

Beginner Programming Fundamentals With Python

Module 4

Repetition Structures

Ali Samanipour

Jan 2022

Module Roadmap

1

Introduction to Repetition Structures

2

The while Loop: A Condition-Controlled Loop

3

The for Loop: A Count-Controlled Loop

4

Sentinels

5

Input Validation Loops

6

Nested Loops

Introduction to Repetition Structures

Programmers commonly have to write code that **performs the same task over and over.**

A **repetition structure** causes a statement or **set of statements to execute repeatedly.**

Introduction to Repetition Structures (An Example)

suppose you have been asked to write a program that calculates a 10 percent sales commission for several salespeople. Although it would not be a good design, one approach would be to write the code to calculate one salesperson's commission, and then repeat that code for each salesperson

Introduction to Repetition Structures (An Example)

```
1  # Get a salesperson's sales and commission rate.
2  sales = float(input('Enter the amount of sales: '))
3  comm_rate = float(input('Enter the commission rate: '))
4  # Calculate the commission.
5  commission = sales * comm_rate
6  # Display the commission.
7  print('The commission is $', format(commission, ',.2f'), sep='')
8
9  # Get another salesperson's sales and commission rate.
10 sales = float(input('Enter the amount of sales: '))
11 comm_rate = float(input('Enter the commission rate: '))
12 # Calculate the commission.
13 commission = sales * comm_rate
14 # Display the commission.
15 print('The commission is $', format(commission, ',.2f'), sep='')
16
17 # Get another salesperson's sales and commission rate.
18 sales = float(input('Enter the amount of sales: '))
19 comm_rate = float(input('Enter the commission rate: '))
20 # Calculate the commission.
21 commission = sales * comm_rate
22 # Display the commission.
23 print('The commission is $', format(commission, ',.2f'), sep='')
```

Introduction to Repetition Structures ...

Instead of writing the same sequence of statements over and over, a better way to repeatedly perform an operation is to write the code for the operation once, then place that code in a structure that makes the computer repeat it as many times as necessary. This can be done with ***a repetition structure***, which is more commonly ***known as a loop***.

Condition-Controlled and Count-Controlled Loops

for

A **count-controlled** loop repeats a specific number of times

while

A **condition-controlled** loop uses a true/false condition to control the number of times that it repeats

