
# COMPSCI 101
# Principles of Programming

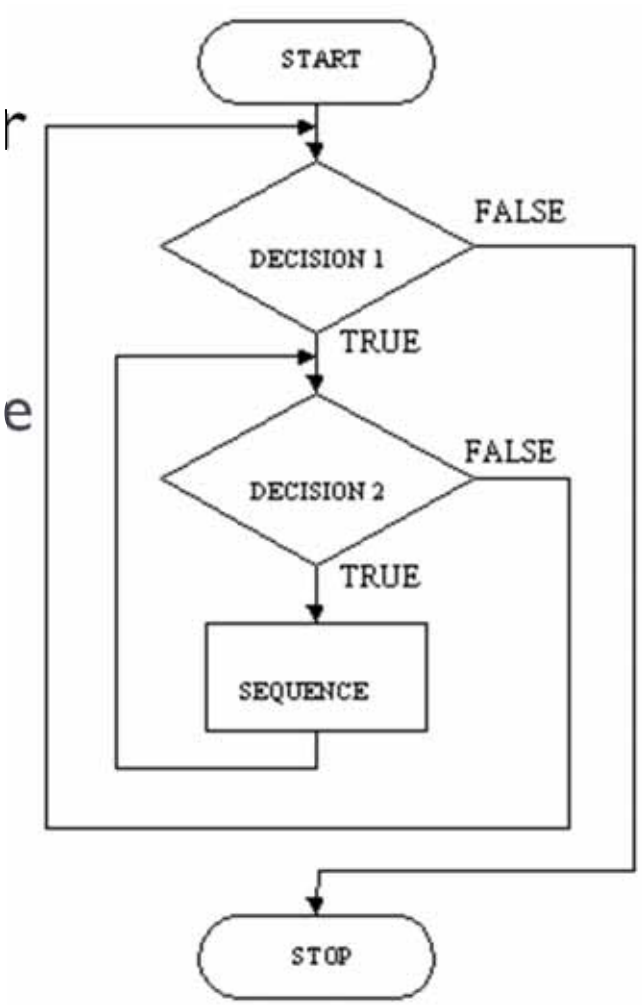
## Lecture 27 – Nested loops, passing mutable objects as parameters

# Learning outcomes

- At the end of this lecture, students should be able to:
  - understand that the body of a loop can contain any types of statements including **another loop**
  - show the output of code containing **nested loops**
  - code trace functions which have **mutable** objects as parameters

# Nested loops

- The body of a for … in loop can include any code structures (e.g. if, if … else, if … elif, assignment statements) including other for … in loops or while loops. These are called **nested loops**.
  - When nested, the inner loop iterates from the beginning to the end for each single iteration of the outer loop.
  - There is no limit in Python as to how many levels you can nest loops. It is usually not more than three levels.



# Example 1

Example01.py

DEMO

- In order to print 5 numbers in a single line, we can do:

```
def print_numbers(n):
    for num1 in range(n):
        print(num1, end=" ")
```

```
0 1 2 3 4
```

- In order to get 5 such lines, all we need to do is repeat the loop 5 times. We can do that with an additional outer for loop, which will repeatedly execute the inner for loop:
  - First Attempt :

```
def print_numbers(n):
    for num2 in range(n):
        for num1 in range(n):
            print(num1, end=" ")
```

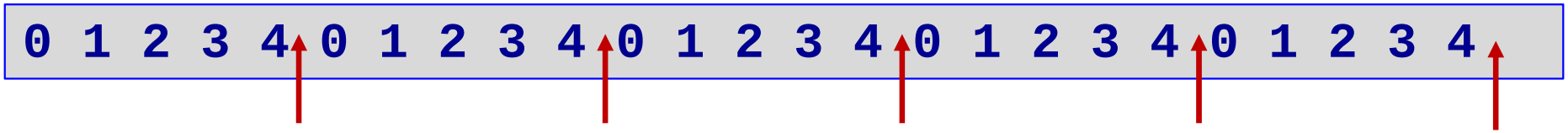
All the numbers in one line:

```
0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4 0 1 2 3 4
```

