GET214: COMPUTING AND SOFTWARE ENGINEERING

Lesson 4

LOOPS

Learning outcomes

At the end of the lesson, students should be able:

1. Explain the two loop constructs in Python
2. Use a while loop to implement repeated tasks
3. Use a for loop to implement repeated tasks
4. Use the range function with the for loop
5. Implement nested while loop
6. Implement nested for loop
7. Use the break and continue statements in while and for loops
8. Implement a loop else statement with a for or a while loop

Introduction to loops

- A **loop** is a code block that runs a set of statements while a given condition is true.
- A loop is often used for performing a repeating task.
- Python has two loop constructs to handle iterative structures.
- The for loop is mostly used in situations where the actual number of iterations or number of items required to iterate over is known at program time.
- If however, the number of iterations can only be known at run time, the while loop is preferred.

The while loop

- A **while loop** is a code construct that runs a set of statements, known as the loop body, while a given condition, known as the loop expression, is true.
- At each iteration, once the loop statement is executed, the loop expression is evaluated again.
- If true, the loop body will execute at least one more time (also called looping or iterating one more time).
- If false, the loop's execution will terminate and the next statement after the loop body will execute.

Counting with while loop

- A while loop can be used to count up or down.
- A counter variable can be used in the loop expression to determine the number of iterations executed.
- A programmer may want to print all even numbers between 1 and 20.
- The task can be done by using a counter initialized with 1.
- In each iteration, the counter's value is increased by one, and a condition can check whether the counter's value is an even number or not.

Counting with a step value

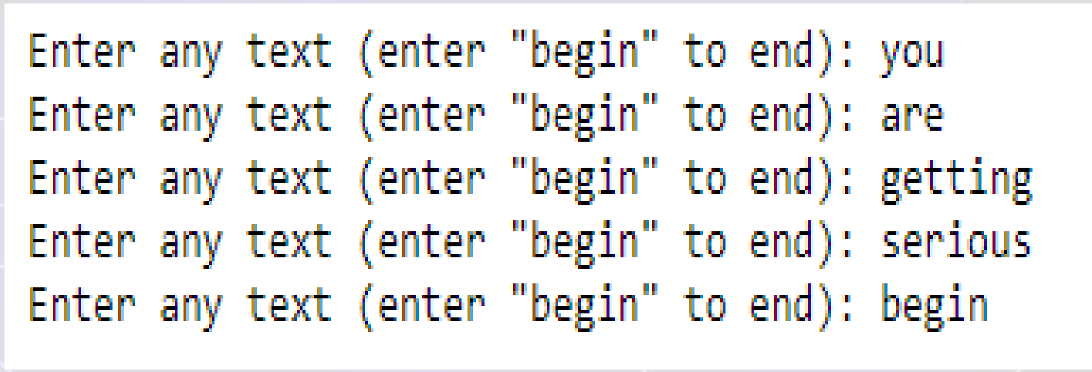
- The change in the counter's value in each iteration is called the **step size**.
- The step size can be any positive or negative value.
- If the step size is a positive number, the counter counts in ascending order, and if the step size is a negative number, the counter counts in descending order.

Example 4.1: A program printing all odd numbers between 1 and 10

```
# Initialization
counter = 1
# While loop condition
while counter <= 10:
    if counter % 2 == 1:
        print(counter)
# Counting up and increasing counter's value by
#1 in each iteration
    counter += 1
```

