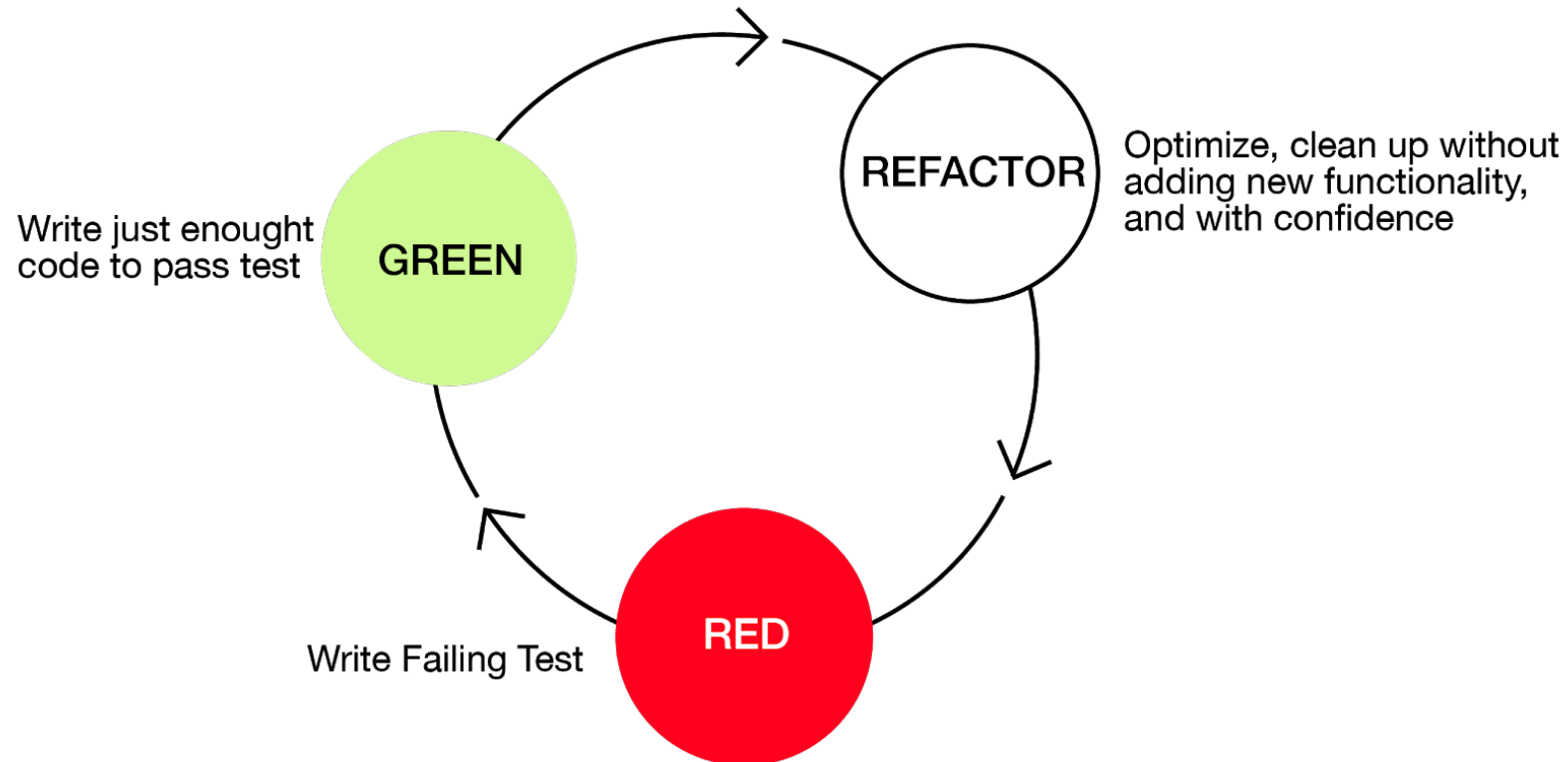
- How to re-write your code with both iterative/recursion version mentioned in this course before?
 - `burgerPrice()`
 - `checkAllAlpha()`
 - Etc.
- The answer for all $k-1$ terms?
- Base case?
- K th term?

Code Refactoring



Code Refactoring

- **Refactoring** is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.



Nested Functions

Nested Functions

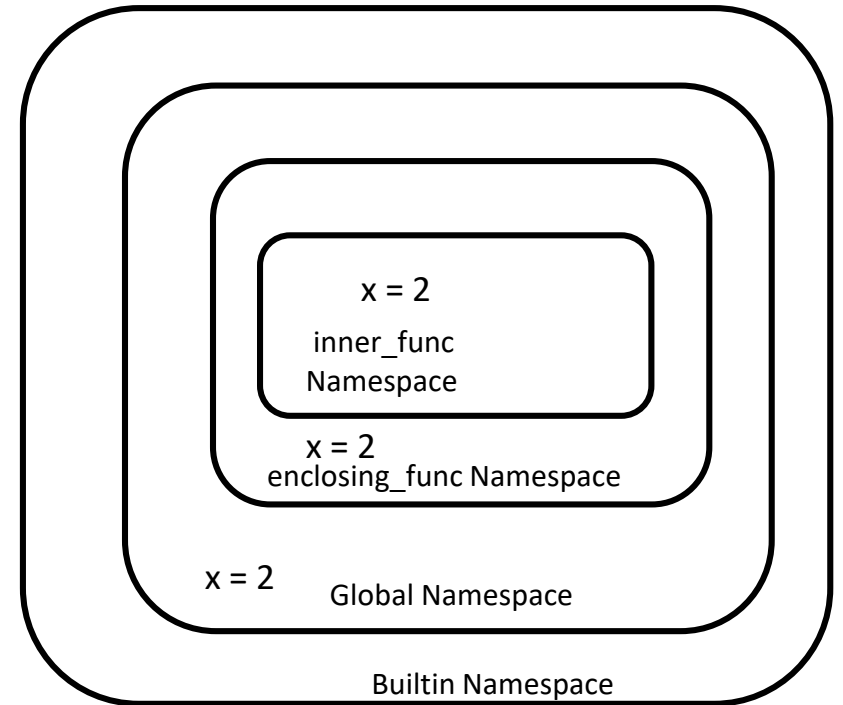
- Functions defined inside other functions

```
x = 2
def enclosing_func(x):
    def inner_func(x):
        return x**2
    output = inner_func(x)
    return output

print(enclosing_func(x))
```

Output:

4



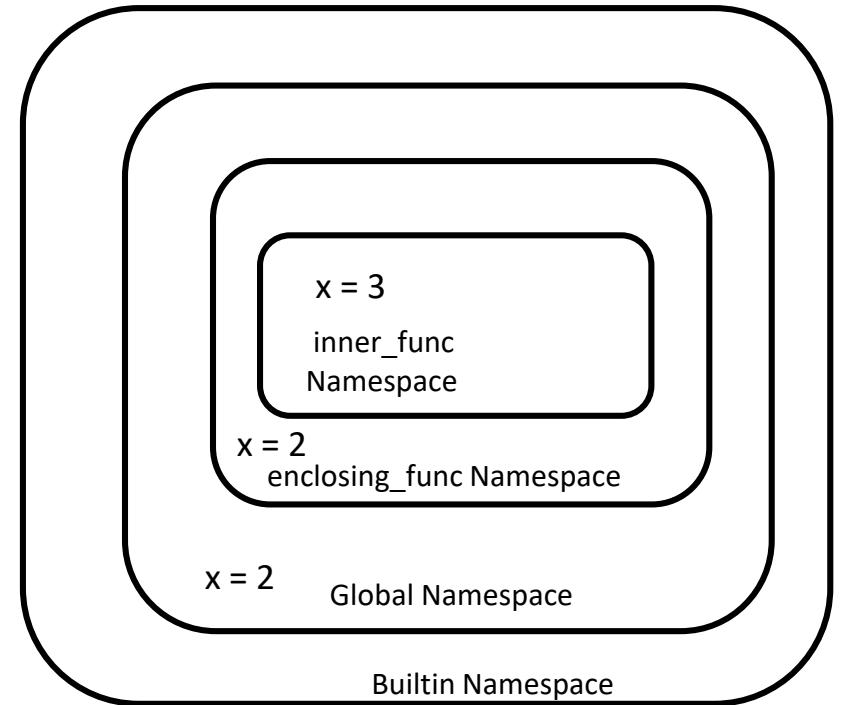
Nested Functions

Namespace of inner function is different from enclosing function and global namespace

```
def enclosing_func(x):  
    def inner_func(x):  
        x += 1  
        return x**2  
    print(x)  
    output = inner_func(x)  
    print(x)  
    return output  
  
x = 2  
print(x)  
print(enclosing_func(x))  
print(x)
```

Output:

2
2
2
9
2



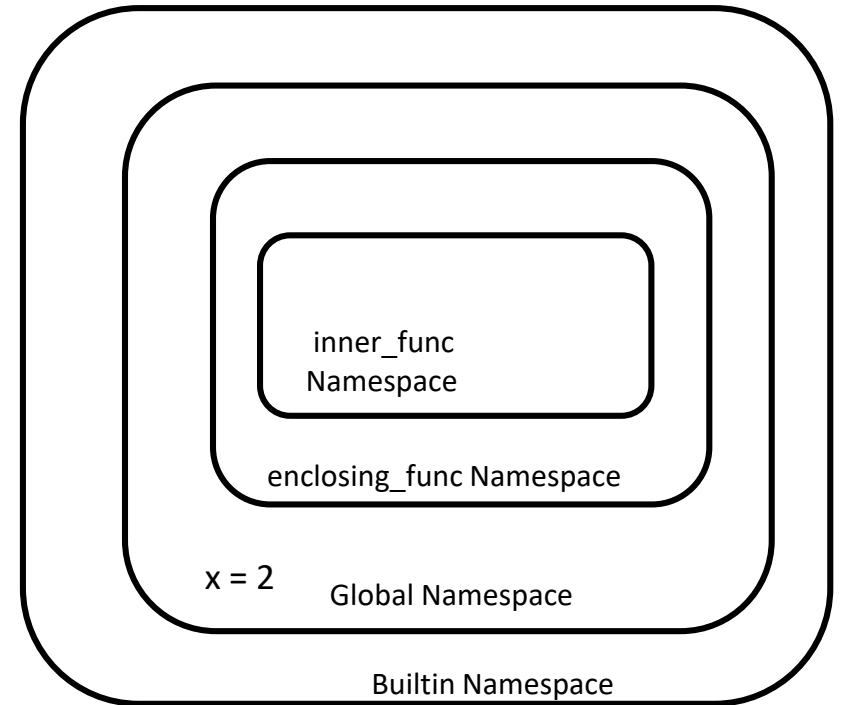
Nested Functions

Inner functions can access global variables

```
def enclosing_func():  
    def inner_func():  
        return x**2  
    output = inner_func()  
    return output  
  
x = 2  
print(enclosing_func())
```

Output:

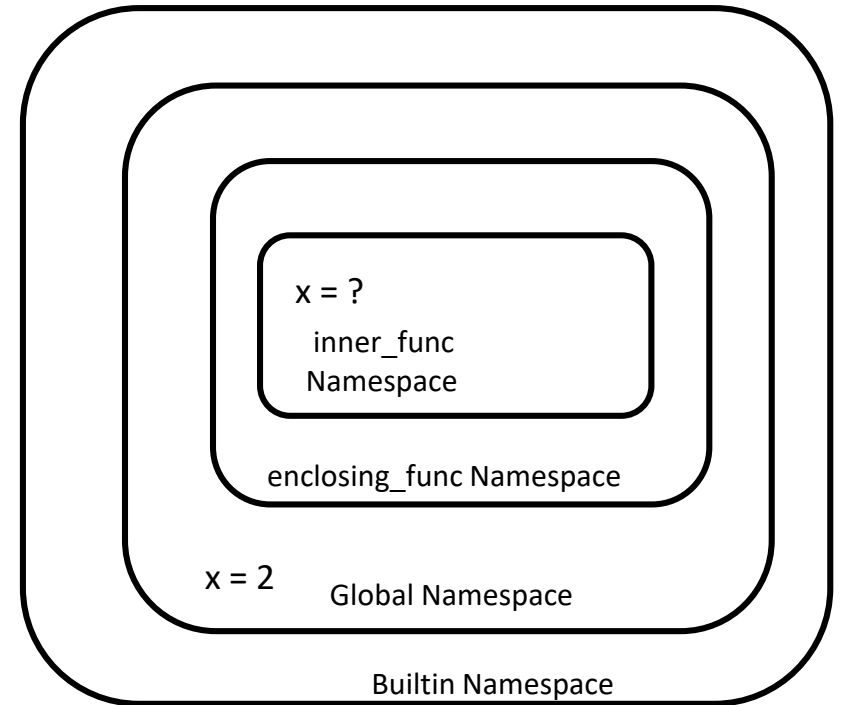
4



Nested Functions

Inner functions cannot modify global variables

```
def enclosing_func():  
    def inner_func():  
        x = x+2  
        return x**2  
    output = inner_func()  
    return output  
  
x = 2  
print(enclosing_func())
```



