Unit III: Understanding Control Structures

Programming Methodology (CSF101)

Outline

- Loops
- Functions, Call Stack, Scope and Function
- Memory Addresses and Pointers
- Call by Value and Call by Reference

How do you execute a code block 100 times?
I just copy-paste it 100 times.
imgflip.com

Loops

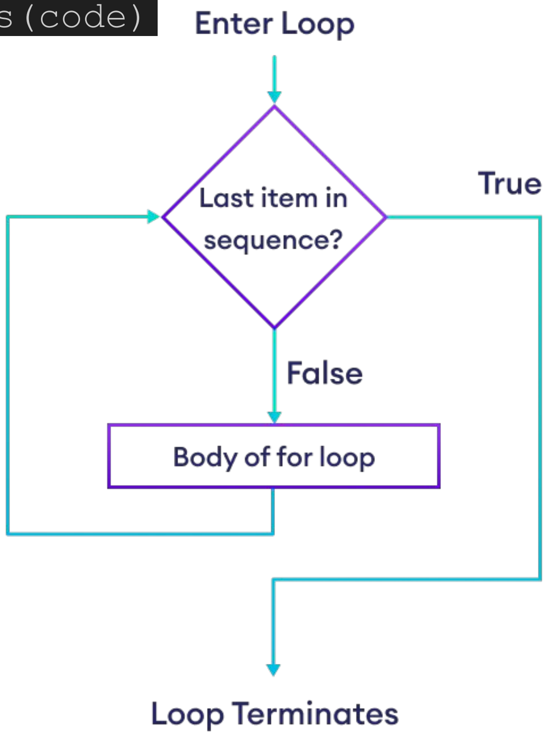
- Loops help us remove the redundancy of code when a task has to be repeated several times.
- Types of loops in python:

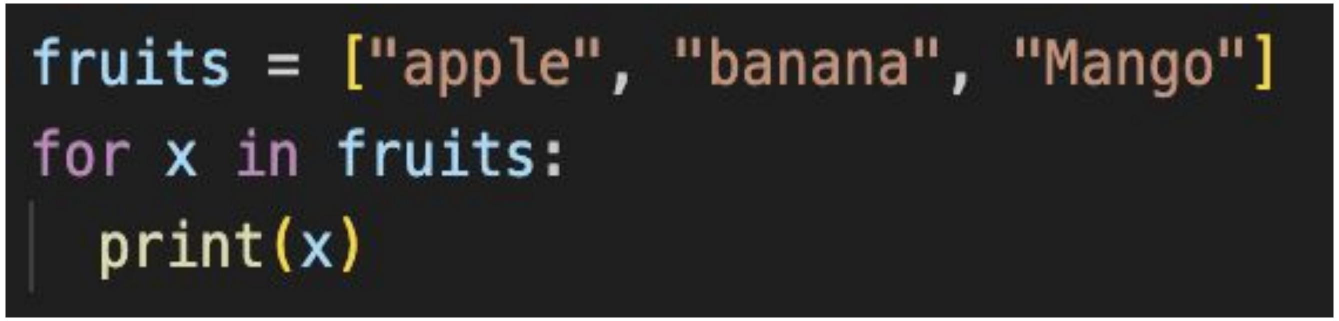
  1. For loop
  2. While loop

For Loop

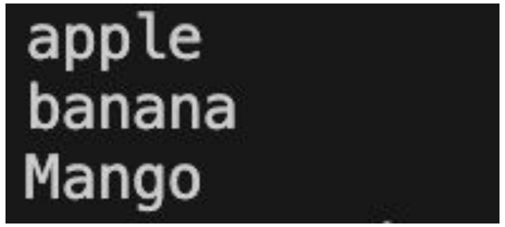
- It is used for iterating over a sequence.
- Syntax:

```
for item in sequence:
    statements(code)
```





```
fruits = ["apple", "banana", "Mango"]
for x in fruits:
    print(x)
```

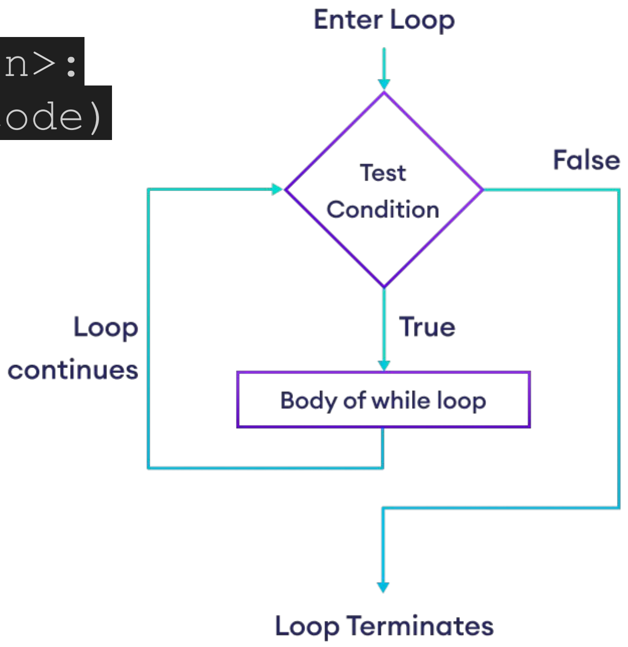


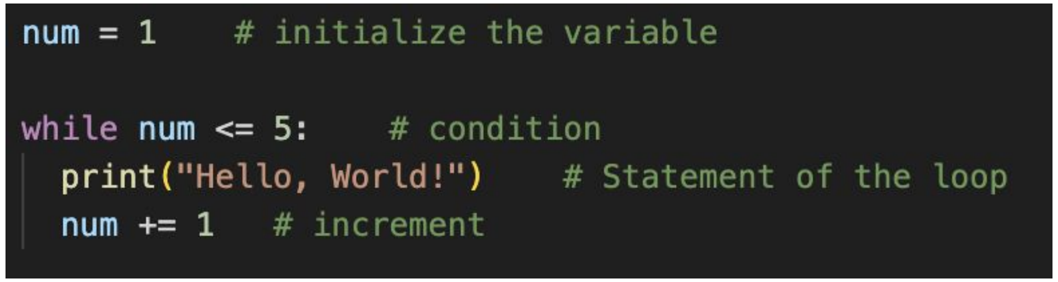
```
apple
banana
Mango
```

While Loop

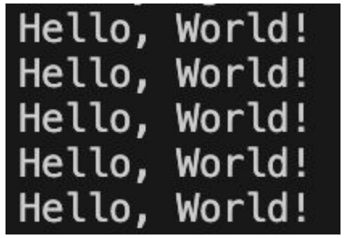
- It continually executes the statements (code) as long as the given condition is TRUE.
- Syntax:

```
while <condition>:
   statements(code)
```