Module Roadmap

1

Introduction to Repetition Structures

2

The while Loop: A Condition-Controlled Loop

3

The for Loop: A Count-Controlled Loop

4

Sentinels

5

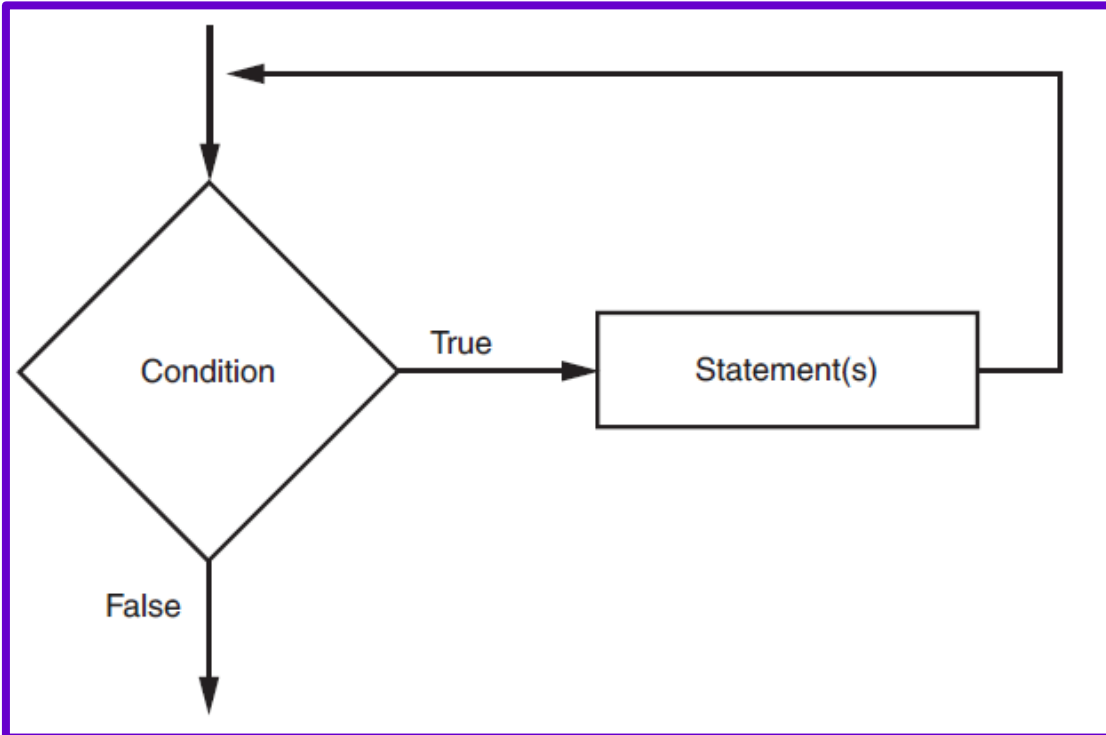
Input Validation Loops

6

Nested Loops

The while Loop: A Condition-Controlled Loop

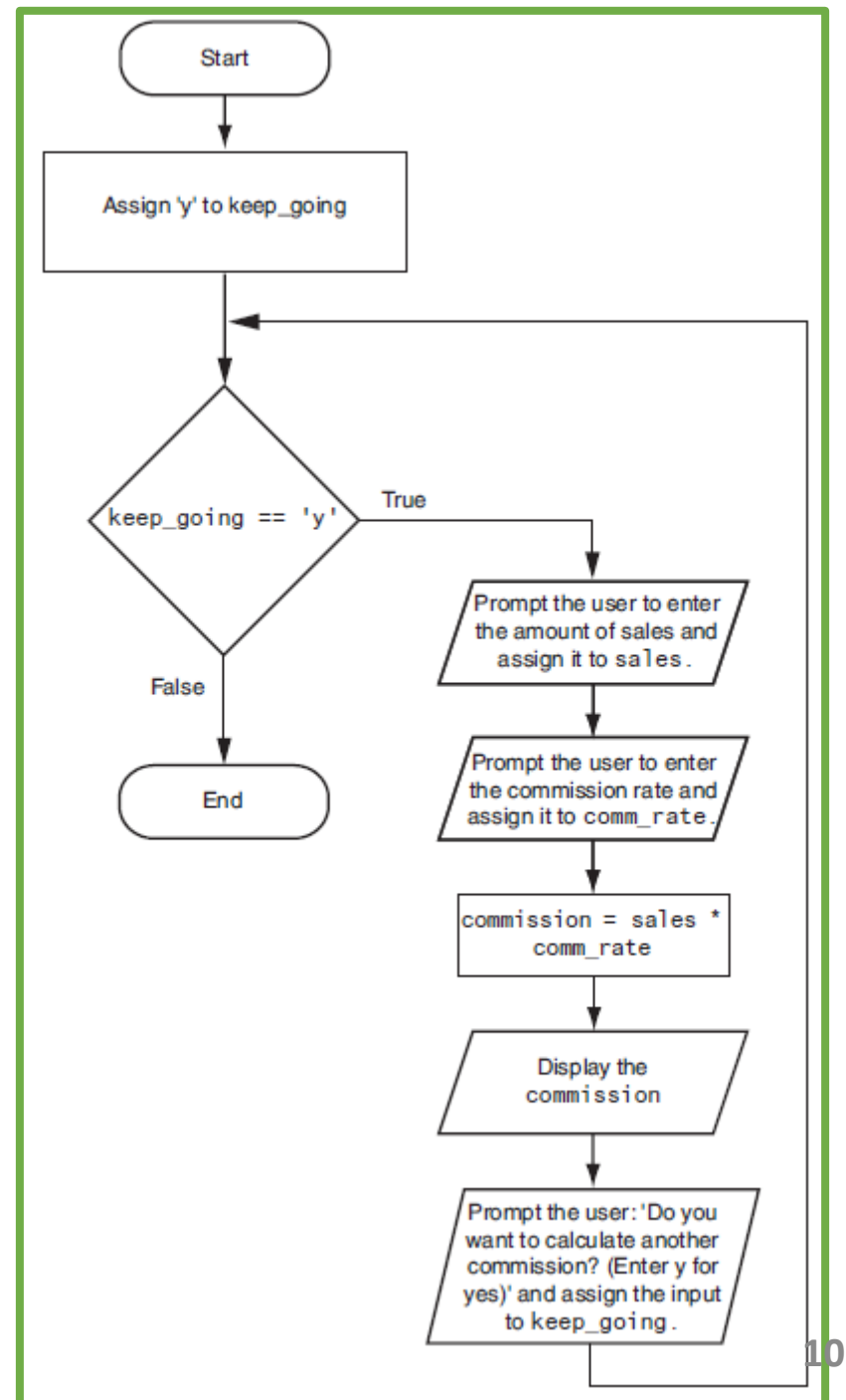
A condition-controlled loop causes a statement or set of statements to repeat as long as a condition is true



```
while condition:  
    statement  
    statement  
etc.
```

Example : Calculating Commission (Logic)

suppose you have been asked to write a program that calculates a 10 percent sales commission for several salespeople. Although it would not be a good design, one approach would be to write the code to calculate one salesperson's commission, and then repeat that code for each salesperson



Example : Calculating Commission (Code)

```
1 # This program calculates sales commissions.
2 # Create a variable to control the loop.
3 keep_going = 'y'
4 # Calculate a series of commissions.
5 while keep_going == 'y':
6     # Get a salesperson's sales and commission rate.
7     sales = float(input('Enter the amount of sales: '))
8     comm_rate = float(input('Enter the commission rate: '))
9     # Calculate the commission.
10    commission = sales * comm_rate
11    # Display the commission.
12    print(f'The commission is ${commission:,.2f}.')
13    # See if the user wants to do another one.
14    keep_going = input('Do you want to calculate another commission (Enter y for yes): ')
```

Infinite Loops

An *infinite* loop continues to repeat until the program is interrupted



```
1  # This program demonstrates an infinite loop.
2  # Create a variable to control the loop.
3  keep_going = 'y'
4  # Warning! Infinite loop!
5  while keep_going == 'y':
6      # Get a salesperson's sales and commission rate.
7      sales = float(input('Enter the amount of sales: '))
8      comm_rate = float(input('Enter the commission rate: '))
9      # Calculate the commission.
10     commission = sales * comm_rate
11     # Display the commission.
12     print(f'The commission is ${commission:,.2f}.')
```

Module Roadmap

1

Introduction to Repetition Structures

2

The while Loop: A Condition-Controlled Loop

3

The for Loop: A Count-Controlled Loop

4

Sentinels

5

Input Validation Loops

6

Nested Loops

The for Loop: A Count-Controlled Loop

A ***count-controlled*** loop iterates a specific number of times.

```
for variable in [value1, value2, etc.]:  
    statement  
    statement  
    etc.
```

The for Loop: A Count-Controlled Loop (Example)

1st iteration: `for num in [1, 2, 3, 4, 5]:`
`print(num)`

2nd iteration: `for num in [1, 2, 3, 4, 5]:`
`print(num)`

3rd iteration: `for num in [1, 2, 3, 4, 5]:`
`print(num)`

4th iteration: `for num in [1, 2, 3, 4, 5]:`
`print(num)`

5th iteration: `for num in [1, 2, 3, 4, 5]:`
`print(num)`



```
1 # This program demonstrates a simple for loop
2 # that uses a list of numbers.
3 print('I will display the numbers 1 through 5.')
4 for num in [1, 2, 3, 4, 5]:
5     print(num)
```


The for Loop: A Count-Controlled Loop (Example)




```
1 # This program also demonstrates a simple for
2 # loop that uses a list of strings.
3 for name in ['Shiraz', 'Tehran', 'Kish']:
4     print(name)
```


Using the range Function with the for Loop

The *range* function creates a type of object known as an iterable. An *iterable* is an object that is similar to a list. It contains a sequence of values that can be iterated over with something like a loop



```
1 for num in [0, 1, 2, 3, 4]:  
2     print(num)
```



```
1 for num in range(5):  
2     print(num)
```