UnboundLocalError: local variable 'x' referenced before assignment

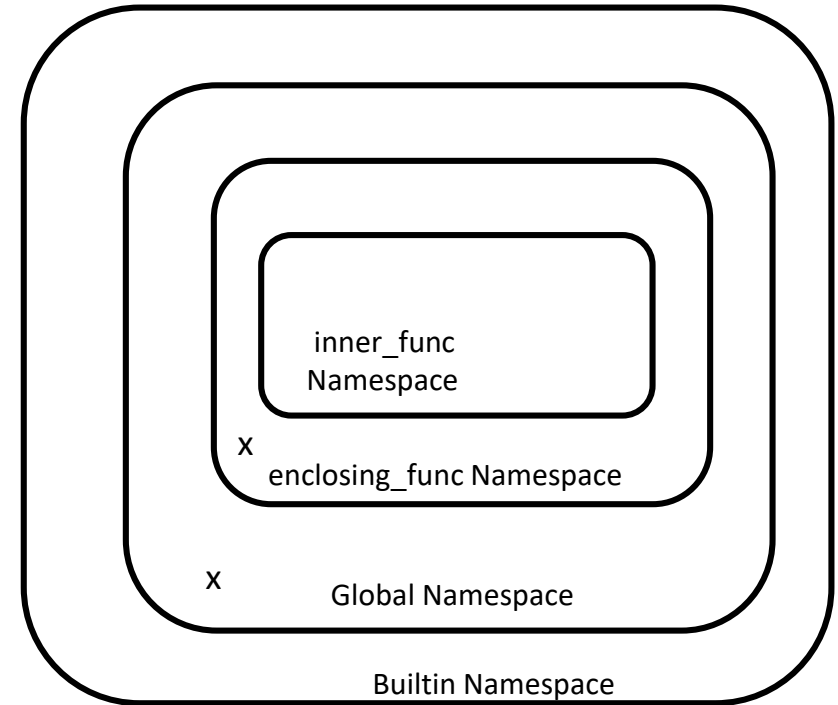
Nested Functions: *global* keyword

Modifying global variable from inner function

```
def enclosing_func():  
    x = 3  
    def inner_func():  
        global x  
        print(x)  
        x = 1  
        print(x)  
    print(x)  
    inner_func()  
    print(x)  
  
x = 5  
print(x)  
print(enclosing_func())  
print(x)
```

nonlocal/enclosing namespace for inner function

global keyword binds this variable to global variable x



Output:

```
5  
3  
5  
1  
3  
None  
1
```

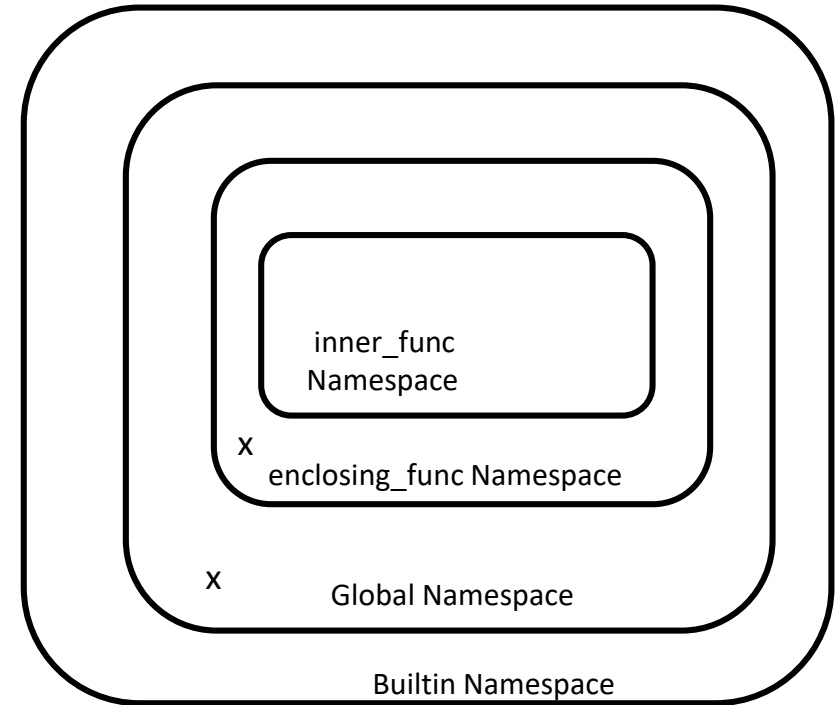
Nested Functions

Inner functions can access nonlocal variables

```
def enclosing_func():  
    x = 3  
    def inner_func():  
        print(x)  
  
    print(x)  
    inner_func()  
    print(x)  
x = 2  
print(x)  
print(enclosing_func())  
print(x)
```

Output:

```
2  
3  
3  
3  
None  
2
```



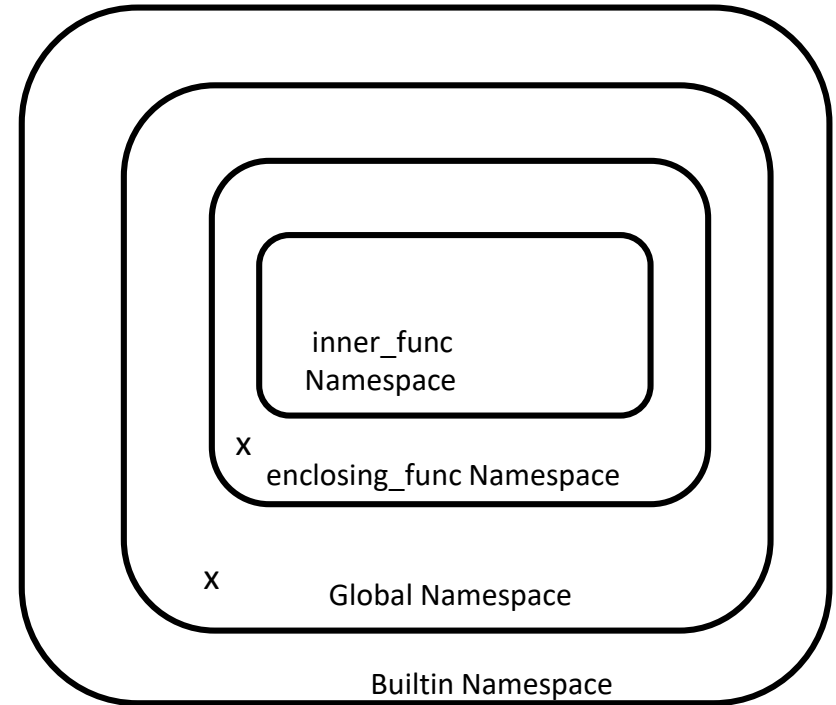
Nested Functions

Inner functions cannot modify nonlocal variables

```
def enclosing_func():  
    x = 3  
    def inner_func():  
        x = x+1  
        print(x)  
  
    print(x)  
    inner_func()  
    print(x)  
x = 2  
print(x)  
print(enclosing_func())  
print(x)
```

Output:

UnboundLocalError: local variable 'x' referenced before assignment



Nested Functions: *nonlocal* keyword

Names in enclosing_func namespace
are nonlocal variables for inner_func

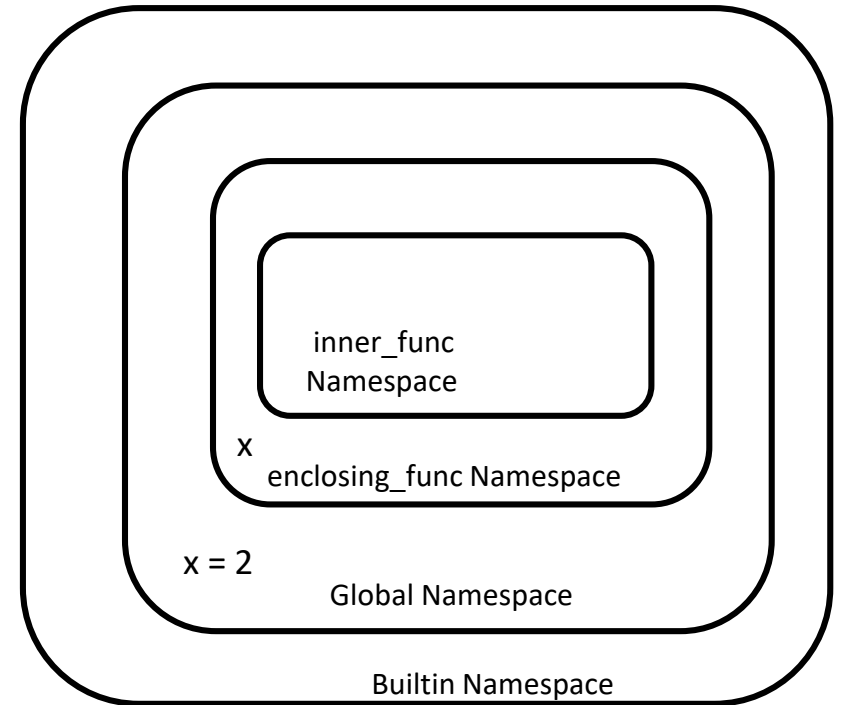
```
def enclosing_func():  
    x = 3  
    def inner_func():  
        nonlocal x  
        print(x)  
        x = x+1  
        print(x)  
    print(x)  
    inner_func()  
    print(x)
```

Binds this name to variable
in nearest enclosing namespace

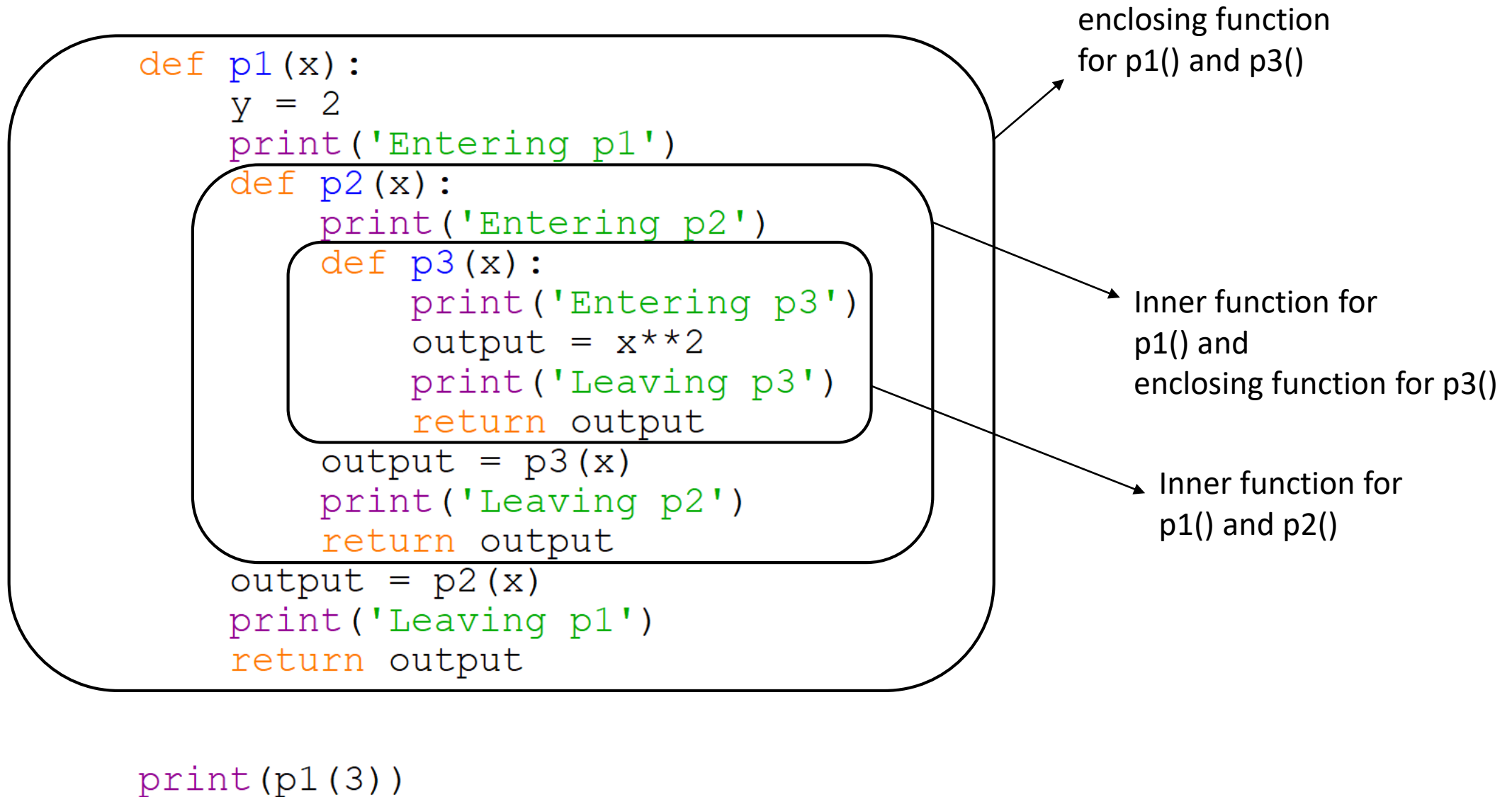
```
x = 2  
print(x)  
print(enclosing_func())  
print(x)
```