# Example 1 con't

```
def print_numbers(n):
    for num2 in range(n):
        for num1 in range(n):
            print(num1, end=" ")
```

- Example:
  - Second Attempt :
  - insert a new line after each sequence 0 1 2 3 4




  - The outer for loop contains two statements:
    - 1) inner for loop
    - 2) print(): move cursor to the next line

```
def print_numbers(n):
    for num2 in range(n):
        for num1 in range(n):
            print(num1, end=" ")
        print() #move cursor to next line
```

Nested Loops!

```
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
0 1 2 3 4
```

# Example 2

Example01.py

DEMO

- For example:

```
for num1 in range(5):
    print("A", end=" ")
    for num2 in range(3):
        print("B", end=" ")
    print()
print("C", end=" ")
```

  - The outer for loop contains three statements:
    - 1) print A
    - 2) inner for loop
    - 3) print(): move cursor to the next line
- Questions:
  - how many times is "A" printed?
  - how many times is "B" printed?
  - how many times is "C" printed?

3 columns of "B"

```
A B B B
A B B B
A B B B
A B B B
A B B B
C
```

5 rows

# Exercise 1

Exercise01.py

- How many times is the word "hello" printed? What is the output of the following code?

```
def main():
    for i in range(3):
        for j in range(4):
            print("hello")

main()
```

- How many times is the word "hello" printed? What is the output of the following code?

```
def main():
    for i in range(3):
        for j in range(4):
            print("hello",end=" ")

main()
```

# Example 3

Example01.py

DEMO

```
def main():
    number = 0
    for i in range(3):
        number += 1
        for j in range(4):
            print(number, end = " ")
    print()

main()
```

Execute the increment statement three times

| i | j | number | output |
|---|---|---|---|
| | | 0 | |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 1 |
| 0 | 2 | 1 | 1 1 1 |
| 0 | 3 | 1 | 1 1 1 1 |
| 1 | 0 | 2 | 1 1 1 1 2 |
| 1 | 1 | 2 | 1 1 1 1 2 2 |
| 1 | 2 | 2 | 1 1 1 1 2 2 2 |
| 1 | 3 | 2 | 1 1 1 1 2 2 2 2 |
| 2 | 0 | 3 | 1 1 1 1 2 2 2 2 3 |
| 2 | 1 | 3 | ... |
| 2 | 2 | 3 | ... |
| 2 | 3 | 3 | ... |

- The outer for loop contains two statements:
  - 1) statement which increments number by 1
  - 2) inner for loop
- The inner for loop contains one statement:
  - statement which prints the number

1 1 1 1 2 2 2 2 3 3 3 3

# Example 4

Example01.py

DEMO

```
def main():
    number = 0
    for i in range(3):
        print(number, end = " ")
        for j in range(4):
            number += 1
    print()
    print(number)

main()
```

Move the increment statement to the inner body!

| i | output | j | num |
|---|---|---|---|
| 0 | 0 | | 0 |
| 0 | | 0 | 1 |
| 0 | | 1 | 2 |
| 0 | | 2 | 3 |
| 0 | | 3 | 4 |
| 1 | 0 4 | 0 | 5 |
| 1 | | 1 | 6 |
| 1 | | 2 | 7 |
| 1 | | 3 | 8 |
| 2 | 0 4 8 | 0 | 9 |
| 2 | | 1 | 10 |
| 2 | | 2 | 11 |
| 2 | | 3 | 12 |
| | 0 4 8<br>12 | | |

- The outer for loop contains two statements:
  - 1) statement which prints the number
  - 2) inner for loop
- The inner for loop contains one statement:
  - statement which increments number by 1

```
0 4 8
12
```

# Exercise 2

Exercise02.py

(2,0) …

| i | j |
|---|---|
| 2 | 0 |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

- What is the output after executing the following code?

```
def main():
    for i in range(2, 5):
        for j in range(3):
            print("(", i, ",", j, ")", sep="", end=" ")
        print()

main()
```