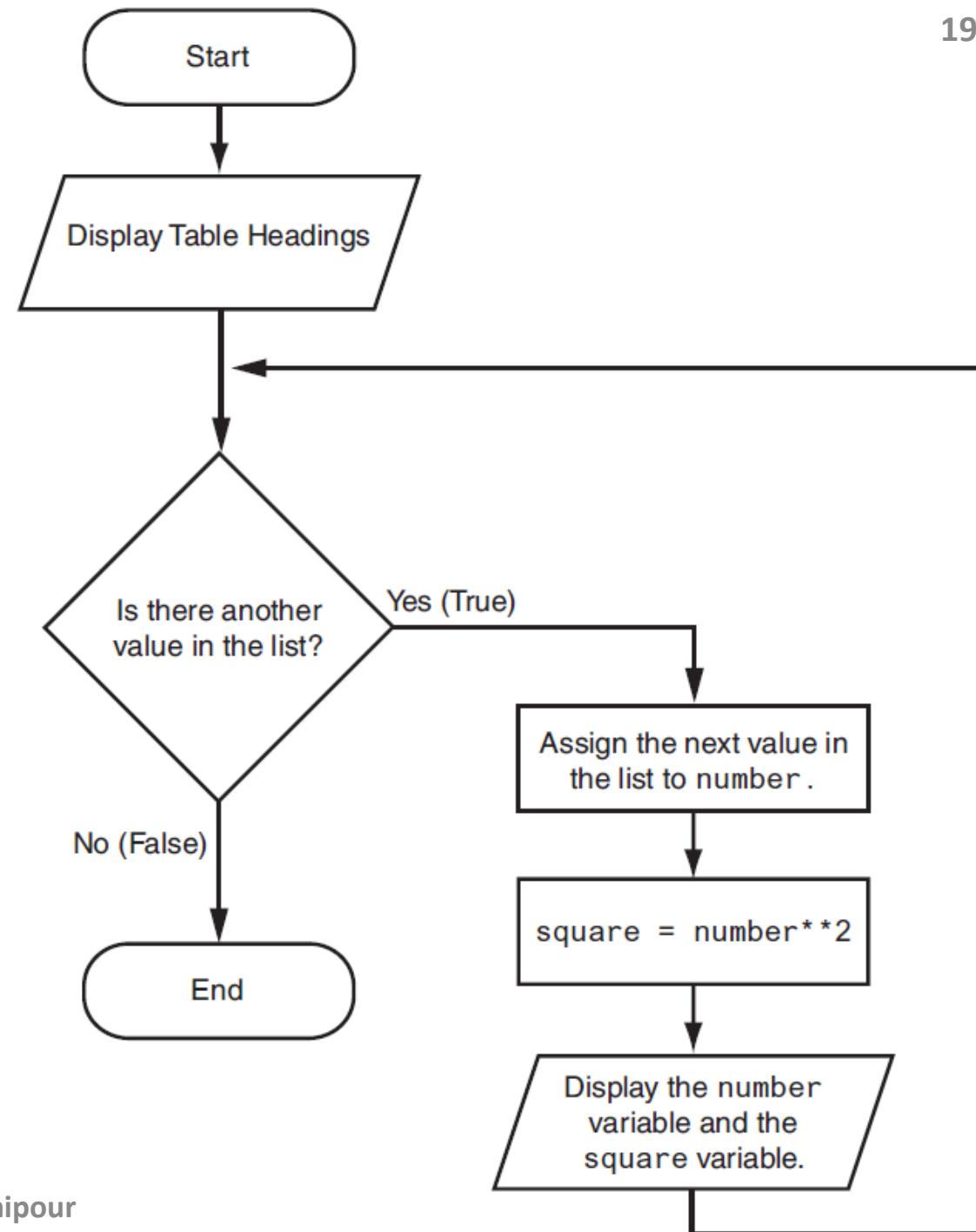
Using the range Function with the for Loop (Example)



```
1  # This program demonstrates how the range
2  # function can be used with a for loop.
3  # Print a message five times.
4  for x in range(5):
5      print('Hello world')
```

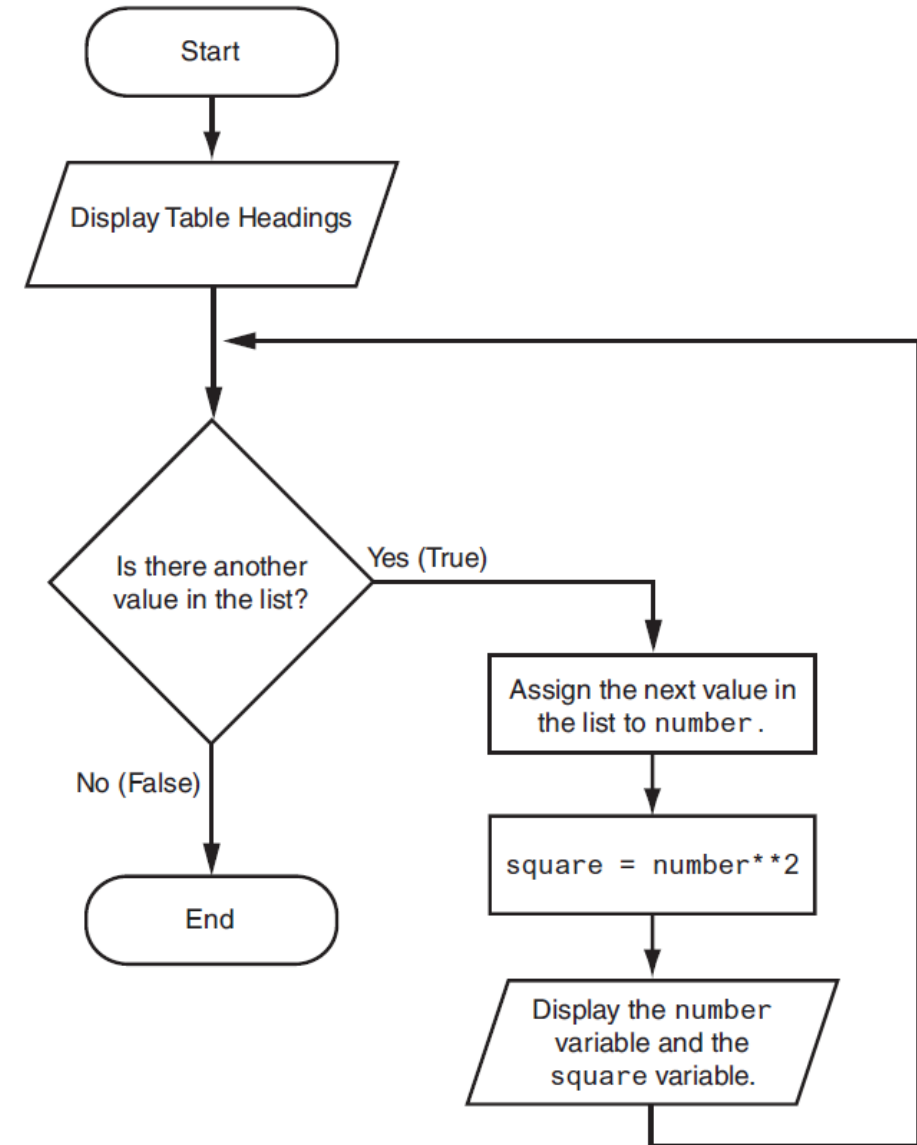
For Loop (Example)