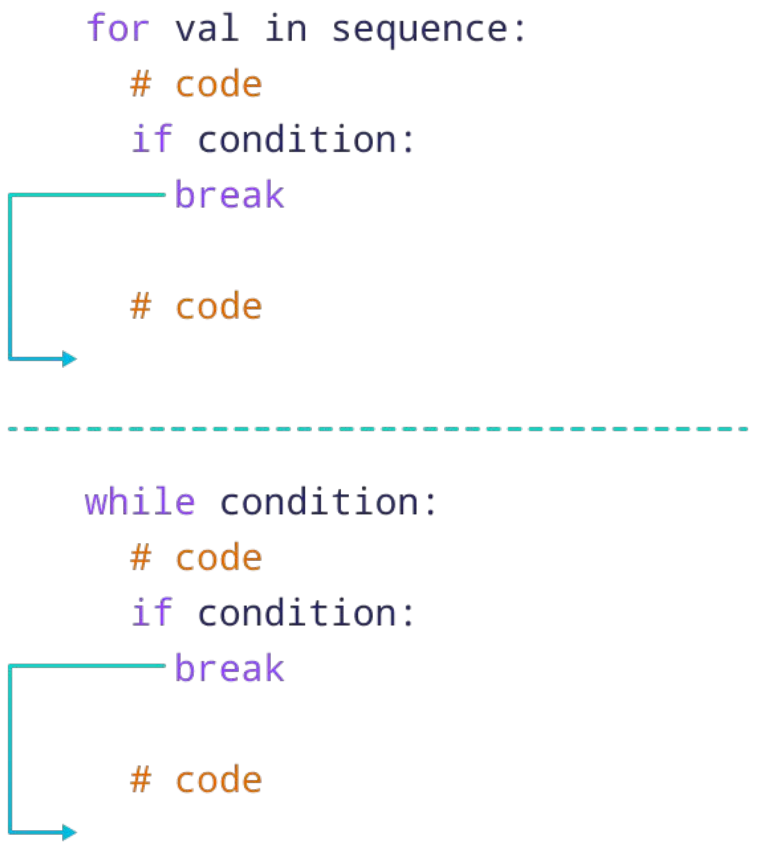
```
num = 1     # initialize the variable

while num <= 5:     # condition
    print("Hello, World!")     # Statement of the loop
    num += 1    # increment
```



```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

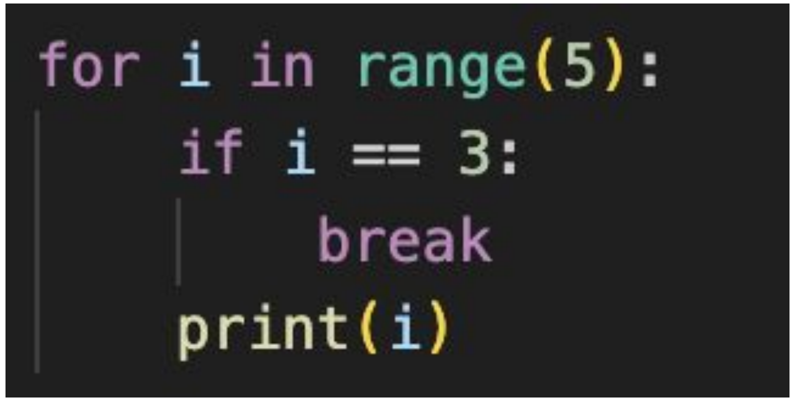
break statement

- Exits the loop entirely



```
for val in sequence:
  # code
  if condition:
    break

  # code
```

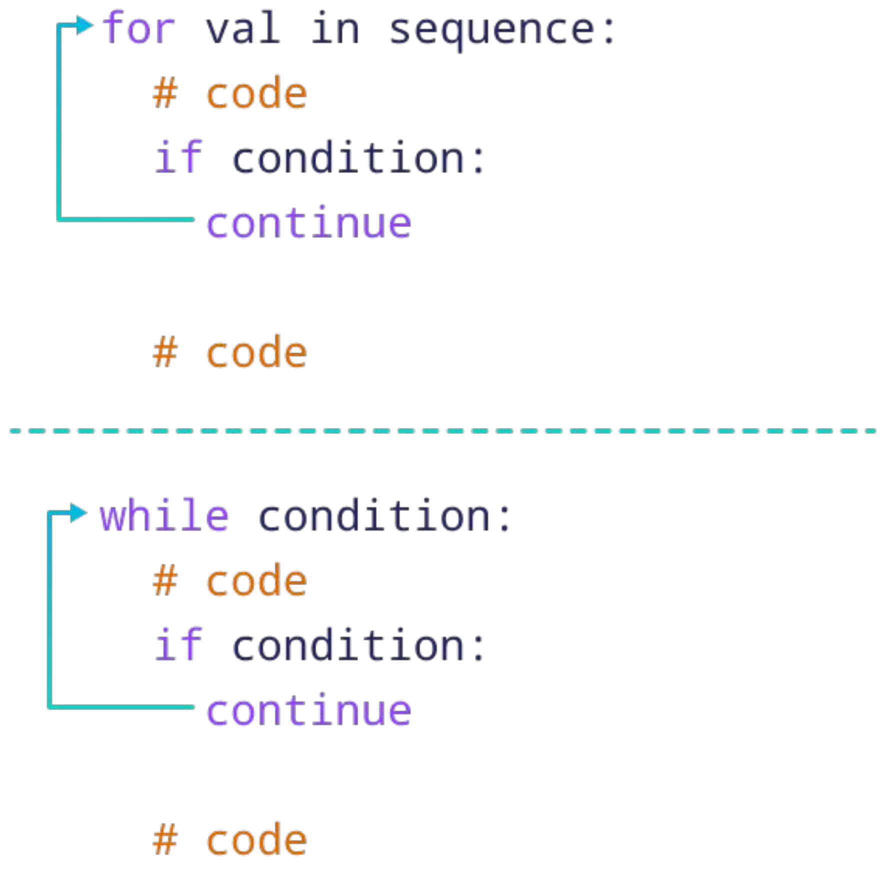
```
while condition:
  # code
  if condition:
    break

  # code
```



```
for i in range(5):
    if i == 3:
        break
    print(i)
```

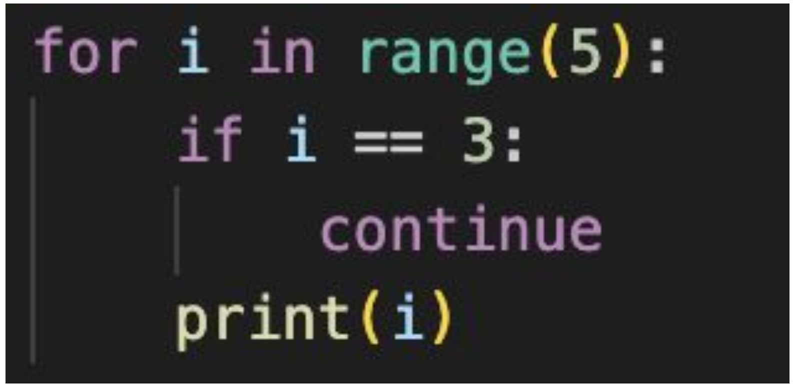
continue statement

- skips the current iteration and proceeds to the next one