- The outer loop contains _____ statements (executes ___ times)
  - Inner for loop
  - print()
- The inner loop contains _____ statement (executes ____ times)
  - print(…)

# Nested Loop & Lists

Example05.py

DEMO

```
def main():
    list1 = [5, 4, 3, 2]
    list2 = [3, 4]
    list3 = []
    for num1 in list1:
        for num2 in list2:
            list3.append(num1 + num2)
    print(list3)

main()
```

Append a new element onto list3

| list1 | list2 | list3 |
|---|---|---|
| 5 | 3 | 8 |
| 5 | 4 | 8,9 |
| 4 | 3 | 8,9,7 |
| 4 | 4 | 8,9,7,8 |
| 3 | 3 | 8,9,7,8,6 |
| 3 | 4 | 8,9,7,8,6,7 |
| 2 | 3 | 8,9,7,8,6,7,5 |
| 2 | 4 | 8,9,7,8,6,7,5,6 |

[8, 9, 7, 8, 6, 7, 5, 6]

- The outer loop contains _____ statement (executes ____ times)
  - Inner for loop
- The inner loop contains _____ statement (executes _____ times)
  - Append a new element onto list3

# Example 6: Counting Vowel Letters

Example06.py

DEMO

- Task:
  - Complete the **`get_list_of_vowel_counts()`** function which returns a list of the **number of vowels** in each word in the parameter list.

```
def main():
    name_list = ["Mirabelle", "John",   "Kelsey", …]
    vowel_counts = get_list_of_vowel_counts(name_list)
    print(vowel_counts)

main()
```

  - Examples:
    - Mirabelle : 4 vowels
    - John: 1 vowel
    - etc

[4, 1, 2, 3, 4, 3, 4, 3, 1, 2, 3]

# Working on the inner Loop

- Your inner loop should:
  - count the number of vowels in **ONE** word only
  - Examples:
    - "Mirabelle" : gives 4
    - "John" : gives 1
    - "Kelsey" : gives 2

For each letter in the word

- If it is in the list of vowels
- Increment the count

# Working on the outer loop

- Your outer loop should:
  - append the number of vowels in each word in the parameter list to the output list
    - In the example, the output list (vowel_counts) should contain the following elements step by step:
    - [4]
    - [4, 1]
    - [4,1,2]
    - ...

For each word in the parameter list

- Set count = 0
- Calculate the number of vowels in the word
- Append the number to the output list

# The get_list_of_vowel_counts() function

- function returns a list of the number of vowels in each word in the parameter list.

```
def get_list_of_vowel_count(word_list):
  vowels = "aeiouAEIOU"
  vowel_counts = []
  for word _____   _____________:
    count = _____
    for letter in ______:
      if letter in "aeiouAEIOU":
        count += 1
    vowel_counts += [______]
return vowel_counts
```

# Exercise 3

Exercise03.py

- What is the output of the following code?

```
def main():
    for first in range(2, 5):
        for second in range(1, first):
            print("(", first, ",", second, ")", sep="", end=" ")
        print()

main()
```

# Exercise 4

Exercise04.py

- What is the output of the following code?

```
def main():
    total = 0
    for first in range(1, 5):
        total += first

        for second in range(1, first):
            total += second

    print("Grand total:", total)

main()
```

# Example 7

Example07.py

DEMO

- prints lines of dots. The number of dots per line is given the value in the dot_list,
  - e.g., if the first value in dot_list is 9 then the first line printed has nine dots, etc.

```
def print_dots(dot_list):
  for num1 in dot_list:
    for num in range(num1):
      print(".", end = "")
    print()

def main():
  dot_list = [10, 3, 6, 9, 21, 11]
  print_dots(dot_list)

main()
```

```
..........
...
......
.........
.....................
...........
```

# Exercise 5 (harder)

Exercise05.py

- Complete the print_dot_columns() function which prints line of dots as shown below
  - Hint: get the max of the list elements