suppose you need to write a program that displays the numbers 1 through 10 and their respective squares in a table.



For Loop (Example ...)

```
1  # This program uses a loop to display a
2  # table showing the numbers 1 through 10
3  # and their squares.
4  # Print the table headings.
5  print('Number\tSquare')
6  print('-----')
7  # Print the numbers 1 through 10
8  # and their squares.
9  for number in range(1, 11):
10     square = number**2
11     print(f'{number}\t{square}')
```



Letting the User Control the Loop Iterations

```
1  # This program uses a loop to display a
2  # table of numbers and their squares.
3  # Get the starting value.
4  print('This program displays a list of numbers')
5  print('and their squares.')
6  start = int(input('Enter the starting number: '))
7  # Get the ending limit.
8  end = int(input('How high should I go? '))
9  # Print the table headings.
10 print()
11 print('Number\tSquare')
12 print('-----')
13 # Print the numbers and their squares.
14 for number in range(start, end + 1):
15     square = number**2
16     print(f'{number}\t{square}')
```

Ali Samanipour

[linkedin.com/in/Samanipour](https://www.linkedin.com/in/Samanipour)

Module Roadmap

- 1 Introduction to Repetition Structures
- 2 The while Loop: A Condition-Controlled Loop
- 3 The for Loop: A Count-Controlled Loop
- 4 Sentinels
- 5 Input Validation Loops
- 6 Nested Loops

Sentinels (The Problem)

You are designing a program that will use a loop to process a long sequence of values. At the time you are designing the program, you do not know the number of values that will be in the sequence

In fact, the number of values in the sequence could be different each time the program is executed

Sentinels (The Problem and Tow **Bad** Solution!)

You are designing a program that will use a loop to process a long sequence of values. At the time you are designing the program, you do not know the number of values that will be in the sequence

Simply ask the user, at the end of each loop iteration

Ask the user at the beginning of the program how many items are in the sequence.

Sentinels (The Problem and a Best Practice!)

A ***sentinel*** is a special value that marks the end of a sequence of values.

A sentinel is a special value that marks the end of a sequence of items. When a program reads the sentinel value, it knows it has reached the end of the sequence, so the loop terminates

Sentinels (Example)

```
1 # This program displays property taxes.
2 # Represents the tax factor.
3 TAX_FACTOR = 0.0065
4 # Get the first lot number.
5 print('Enter the property lot number or enter 0 to end.')
6 lot = int(input('Lot number: '))
7 # Continue processing as long as the user
8 # does not enter lot number 0.
9 while lot != 0:
10     # Get the property value.
11     value = float(input('Enter the property value: '))
12     # Calculate the property's tax.
13     tax = value * TAX_FACTOR
14     # Display the tax.
15     print(f'Property tax: ${tax:,.2f}')
16     # Get the next lot number.
17     print('Enter the next lot number or enter 0 to end.')
18     lot = int(input('Lot number: '))
```

Module Roadmap

- 1 Introduction to Repetition Structures
- 2 The while Loop: A Condition-Controlled Loop
- 3 The for Loop: A Count-Controlled Loop
- 4 Sentinels
- 5 Input Validation Loops
- 6 Nested Loops