```
for val in sequence:
  # code
  if condition:
    continue

  # code
```

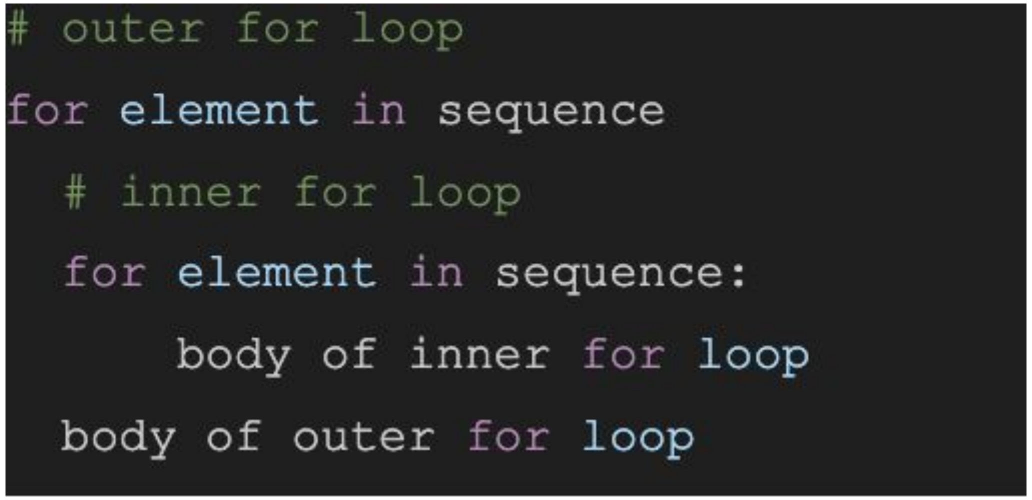
```
while condition:
  # code
  if condition:
    continue

  # code
```



```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Nested Loops

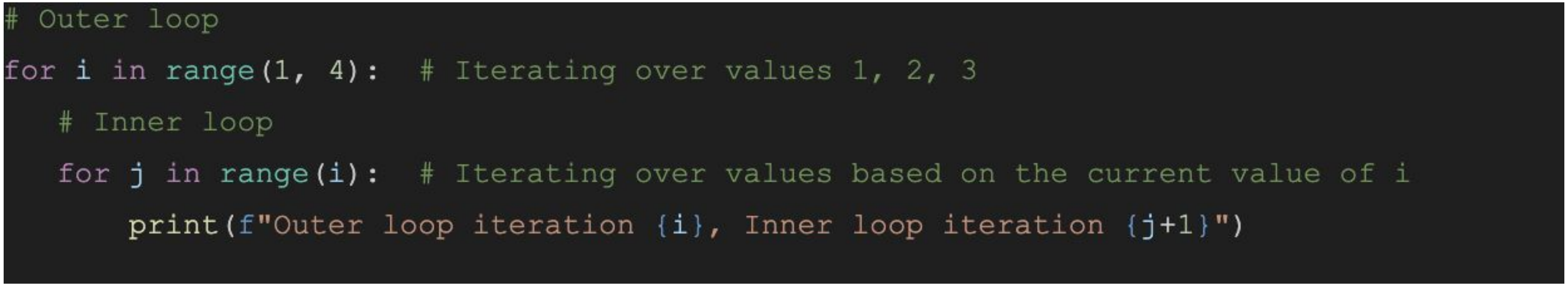
- Loop inside a loop
- In each iteration of the outer loop, the inner loop executes all its iteration.
- Syntax:



```
# outer for loop
for element in sequence
  # inner for loop
  for element in sequence:
      body of inner for loop
  body of outer for loop
```

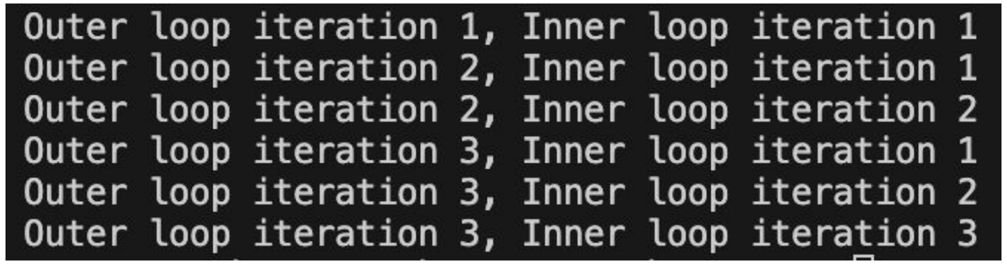
Nested Loops

Example:



```python
# Outer loop
for i in range(1, 4):  # Iterating over values 1, 2, 3
    # Inner loop
    for j in range(i):  # Iterating over values based on the current value of i
        print(f"Outer loop iteration {i}, Inner loop iteration {j+1}")
```

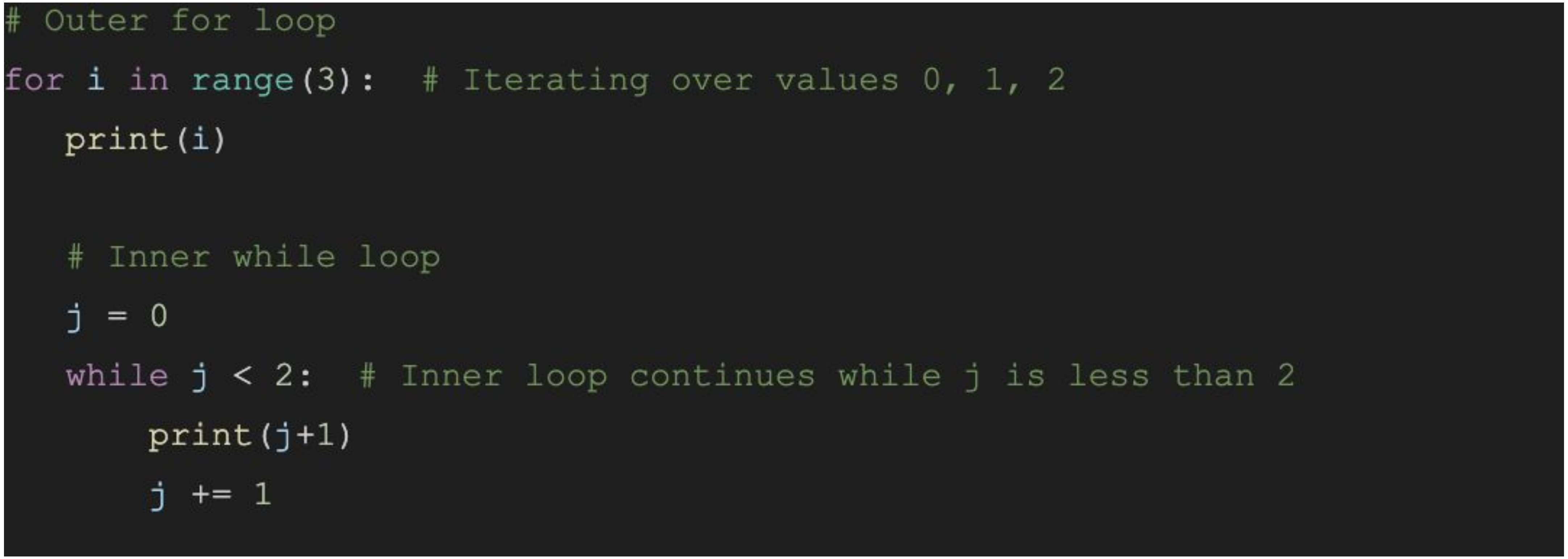
Output:



```
Outer loop iteration 1, Inner loop iteration 1
Outer loop iteration 2, Inner loop iteration 1
Outer loop iteration 2, Inner loop iteration 2
Outer loop iteration 3, Inner loop iteration 1
Outer loop iteration 3, Inner loop iteration 2
Outer loop iteration 3, Inner loop iteration 3
```