=number of rows

```
def print_dot_columns (dot_list):


def main():
  dot_list = [10, 3, 6, 9, 2, 7]
  print_dot_columns(dot_list)

  dot_list = [5, 1, 2, 4]
  print_dot_columns(dot_list)
main()
```

```
10
9
8
7
6
5
4
3
2
1
```

```
.
.       .
.       .
.       .     .
.     . .     .
.     . .     .
.     . .     .
. . . .       .
. . . . . .
. . . . . .
```

```
.
.       .
.       .
.     . .
. . . . .
```

max = 5

At row i, col j, if the $j^{th}$ num in the list > row i, print '.'

e.g. row 3, first col, value is 5, bigger than 3, print .

e.g. row 3, $2^{nd}$ col, value is 1, less than 3, print a space

# String – Immutable objects

- Every UNIQUE string you create will have its own address space in memory.
- Strings are "immutable", i.e., the characters in a string object cannot be changed. Whenever a string is changed in some way, a new string object is created.

```
>>> a = 'foo'
>>> b = 'foo'
>>> id(a)
46065568
>>> id(b)
46065568

>>> a is b
True
>>> a == b
True
>>>
```

Same memory location

```
word1 = "hello"
word2 = word1
print("1.", word1, word2)
print("2.", word1 is word2)

word2 = word1.upper()
print("3.", word1, word2)
print("4.", word1 is word2)
```

1. hello hello
2. True
3. hello HELLO
4. False

# Lists are Mutable

Example08.py

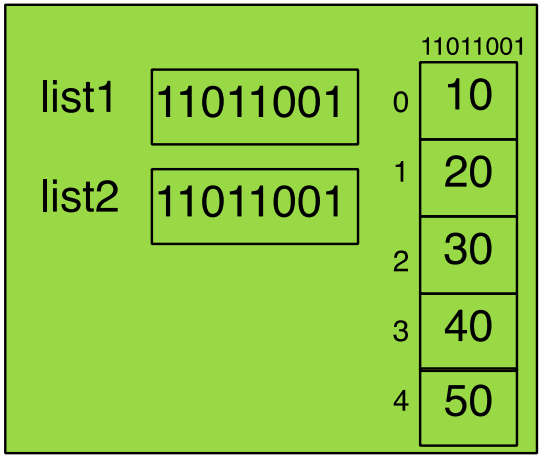
DEMO

- Lists are "mutable", i.e., the contents of a list object can be changed.

```
list1 = [10, 20, 30, 40, 50]
list2 = list1
print("1.", list1 is list2)

list1[3] = 99
list2.append(1)

print("2.", list1)
print("3.", list2)
print("4.", list1 is list2)
```




1. True
2. [10, 20, 30, 99, 50, 1]
3. [10, 20, 30, 99, 50, 1]
4. True

# Passing parameters to functions

- When parameters are passed to functions:
  - the parameter passed in is actually a reference to an object
  - some data types are mutable, but others aren't
- Mutable objects:
  - If you pass a mutable object into a function, the function gets a reference to that same object and you can mutate it,
  - but if you rebind the reference in the function, the outer scope will know nothing about it, and after you're done, the outer reference will still point at the original object.
- Immutable Objects:
  - If you pass an immutable object to a function, you still can't rebind the outer reference, and you can't even mutate the object.