Input validation

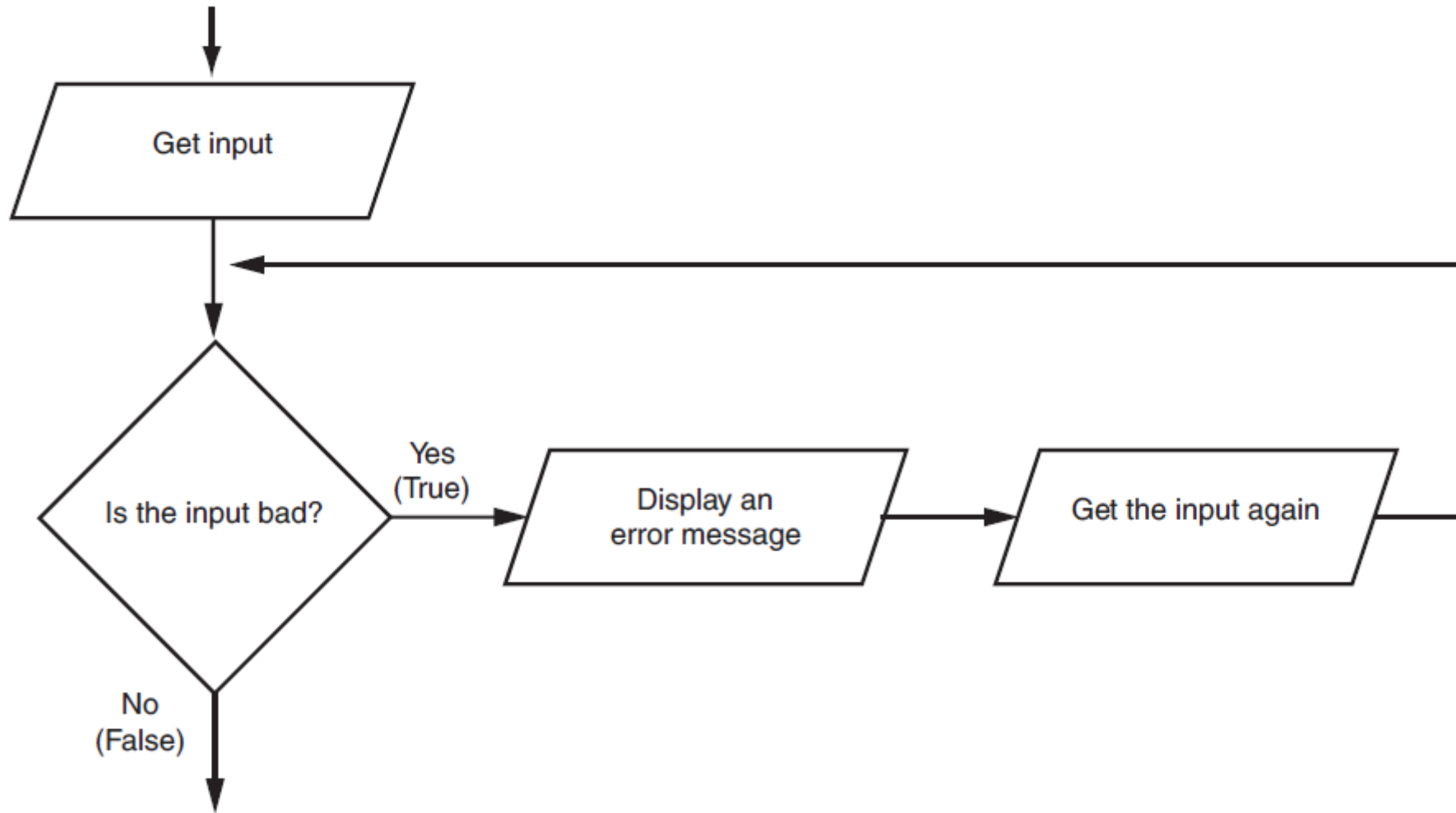
Input ***validation*** is the process of inspecting data that has been input to a program, to make sure it is valid before it is used in a computation.

Input validation (An Example Without Validation)



```
1 # This program displays gross pay.
2 # Get the number of hours worked.
3 hours = int(input('Enter the hours worked this week: '))
4 # Get the hourly pay rate.
5 pay_rate = float(input('Enter the hourly pay rate: '))
6 # Calculate the gross pay.
7 gross_pay = hours * pay_rate
8 # Display the gross pay.
9 print(f'Gross pay: ${gross_pay:,.2f}')
```

Logic containing an input validation loop



Input validation (An Example Without Validation)

```
1 # This program calculates retail prices.
2 MARK_UP = 2.5 # The markup percentage
3 another = 'y' # Variable to control the loop.
4 # Process one or more items.
5 while another == 'y' or another == 'Y':
6     # Get the item's wholesale cost.
7     wholesale = float(input("Enter the item's wholesale cost: "))
8     # Calculate the retail price.
9     retail = wholesale * MARK_UP
10    # Display the retail price.
11    print(f'Retail price: ${retail:,.2f}')
12    # Do this again?
13    another = input('Do you have another item? (Enter y for yes): ')
```

Input validation (An Example With Validation)

```
1  # This program calculates retail prices.
2  MARK_UP = 2.5 # The markup percentage
3  another = 'y' # Variable to control the loop.
4  # Process one or more items.
5  while another == 'y' or another == 'Y':
6      # Get the item's wholesale cost.
7      wholesale = float(input("Enter the item's wholesale cost: "))
8      # Validate the wholesale cost.
9      while wholesale < 0:
10         print('ERROR: the cost cannot be negative.')
11         wholesale = float(input('Enter the correct wholesale cost: '))
12     # Calculate the retail price.
13     retail = wholesale * MARK_UP
14     # Display the retail price.
15     print(f'Retail price: ${retail:,.2f}')
16     # Do this again?
17     another = input('Do you have another item? (Enter y for yes): ')
```