Output:

```
2  
3  
3  
4  
4  
None  
2
```

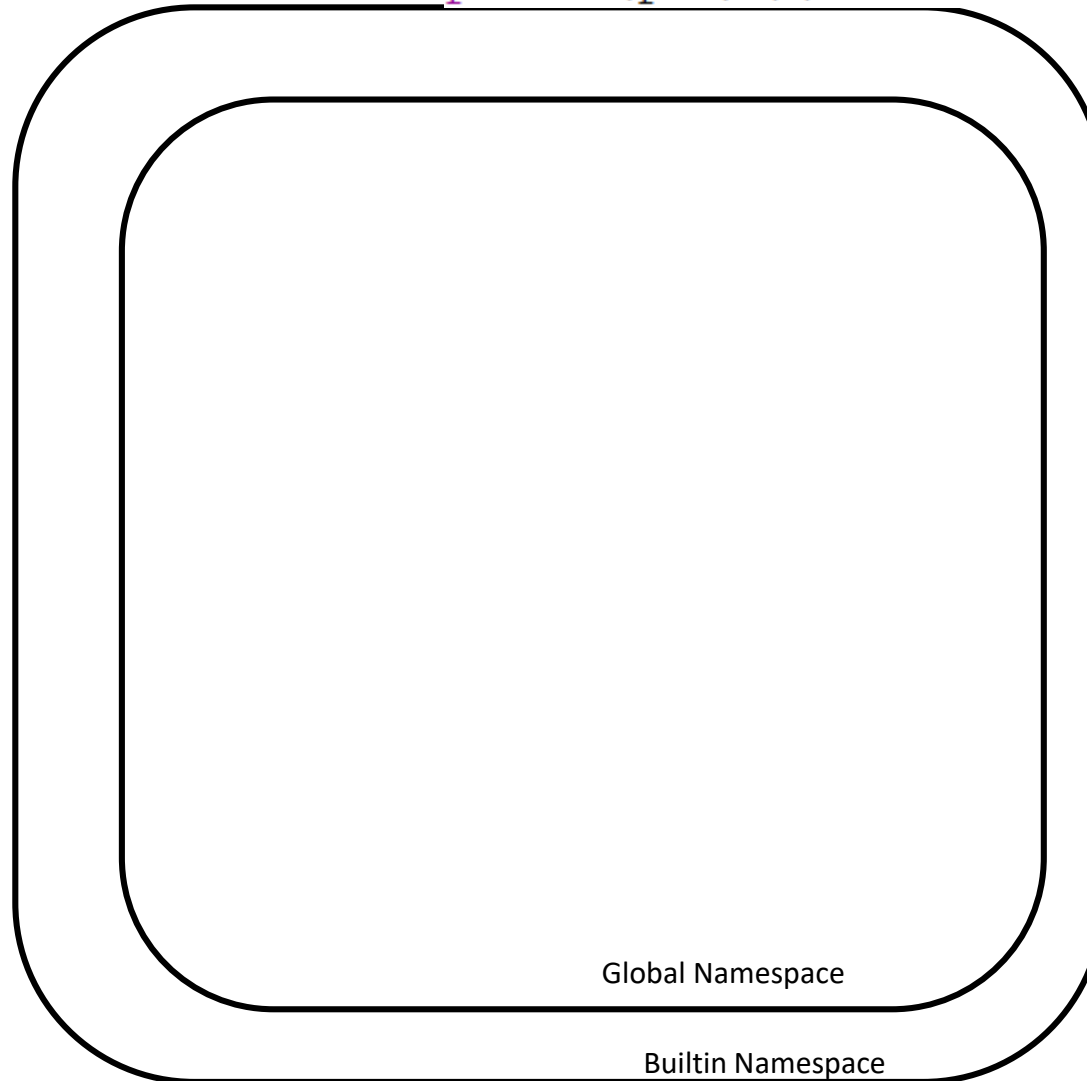


Nested Functions

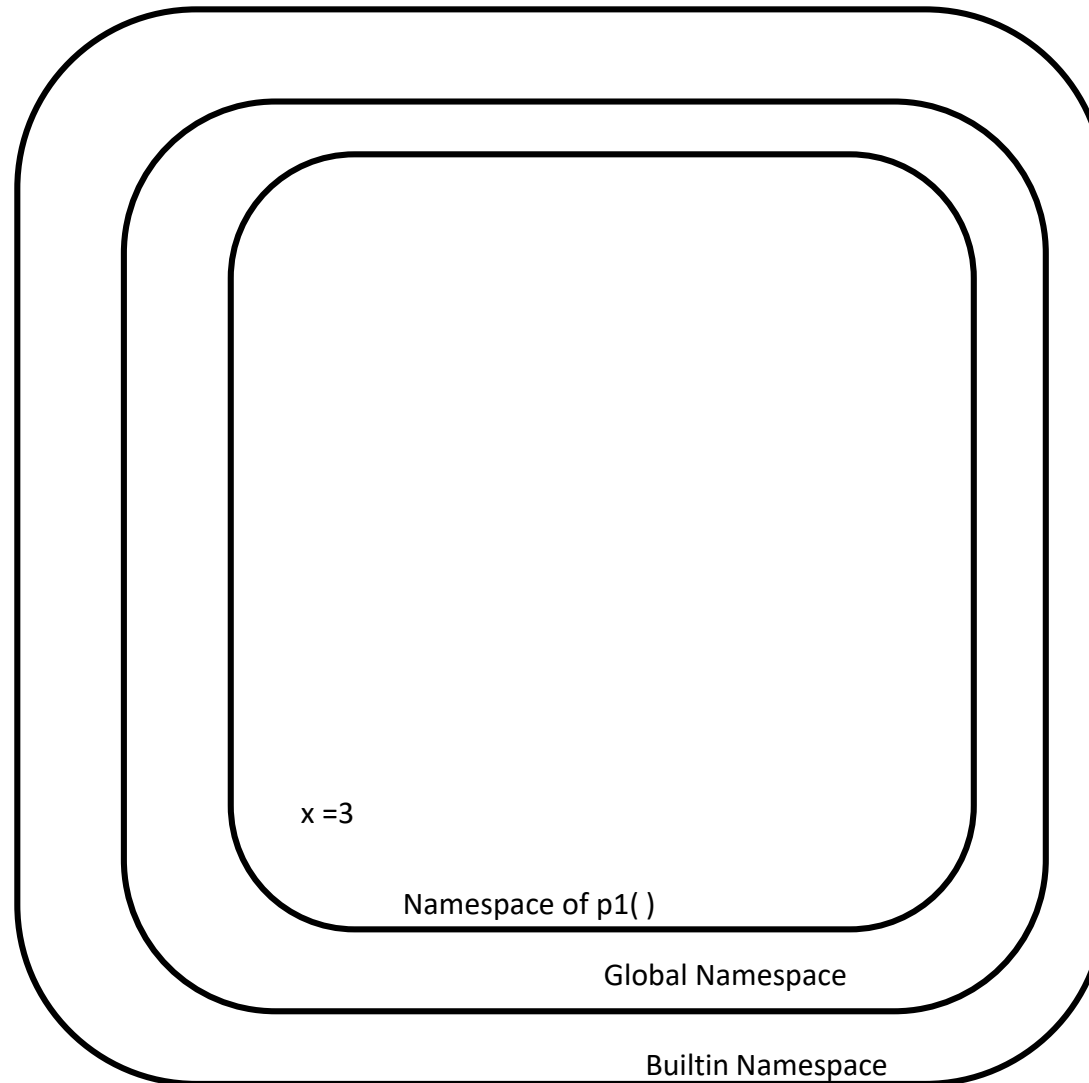


Namespaces: Nested Functions

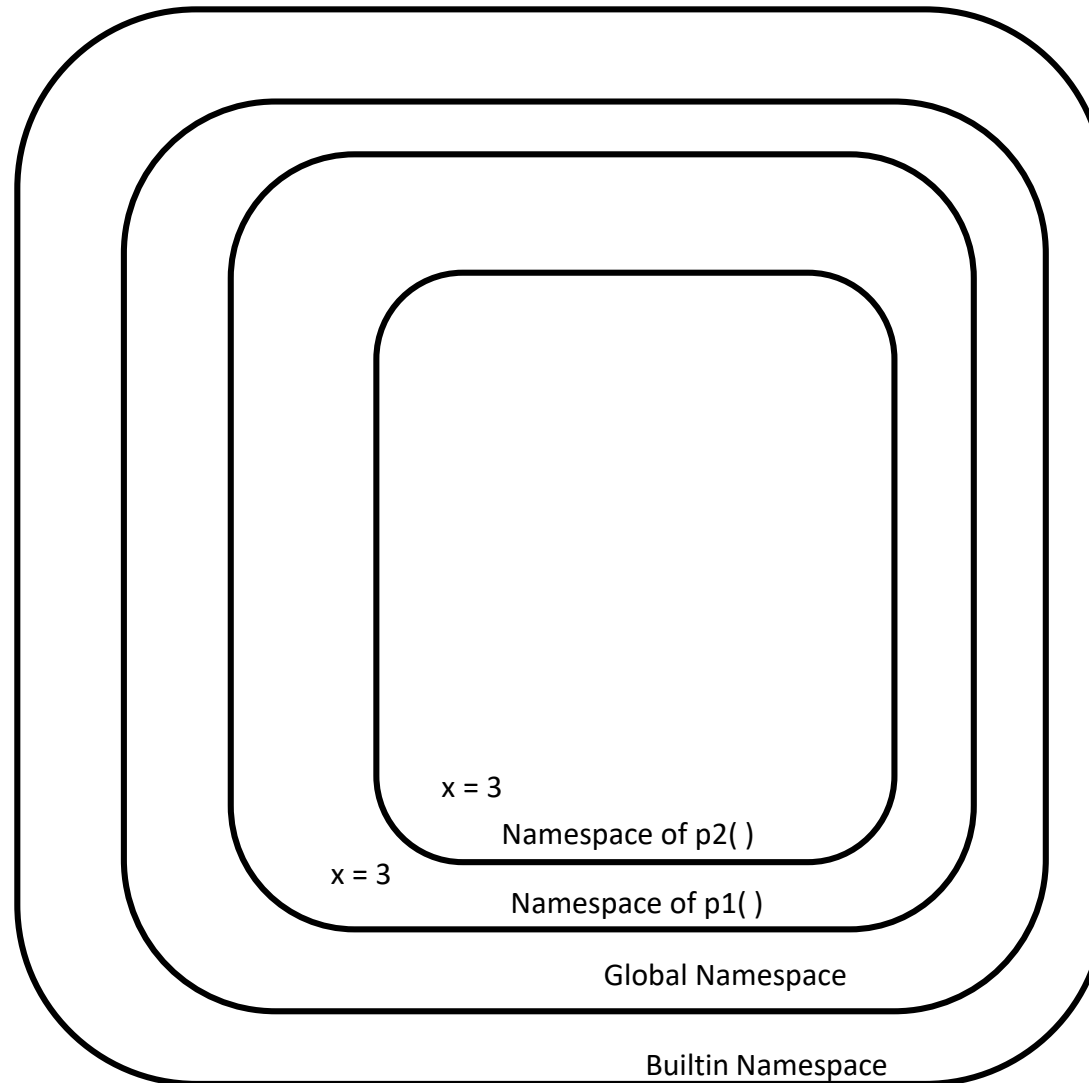
```
print(p1(3))
```



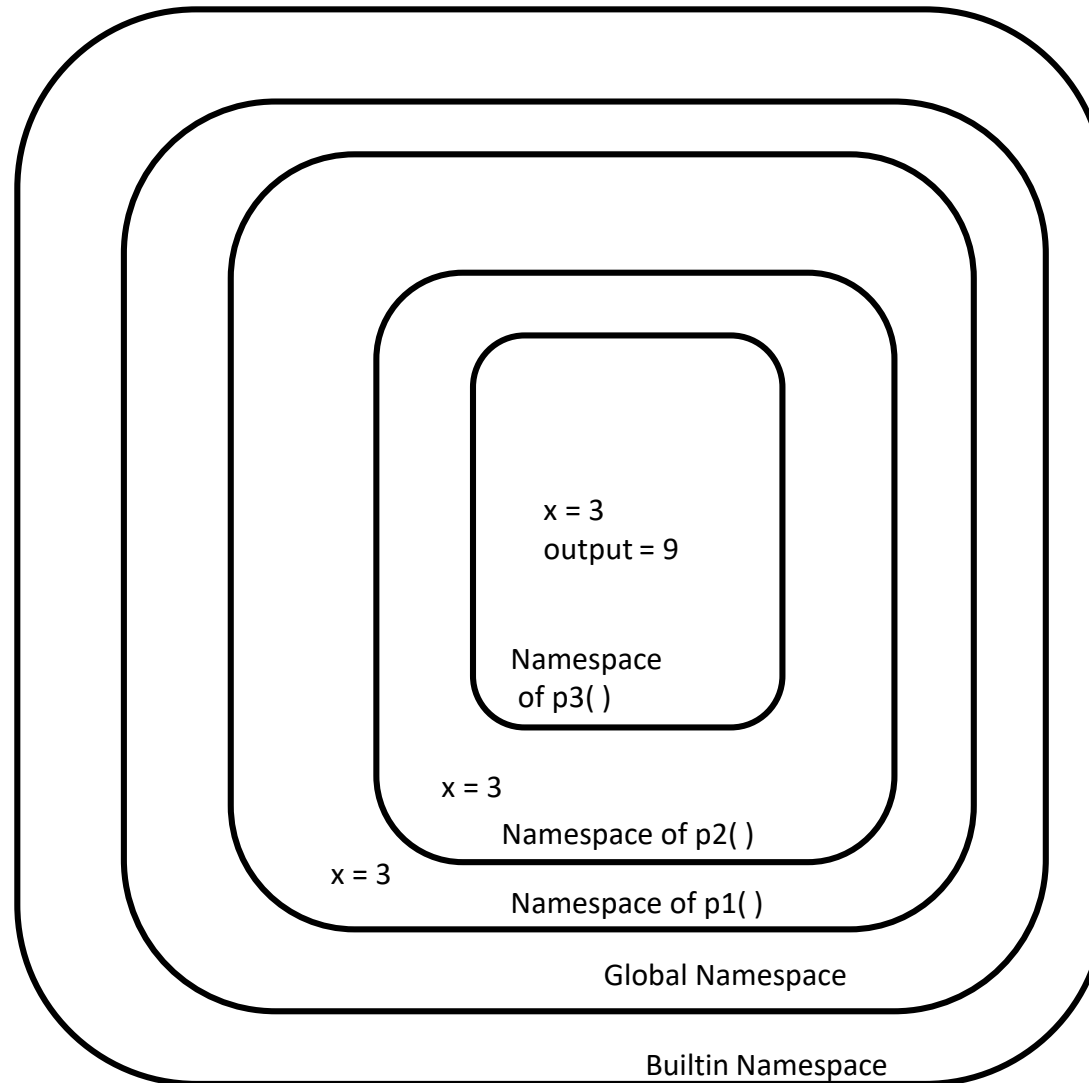
Namespaces: Nested Functions



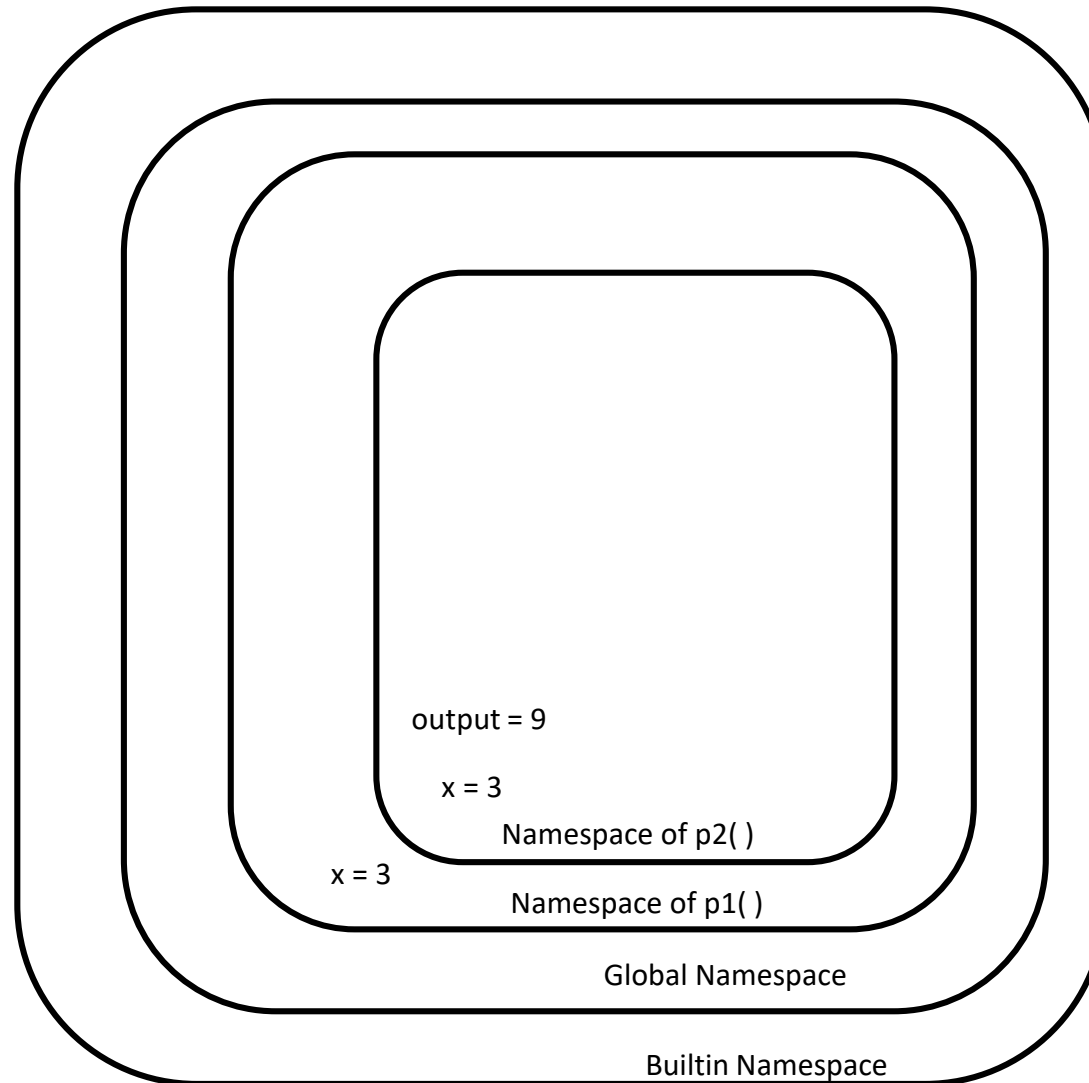
Namespaces: Nested Functions