Case 1

Case 2

Case 4

Example09.py

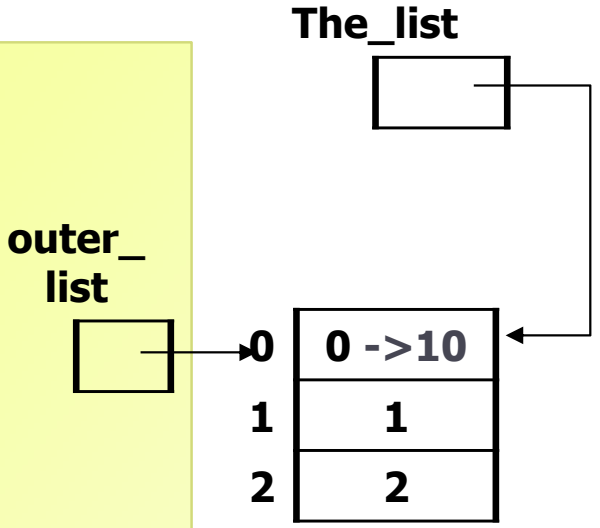
DEMO

# Passing Mutable Objects as parameters

- Case 1: Modify the list that was passed to a function:

```
def try_to_change_list_contents(the_list):
    print ('got', the_list)
    the_list[0] = 10
    print ('changed to', the_list)

outer_list = [0,1,2]
print ('before, outer_list =', outer_list)
try_to_change_list_contents(outer_list)
print ('after, outer_list =', outer_list)
```




```
before, outer_list = [0, 1, 2]
got [0, 1, 2]
changed to [10, 1, 2]
after, outer_list = [10, 1, 2]
```

both the outer_list and the argument the_list hold references to the same object.

- Since the parameter passed in is a reference to outer_list, not a copy of it, we can modify it and have the changes reflected in the outer scope.

Example10.py

DEMO

# Passing Mutable Objects as parameters

- Case 2: Change the reference that was passed in as a parameter

```
def try_to_change_list_reference(the_list):
    print ('got', the_list, 'at', id(the_list))
    the_list = [10,0,0]
    print ('set to', the_list, 'at', id(the_list))

outer_list = [0,1,2]
print ('before,outer_list =',outer_list,'at',id(outer_list))
try_to_change_list_reference(outer_list)
print ('after, outer_list =', outer_list)
```

```
before, outer_list = [0, 1, 2] at 37901192
got [0, 1, 2] at 37901192
set to [10, 0, 0] at 39104648
after, outer_list = [0, 1, 2]
```

the_list points to a new list, but there was no way to change where outer_list pointed.

- Since the reference of the parameter was passed into the function by value, assigning a **new list** to it had no effect that the code outside the function could see.

Example11.py

DEMO

# Immutable Objects as parameters

- Case 3: Strings are immutable, so there's nothing we can do to change the contents of the string.
- Case 4: Change the reference that was passed in as a parameter

```
def try_to_change_string_reference(the_string):
    print ('got', the_string,'at', id(the_string))
    the_string = 'ten'
    print ('set to', the_string,'at', id(the_string))

outer_string = "ZERO"
print ('before, outer_string =', outer_strin
try_to_change_string_reference(outer_string)
print ('after, outer_string =', outer_string
```

```
before, outer string = ZERO
got ZERO at 40987928
set to ten at 40986024
after, outer_string = ZERO
```

- Since the_string parameter was passed by value, assigning a new string to it had **no effect** that the code outside the function could see.
- the_string points to a new string, but there was no way to change where outer_string pointed.

Example12.py

DEMO

# Immutable Objects as parameters

- How do we get around this? How do we get the modified value?
  - Solution: You could **return** the new value. This doesn't change the way things are passed in, but does let you get the information you want back out.

```
def return_a_whole_new_string(the_string):
    print ('got', the_string,'at', id(the_string))
    the_string = 'ten'
    print ('set to', the_string,'at', id(the_string))
    return the_string

outer_string = "ZERO"
print ('before, outer_string =', outer_string)
outer_string = return_a_whole_new_string(outer_string))

print ('after, outer_string =', outer_string)
```

```
before, outer_string = ZERO
got ZERO at 40463640
set to ten at 40461736
after, outer_string = ten
```