Module Roadmap

1

Introduction to Repetition Structures

2

The while Loop: A Condition-Controlled Loop

3

The for Loop: A Count-Controlled Loop

4

Sentinels

5


Input Validation Loops

6

Nested Loops

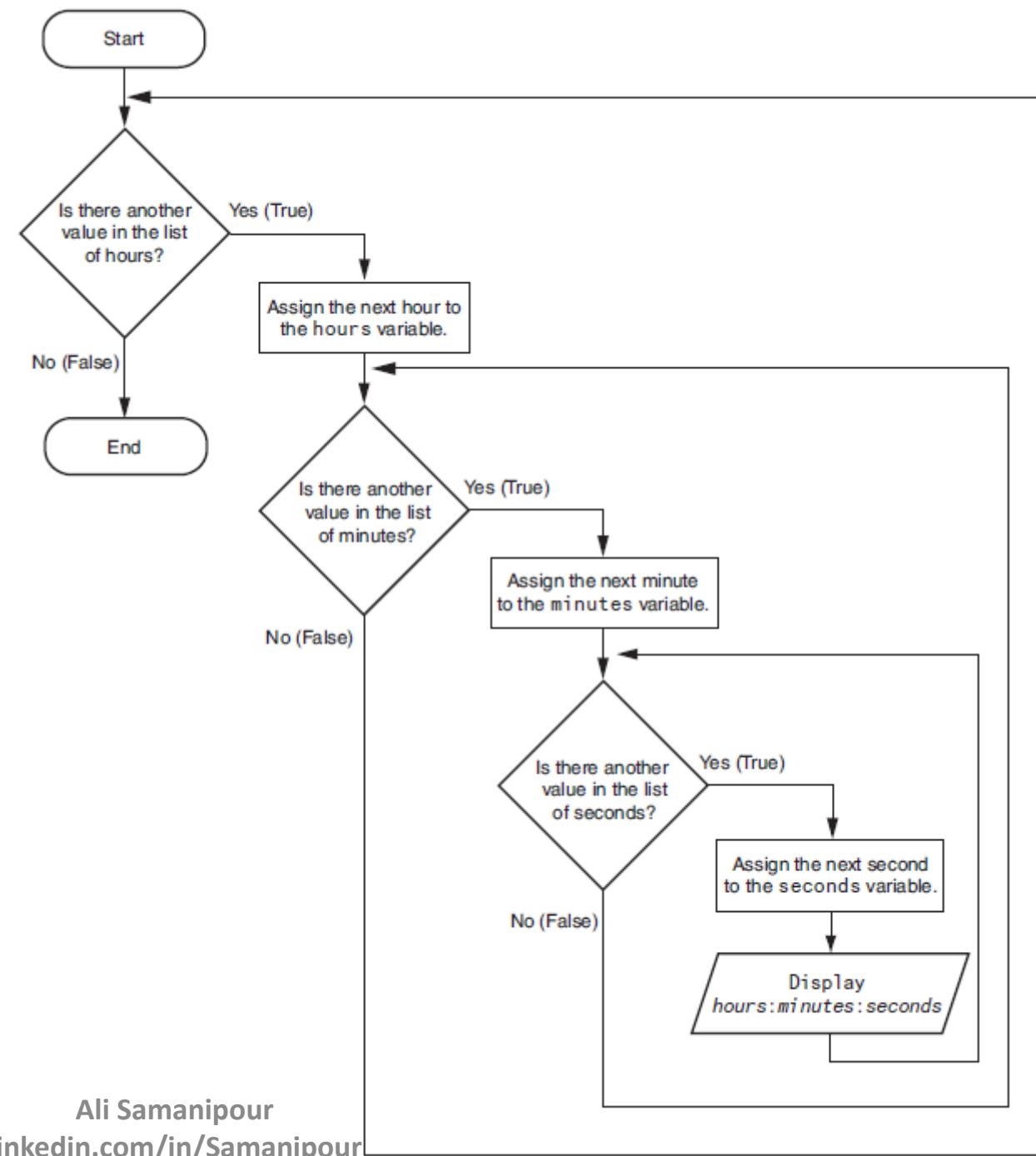
Nested Loops

A loop that is inside another loop is called a ***nested loop***.



```
1 # This program simulating a simple digital clock
2 from time import sleep
3 for hours in range(24):
4     for minutes in range(60):
5         for seconds in range(60):
6             print(hours, ':', minutes, ':', seconds)
7             sleep(1)
```

Nested Loops (Digital Clock logic)



Nested Loops (Test Score Program Example)

```
1 # This program averages test scores. It asks the user for the
2 # number of students and the number of test scores per student.
3 # Get the number of students.
4 num_students = int(input('How many students do you have? '))
5
6 # Get the number of test scores per student.
7 num_test_scores = int(input('How many test scores per student? '))
8
9 # Determine each student's average test score.
10 for student in range(num_students):
11     # Initialize an accumulator for the test scores.
12     total = 0.0
13     # Display the student number.
14     print(f'Student number {student + 1}')
15     print('-----')
16
17     # Get the student's test scores.
18     for test_num in range(num_test_scores):
19         print(f'Test number {test_num + 1}', end='')
20         score = float(input(': '))
21         # Add the score to the accumulator.
22         total += score
23
24     # Calculate the average test score for this student.
25     average = total / num_test_scores
26     # Display the average.
27     print(f'The average for student number {student + 1} ' + f'is: {average:.1f}')
28     print()
```

Appendix : Controlling Loops Using continue and break

for

A Count-Controlled Loop

while

A Condition-Controlled Loop

Use **break** to **exit the loop** before it's finish

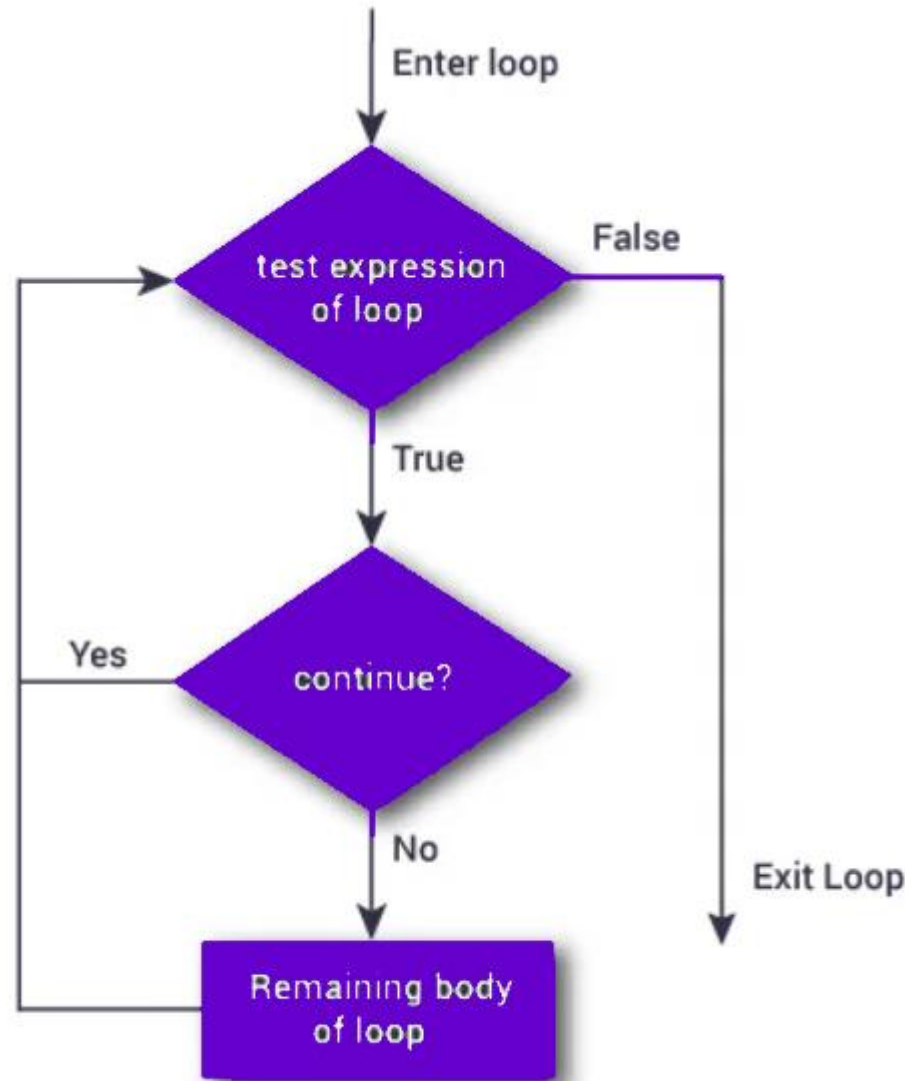
Use **continue** to skip an iteration



Avoid using break and continue as much as possible



Appendix: Controlling loops (continue)