Nested Loops

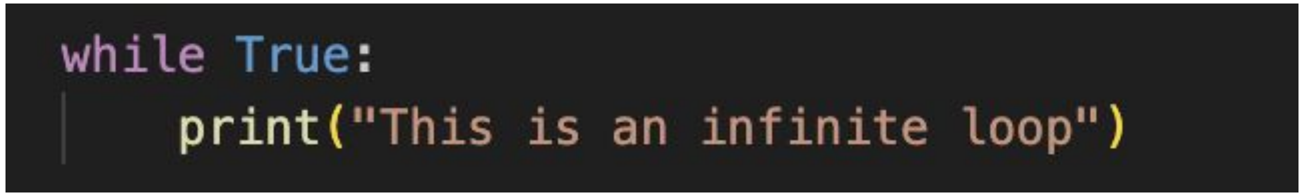
Example of while loop inside a for loop:



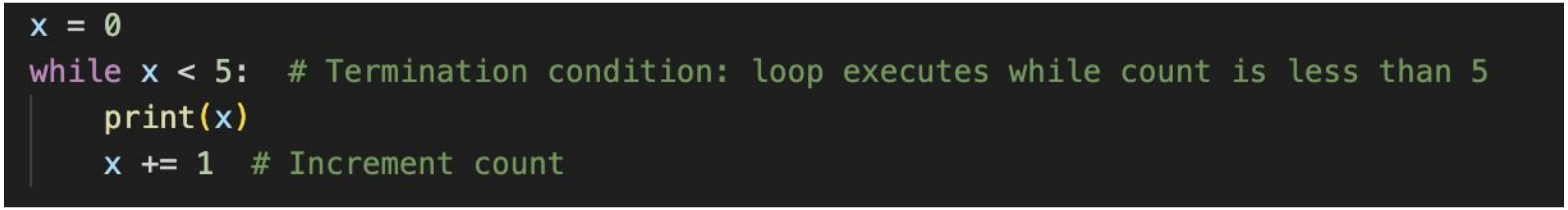
```python
# Outer for loop
for i in range(3):  # Iterating over values 0, 1, 2
    print(i)

    # Inner while loop
    j = 0
    while j < 2:  # Inner loop continues while j is less than 2
        print(j+1)
        j += 1
```

Infinite Loops

To execute indefinitely without reaching a condition to terminate.



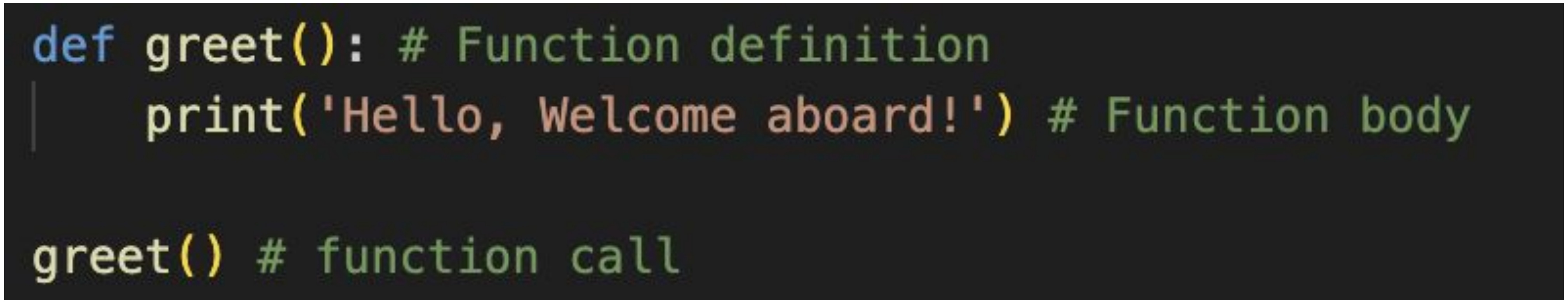
```python
while True:
    print("This is an infinite loop")
```



```python
x = 0
while x < 5:  # Termination condition: loop executes while count is less than 5
    print(x)
    x += 1  # Increment count
```

Functions

- a block of code which runs only when it is called and NOT when the function is defined.

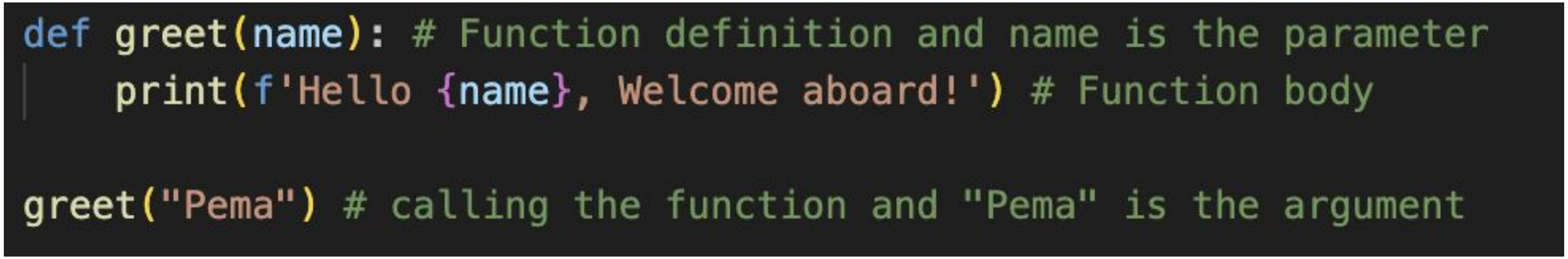


```
def greet(): # Function definition
    print('Hello, Welcome aboard!') # Function body

greet() # function call
```

Functions

- Parameter: Variable/input that is defined for the function.

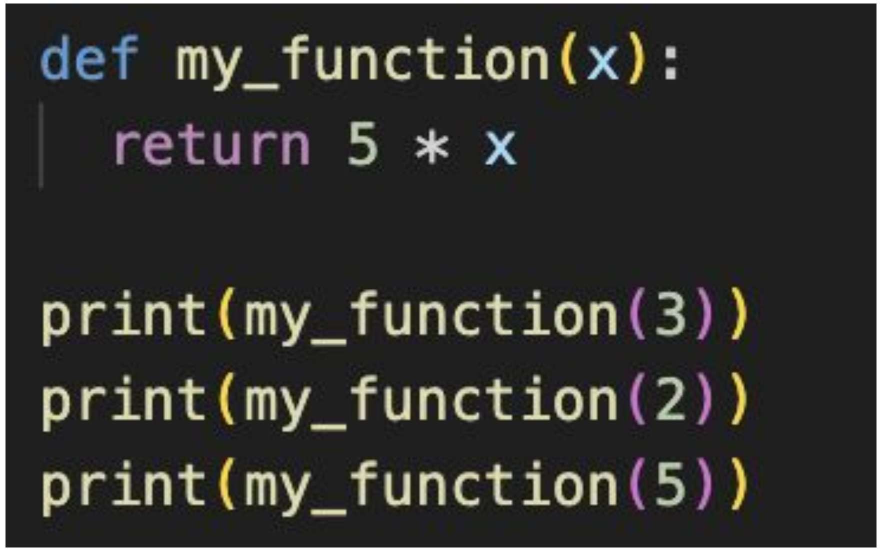
- Arguments: The value that is passed to the function during function call.



```python
def greet(name): # Function definition and name is the parameter
    print(f'Hello {name}, Welcome aboard!') # Function body

greet("Pema") # calling the function and "Pema" is the argument
```

Functions

Example of a function to return a value:



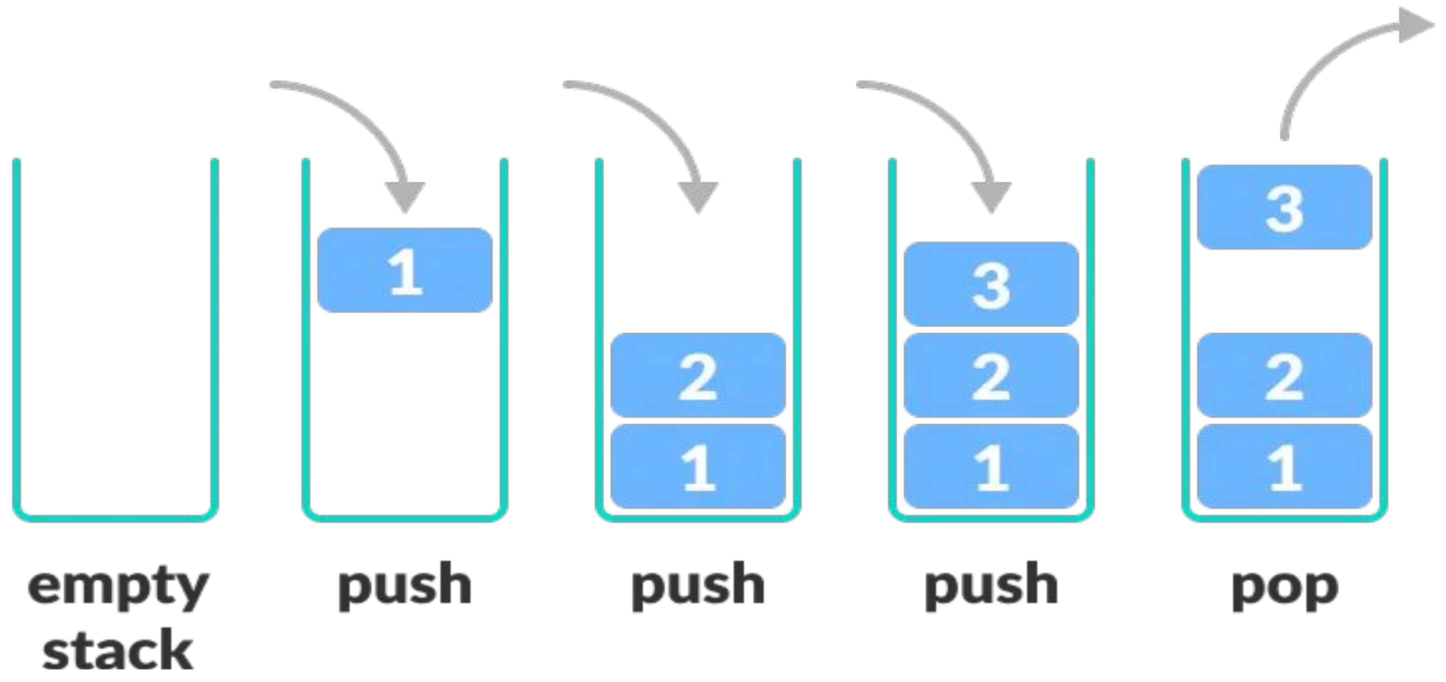
```
def my_function(x):
    return 5 * x

print(my_function(3))
print(my_function(2))
print(my_function(5))
```

Call Stack

- It maintains the record of the function.
- Remembers where to return the execution after each function call.
- When calling a function, a frame object is created on top of the call stack.
- When a function call returns, the frame object from the top of the stack is removed.

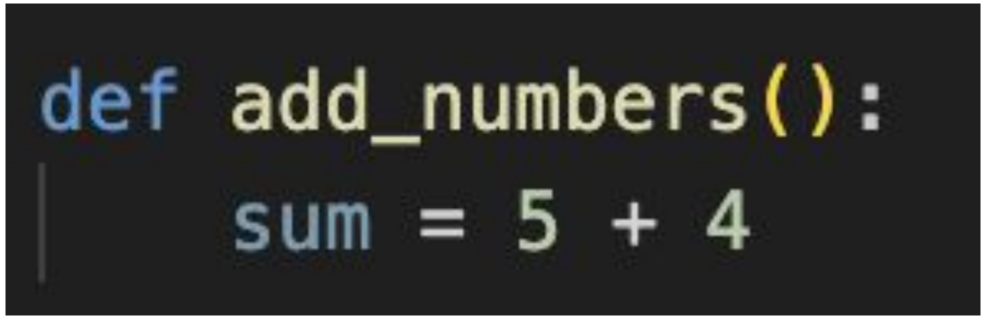
Call stack

The top of the stack is where the function execution is currently in.



Scope

Region of the program where a variable is visible and accessible

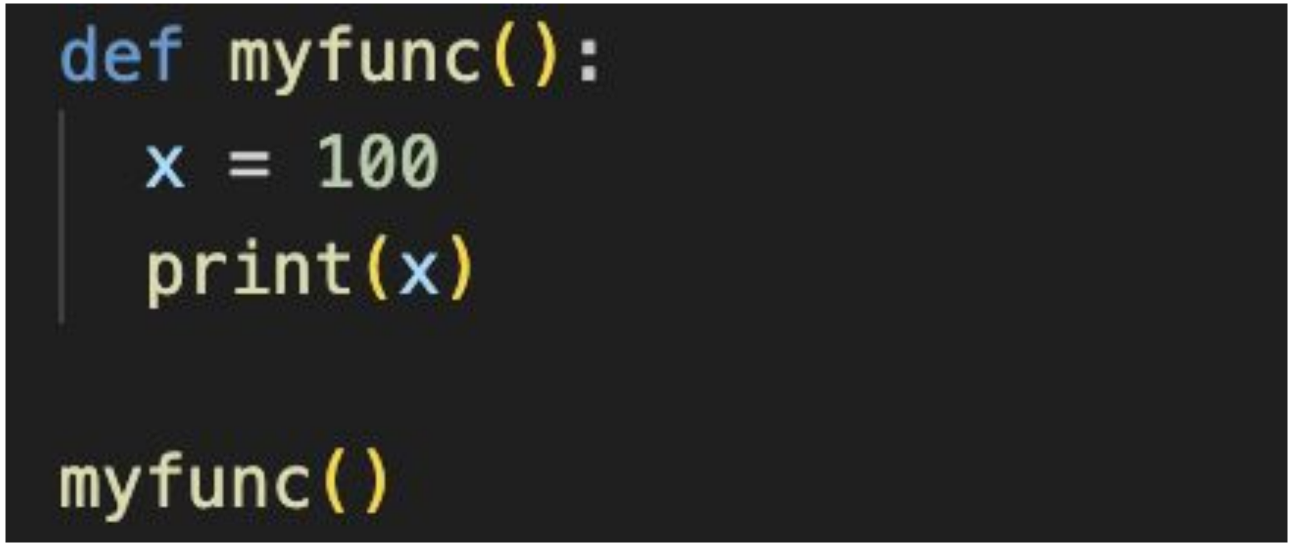


```python
def add_numbers():
    sum = 5 + 4
```