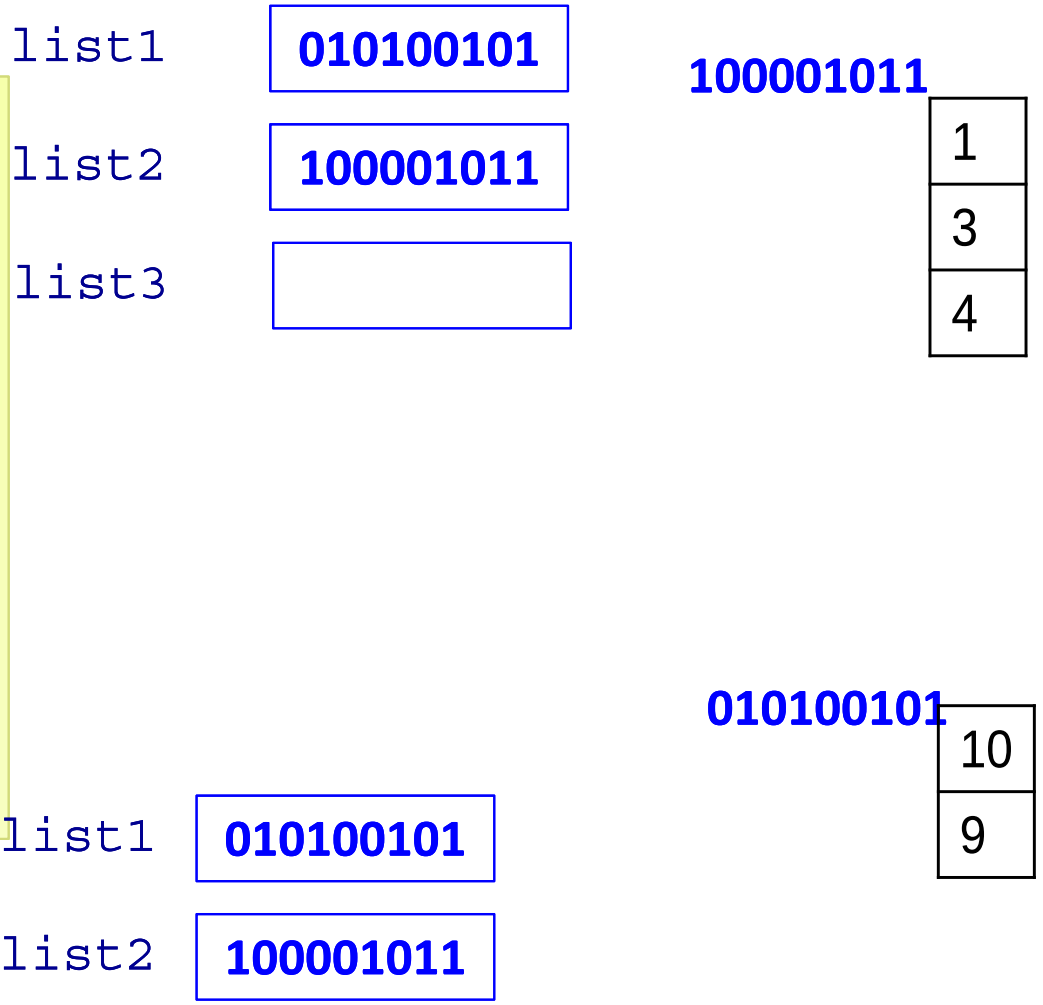
# Exercise 6

Exercise06.py

- What is the output after executing the following code?

```
def function_16(list1, list2):
    #print("  got ", list2)
    list3 = list2
    list3.append(list1[1])
    list2.append(list1[0])
    #print("  set to", list2)

a_list1 = [10, 9]
a_list2 = [1, 3, 4]
#print ('before', a_list2)
function_16(a_list1, a_list2)
print(a_list2)
print()
```