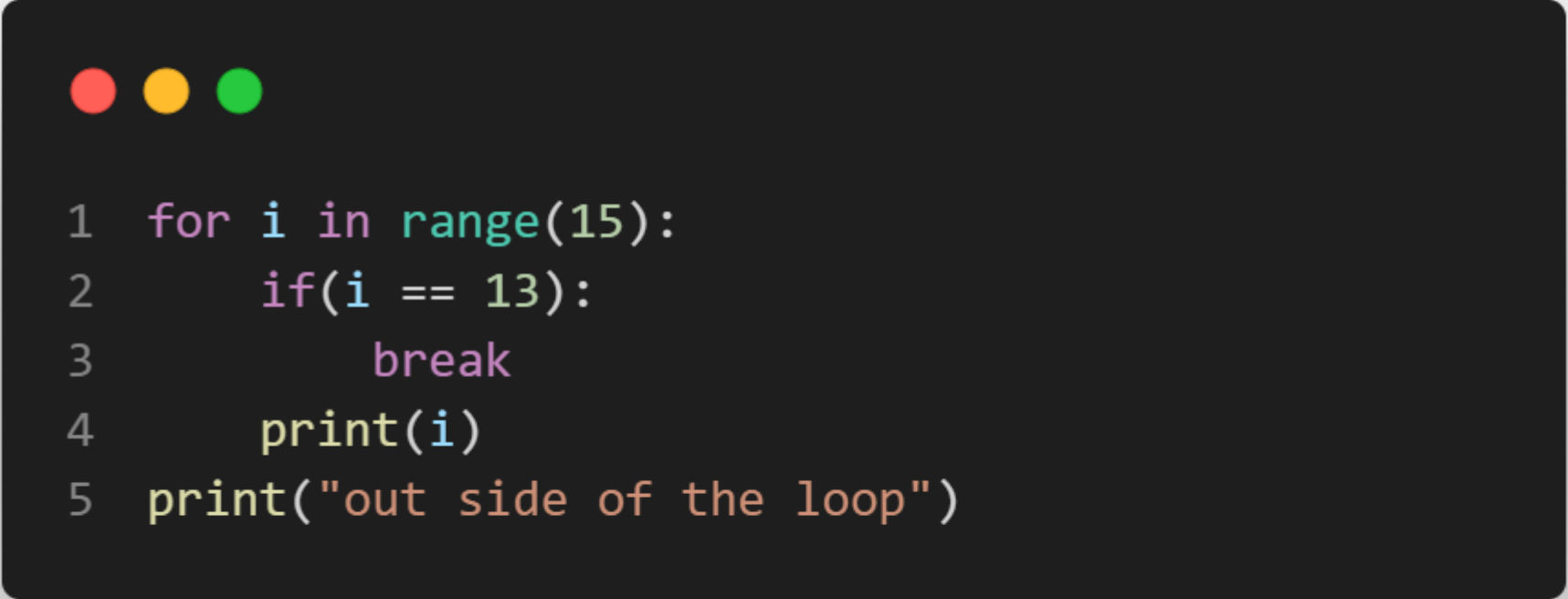
```
for var in sequence:
    # codes inside for loop
    if condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if condition:
        continue
    # codes inside while loop


# codes outside while loop
```

Appendix: Lets try!



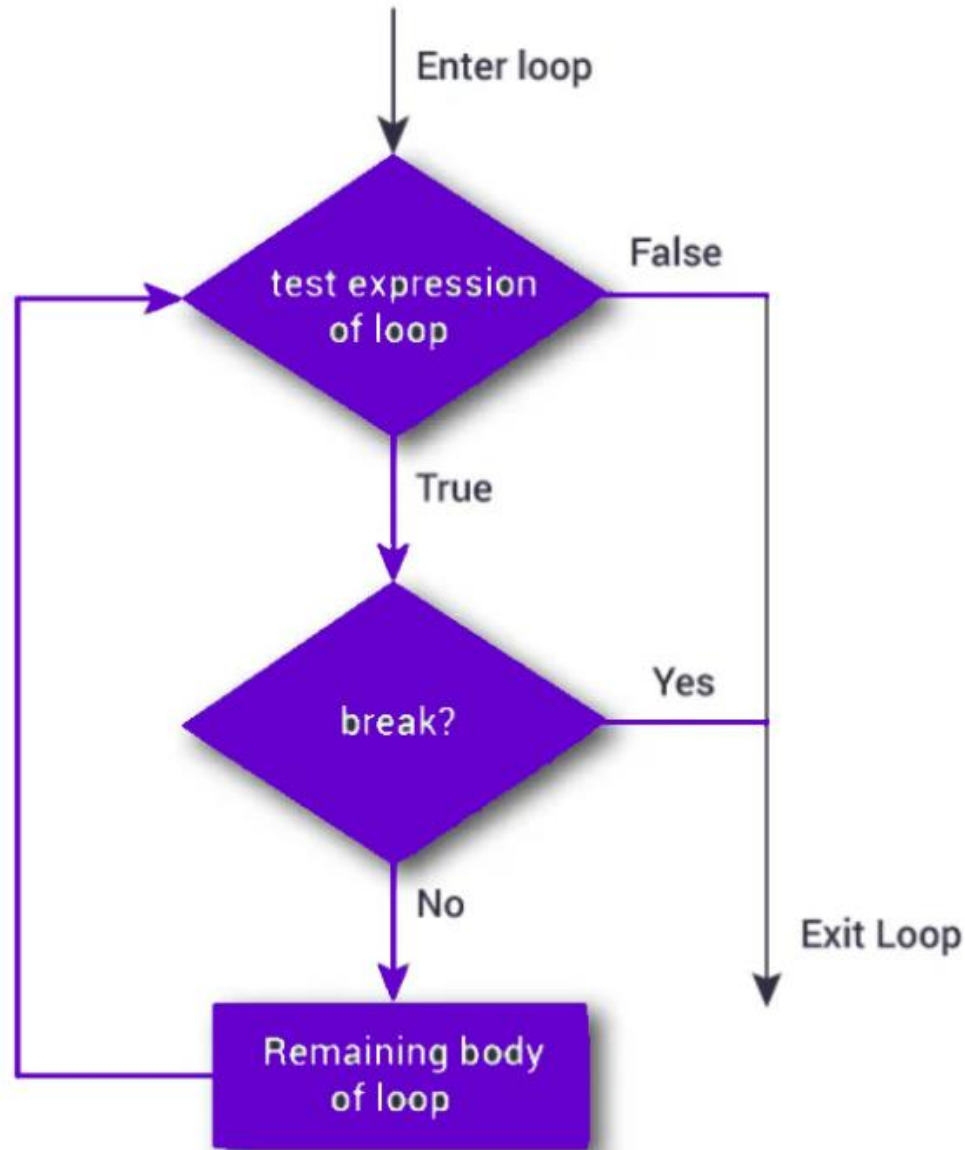
```
1  for i in range(15):  
2      if(i == 13):  
3          break  
4      print(i)  
5  print("out side of the loop")
```