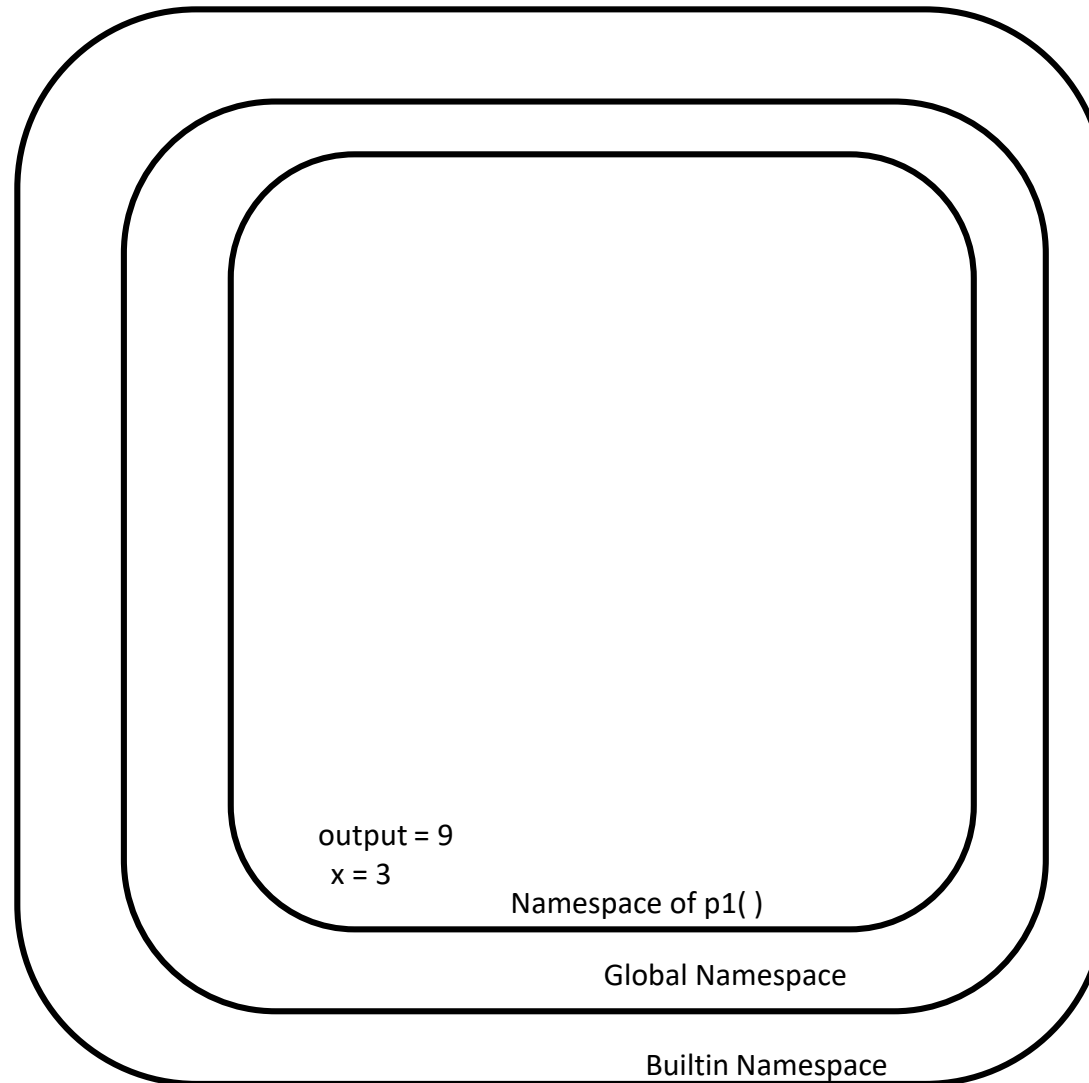
Namespaces: Nested Functions



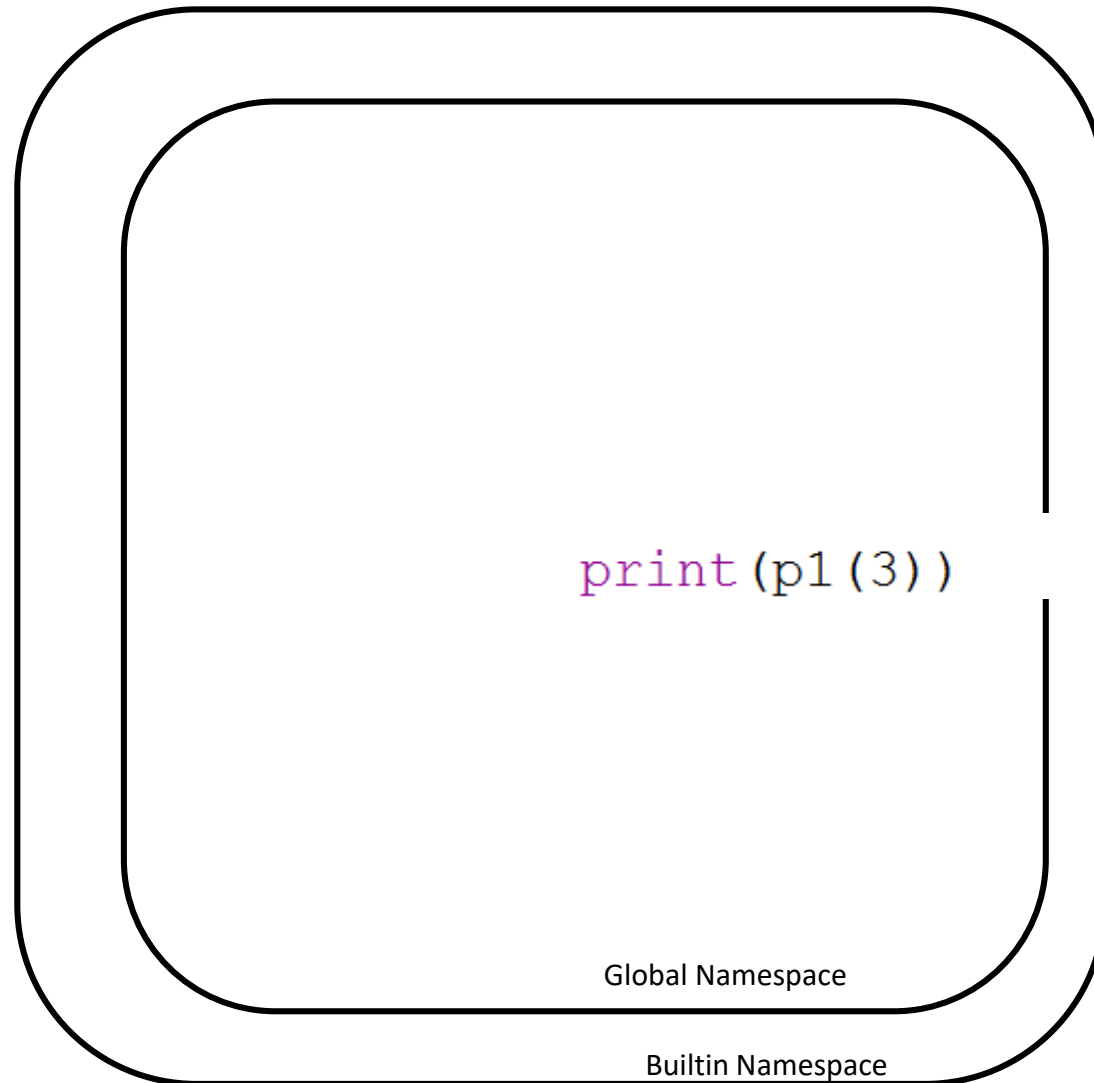
Namespaces: Nested Functions



Namespaces: Nested Functions



Namespaces: Nested Functions



What is the output?

```
def p1(x):  
    y = 2  
    print('Entering p1')  
    def p2(x):  
        print('Entering p2')  
        z = 4  
        def p3(x):  
            print('Entering p2')  
            output = x**2 + y**2 + z**2  
            print('Leaving p3')  
            return output  
        output = p3(x)  
        print('Leaving p2')  
        return output  
    output = p2(x)  
    print('Leaving p1')  
    return output  
  
print(p1(3))
```

What is the output?

```
def p1(x):  
    y = 2  
    print('Entering p1')  
    def p2(x):  
        print('Entering p2')  
        z = 4  
        def p3(x):  
            print('Entering p2')  
            output = x**2  
            print('Leaving p3')  
            return output  
        output = p3(x)  
        print('Leaving p2')  
        return output  
    output = p2(x)+z**2  
    print('Leaving p1')  
    return output  
  
print(p1(3))
```

Where do we use inner functions?

- Higher-Order Functions (Week 5)