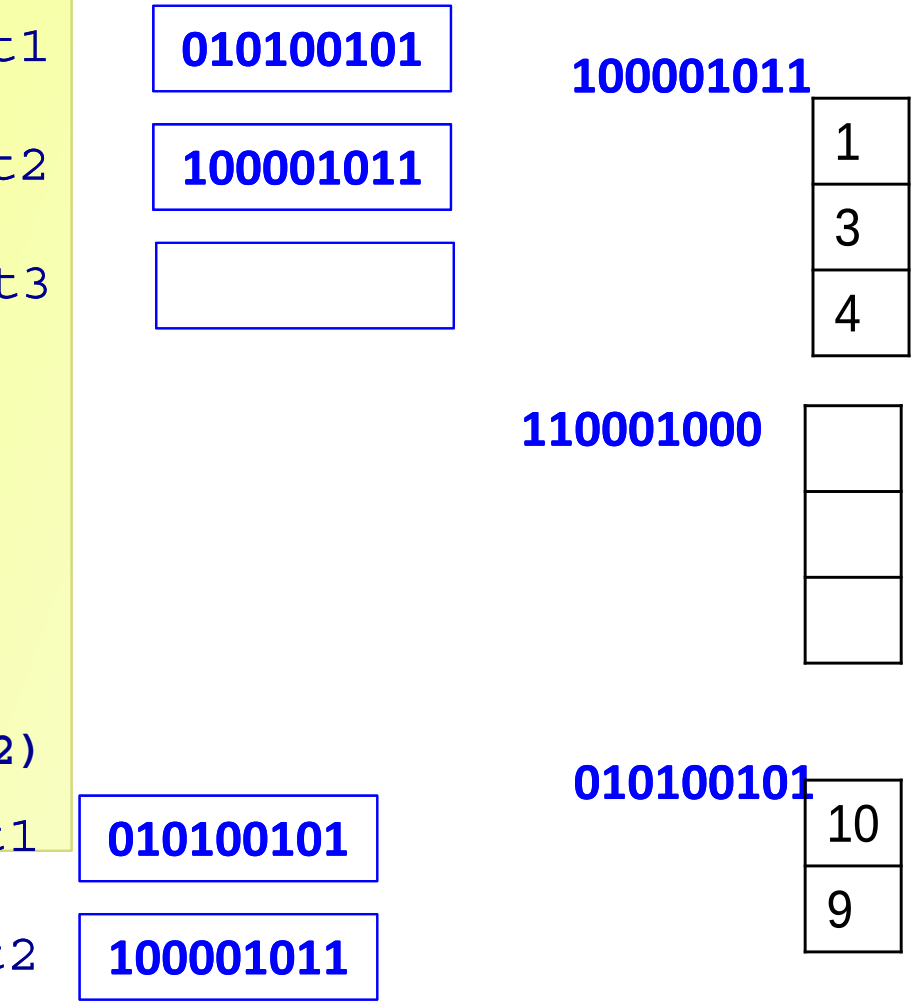
# Exercise 7

Exercise07.py

- What is the output after executing the following code?

```
def function_17(list1, list2):
    #print("  got ", list2)
    list3 = []
    list3.append(list1[1])
    list3.append(list1[0])
    list2 = list3
    list2.append(list3[0])
    #print("  list2:", list2)
    return list3

a_list1 = [10, 9]
a_list2 = [1, 3, 4]
#print ('before', a_list2)
a_list1 = function_17(a_list1, a_list2)
print(a_list1, a_list2)
```



# Exercise 8

Exercise08.py

- What is the output after executing the following code?

```
def function_18(list1, list2):
    list3 = list2
    for i in range(len(list1)):
        list3.append(list1[i])
        list2.append(list1[i])

    #print("  list3:", list3)

a_list1 = [4, 3]
a_list2 = [1, 3, 4]
function_18(a_list1, a_list2)

print(a_list1, a_list2)
```

# Exercise 9

Exercise09.py

- What is the output after executing the following code?

```
def function_19(list1, list2):
    list3 = []
    list3.append(list1[1])
    list3.append(list1[0])
    list2.append(list3[0])
    list2.append(list3[1])
    return list3

a_list1 = [4, 3]
a_list2 = [1, 3, 4]
a_list2 = function_19(a_list1, a_list2)

print(a_list1, a_list2)
```

# Summary

- The body of loops can contain any kind of statements including other loops.
- Passing parameters which are mutable objects to functions means that the function code may change the object's data.