Appendix: Lets try!



```
1 while True:
2     user_input = input("Please enter a String ! : ")
3     if(user_input == "Ali"):
4         continue
5     print("Your input is : ",user_input)
6 print("outside of the loop") # this line of code never excute!
```

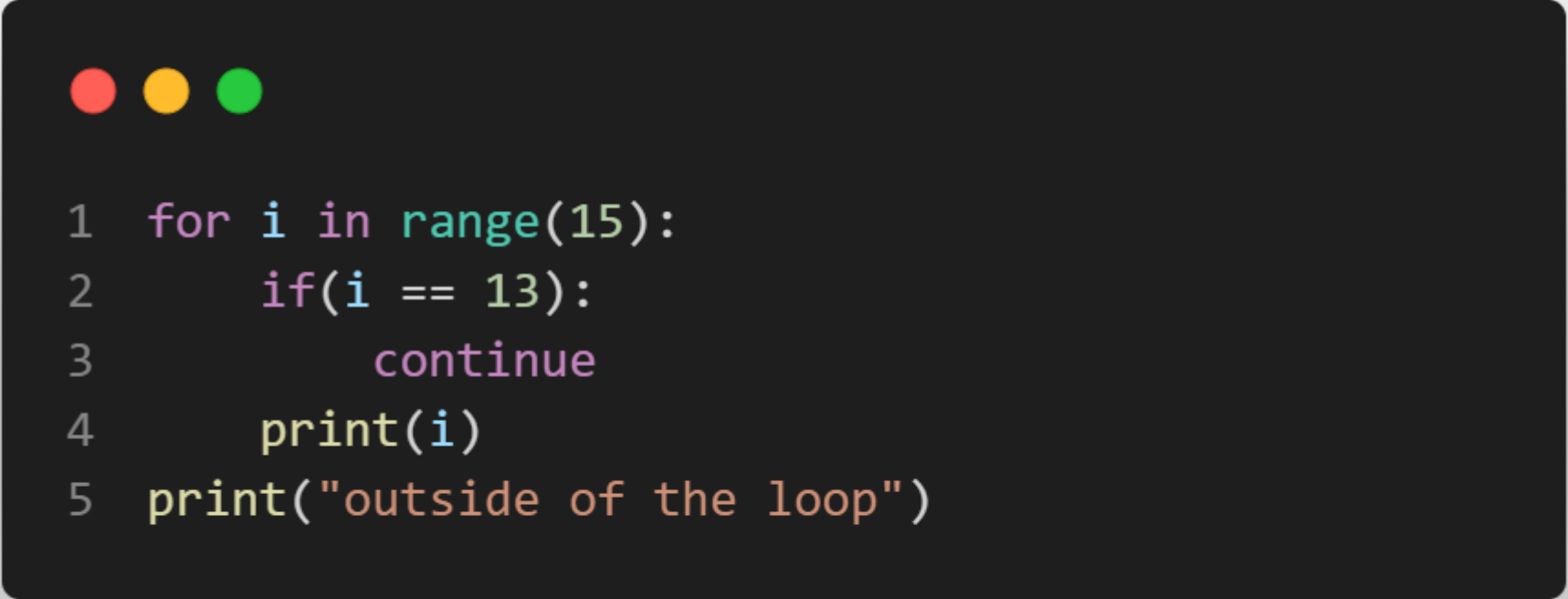

Appendix: Controlling loops (break)



```
for var in sequence:
    # codes inside for loop
    if condition:
        break
    # codes inside for loop
# codes outside for loop
```


```
while test expression:
    # codes inside while loop
    if condition:
        break
    # codes inside while loop
# codes outside while loop
```

Appendix: Lets try!



```
1  for i in range(15):  
2      if(i == 13):  
3          continue  
4      print(i)  
5  print("outside of the loop")
```

Appendix: Lets try!



```
1 while True:
2     user_input = input("Please enter a String ! : ")
3     if(user_input == "exit"):
4         break
5     print("Your input is : ",user_input)
6 print("outside of the loop")
```

Accessing Resources (Codes, Slides, etc.)



github.com/Samanipour



t.me/SamaniGroup

References

- **Automate The Boring Stuff With Python, 2nd Edition: Practical Programming For Total Beginners**
- **Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction To Programming**
- **Supercharged Python: Take Your Code to the Next Level**
- **Python Tricks: A Buffet of Awesome Python Features**
- **Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code**
- **Starting Out with Python, Tony Gaddis, Global Edition**