Types of scopes:
- Local Scope
- Global Scope
- Nonlocal Scope
- Built-in scope

1. Local Scope

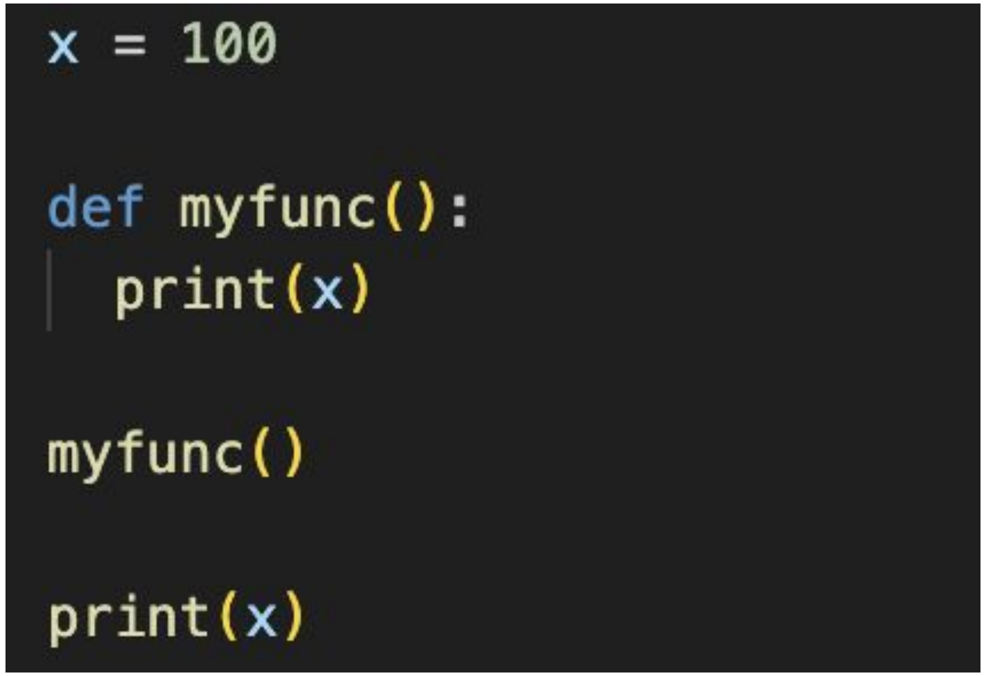
- Code block or body of any function.
- The type of variable is called local variable



```python
def myfunc():
  x = 100
  print(x)

myfunc()
```

2. Global Scope

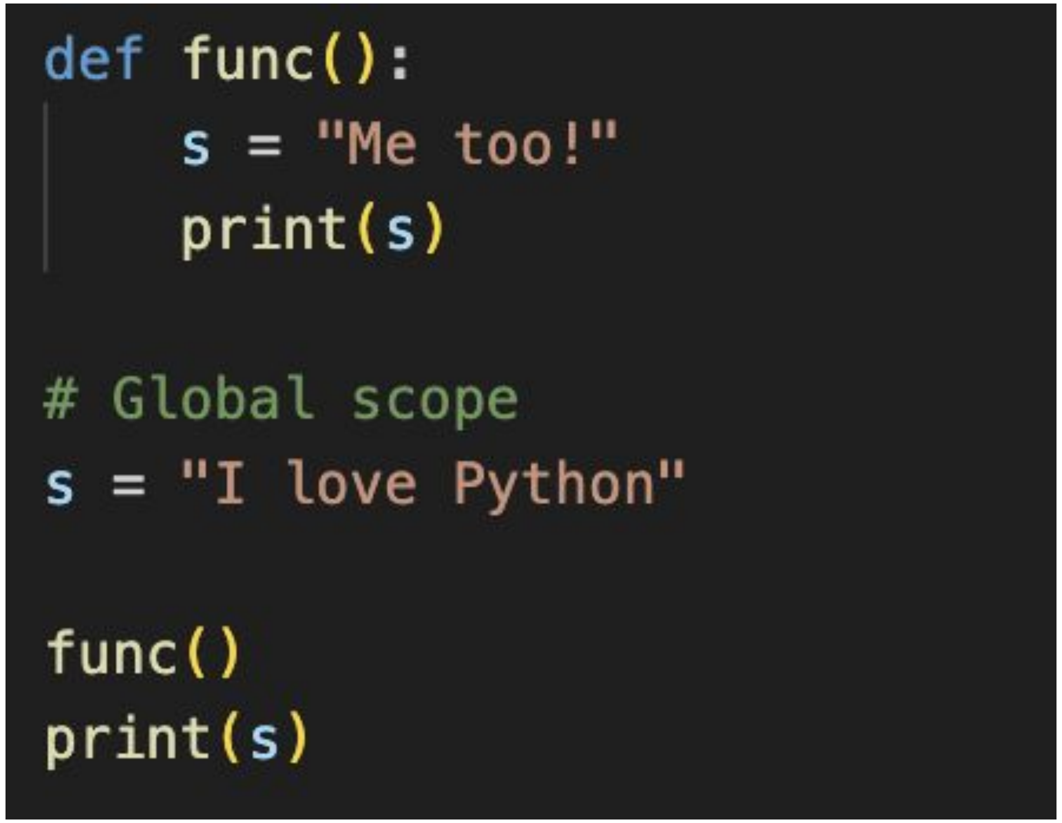
- A variable created in the main body of the code is called a global variable and belongs to the global scope.



```
x = 100

def myfunc():
  print(x)

myfunc()

print(x)
```

Cont…

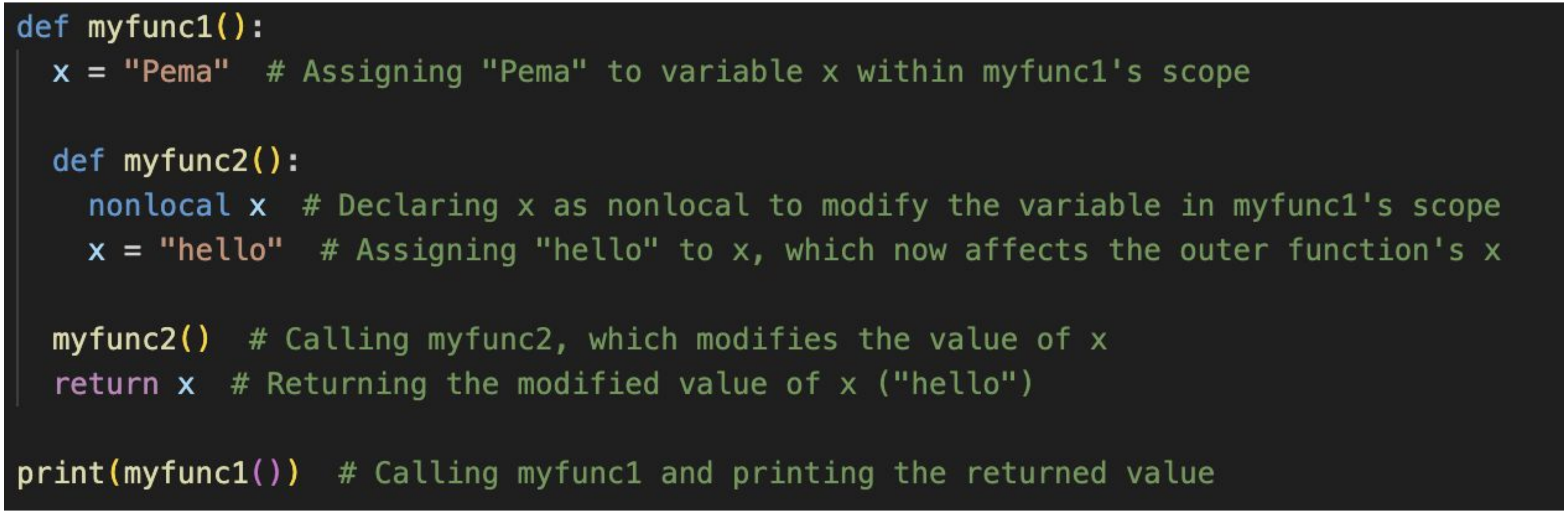
Exercise:



```python
def func():
    s = "Me too!"
    print(s)

# Global scope
s = "I love Python"

func()
print(s)
```

3. Nonlocal (Enclosed) Scope

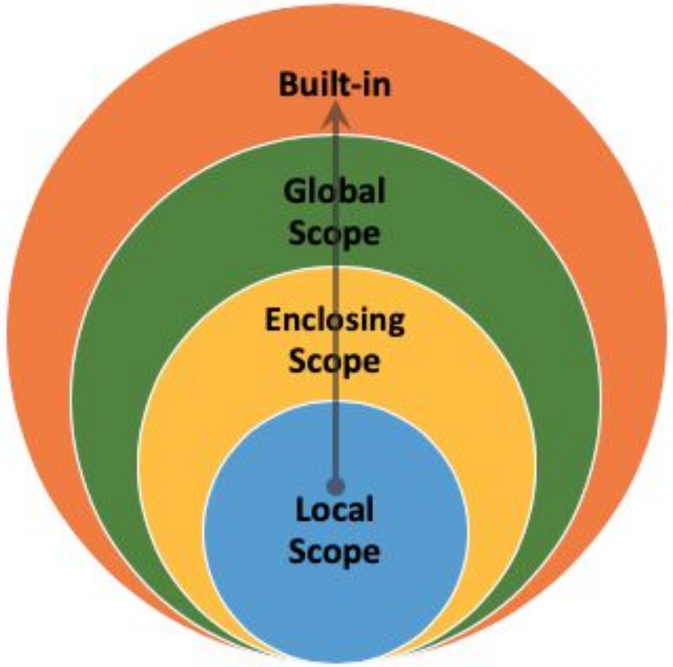
- It is a special scope that exists for nested functions
- ‘nonlocal’ keyword is used



```python
def myfunc1():
    x = "Pema"  # Assigning "Pema" to variable x within myfunc1's scope

    def myfunc2():
        nonlocal x  # Declaring x as nonlocal to modify the variable in myfunc1's scope
        x = "hello"  # Assigning "hello" to x, which now affects the outer function's x

    myfunc2()  # Calling myfunc2, which modifies the value of x
    return x  # Returning the modified value of x ("hello")

print(myfunc1())  # Calling myfunc1 and printing the returned value
```

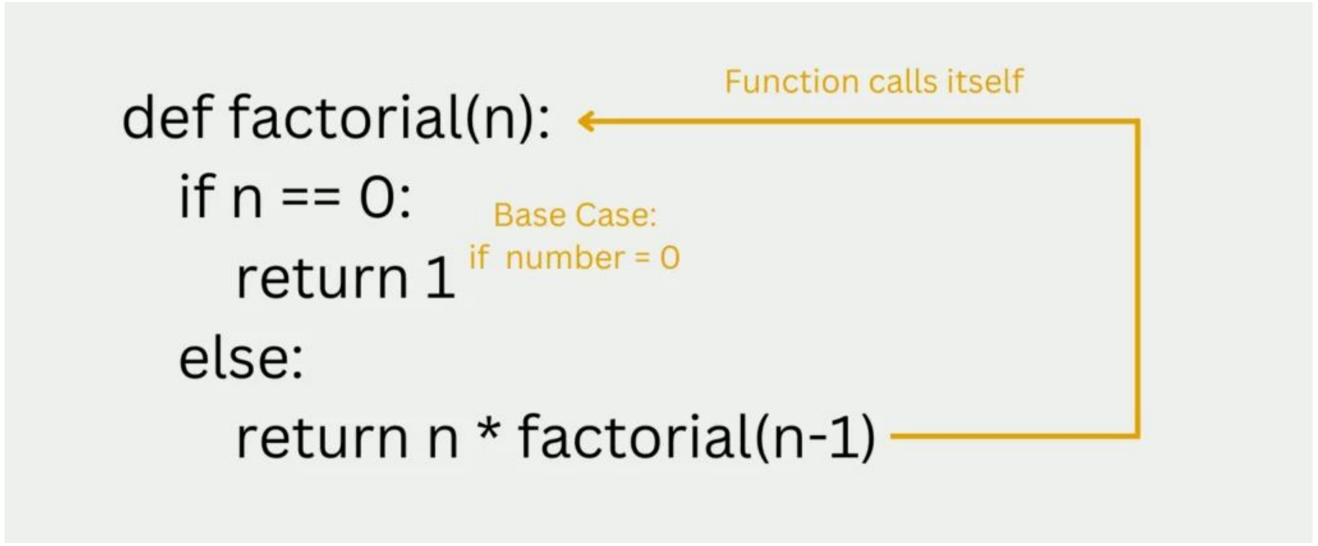
4. Built-in scope

- Scope that's created or loaded whenever we run a program.
- Predefined elements provided by Python.



Function Recursion

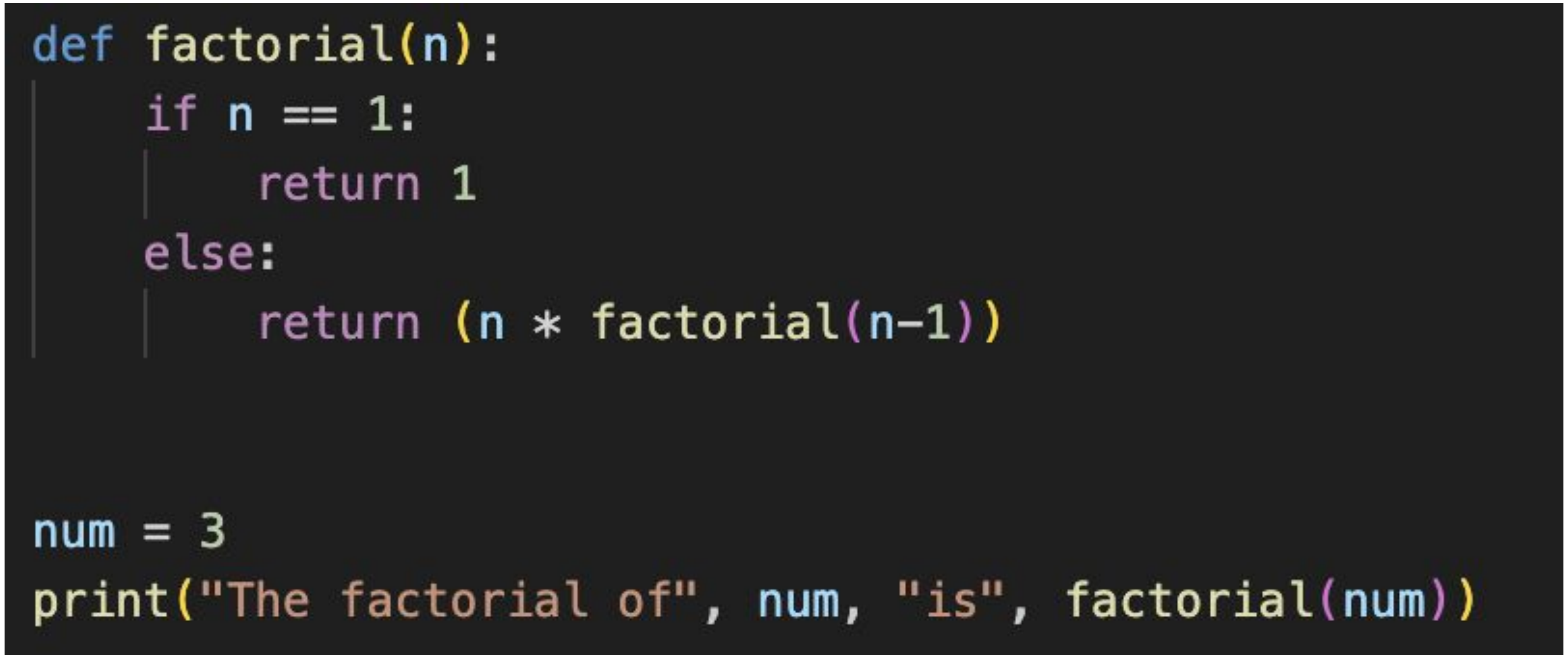
- Function calling itself
- It is defined in terms of itself via self-referential expressions.
- Two Parts:
  - Base Case: After a finite number of steps, it terminates its recursion.
  - Recursive case: Calls functions



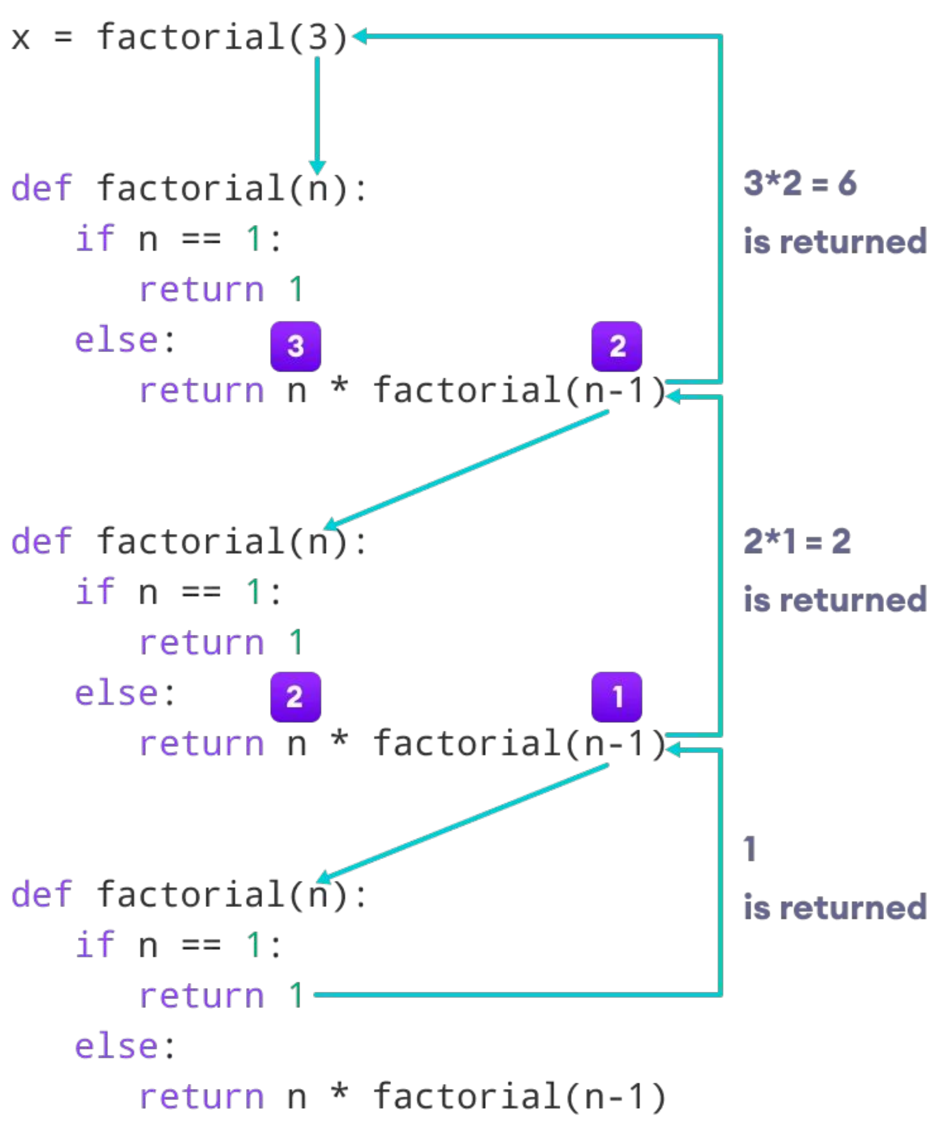
```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Recursion

Example: Recursice function to find the factorial of an interger



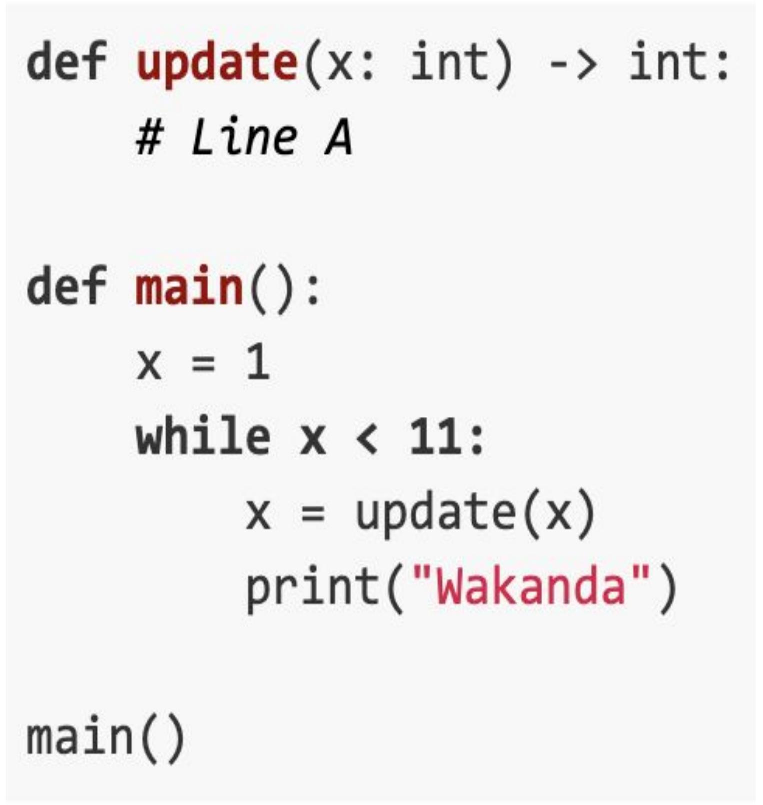
```python
def factorial(n):
    if n == 1:
        return 1
    else:
        return (n * factorial(n-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```



```
x = factorial(3)


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


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


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

Exercise

Functions. Consider the ‘for‘ loop below. Line A in the update function is left blank.



```
def update(x: int) -> int:
    # Line A

def main():
    x = 1
    while x < 11:
        x = update(x)
        print("Wakanda")

main()
```

Which of the following statement, if inserted into Line A, would cause the program to go into an infinite loop and print Wakanda forever?

(i) return x*x;

(ii) return x + 0.0001;

(iii) return x;

A. (i) only

B. (iii) only

C. (i) and (iii) only

D. (ii) and (iii) only

E. (i), (ii), and (iii)

Reference

Learn Python Programming. (n.d.). https://www.programiz.com/python-programming

Python Tutorial. (n.d.). https://www.w3schools.com/python

THANK YOU