Activity 4.1: Reading inputs in a while loop

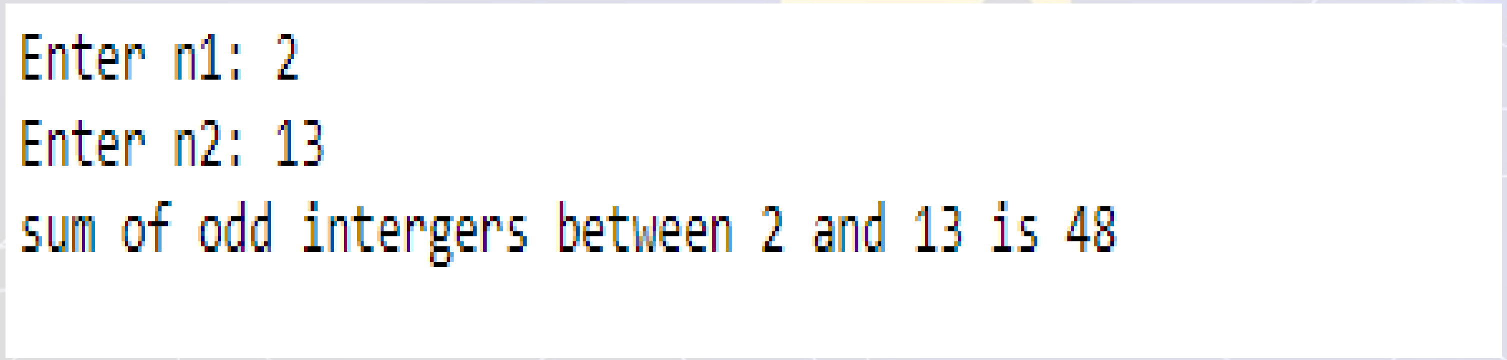
- Write a program that takes user inputs in a while loop until the user enters "begin".
- Test the code with the given input values to check that the loop does not terminate until the input is "begin".
- Once the input "begin" is received, print "The while loop condition has been met.".
- Enter different input words to see when the while loop condition is met.
- See sample output :



```
Enter any text (enter "begin" to end): you
Enter any text (enter "begin" to end): are
Enter any text (enter "begin" to end): getting
Enter any text (enter "begin" to end): serious
Enter any text (enter "begin" to end): begin
```

Activity 4.2: Sum of odd numbers

- Write a program that reads two integer values, n1 and n2.
- Use a while loop to calculate the sum of odd numbers between n1 and n2 (inclusive of n1 and n2).
- Remember, a number is odd if number % 2!= 0.
- See sample output below



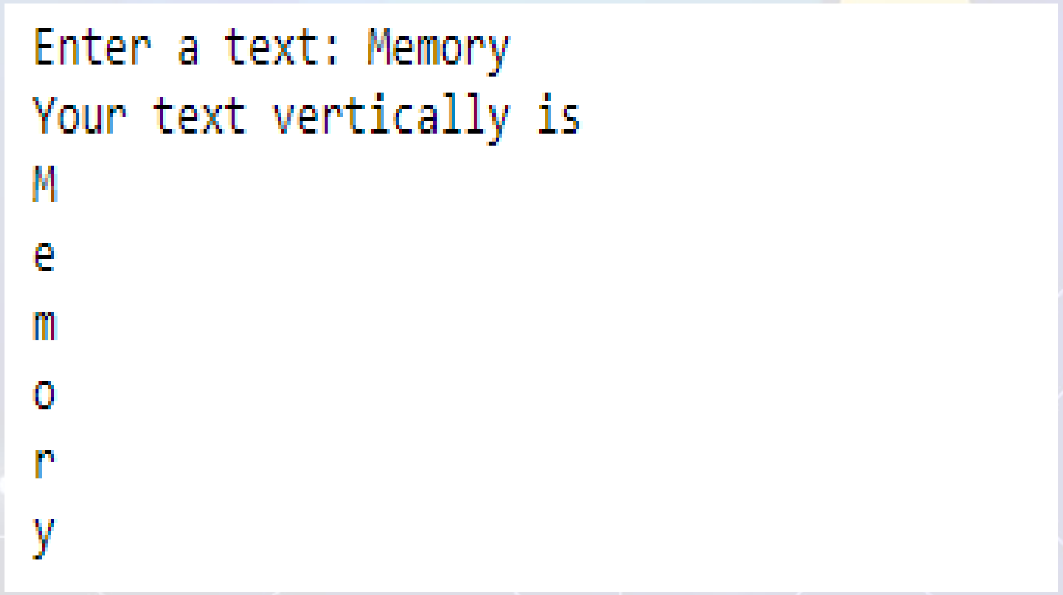
```
Enter n1: 2
Enter n2: 13
sum of odd intergers between 2 and 13 is 48
```

The for loop

- In Python, a **container** can be a range of numbers, a string of characters, or a list of values.
- To access objects within a container, an iterative loop can be designed to retrieve objects one at a time.
- A **for loop** iterates over all elements in a container.
- For example, iterating over a class roster and printing students' names.

Example 4.2: Printing string vertically

- Write a python program that calls for user input of a string and print the string vertically.
- See sample output below.

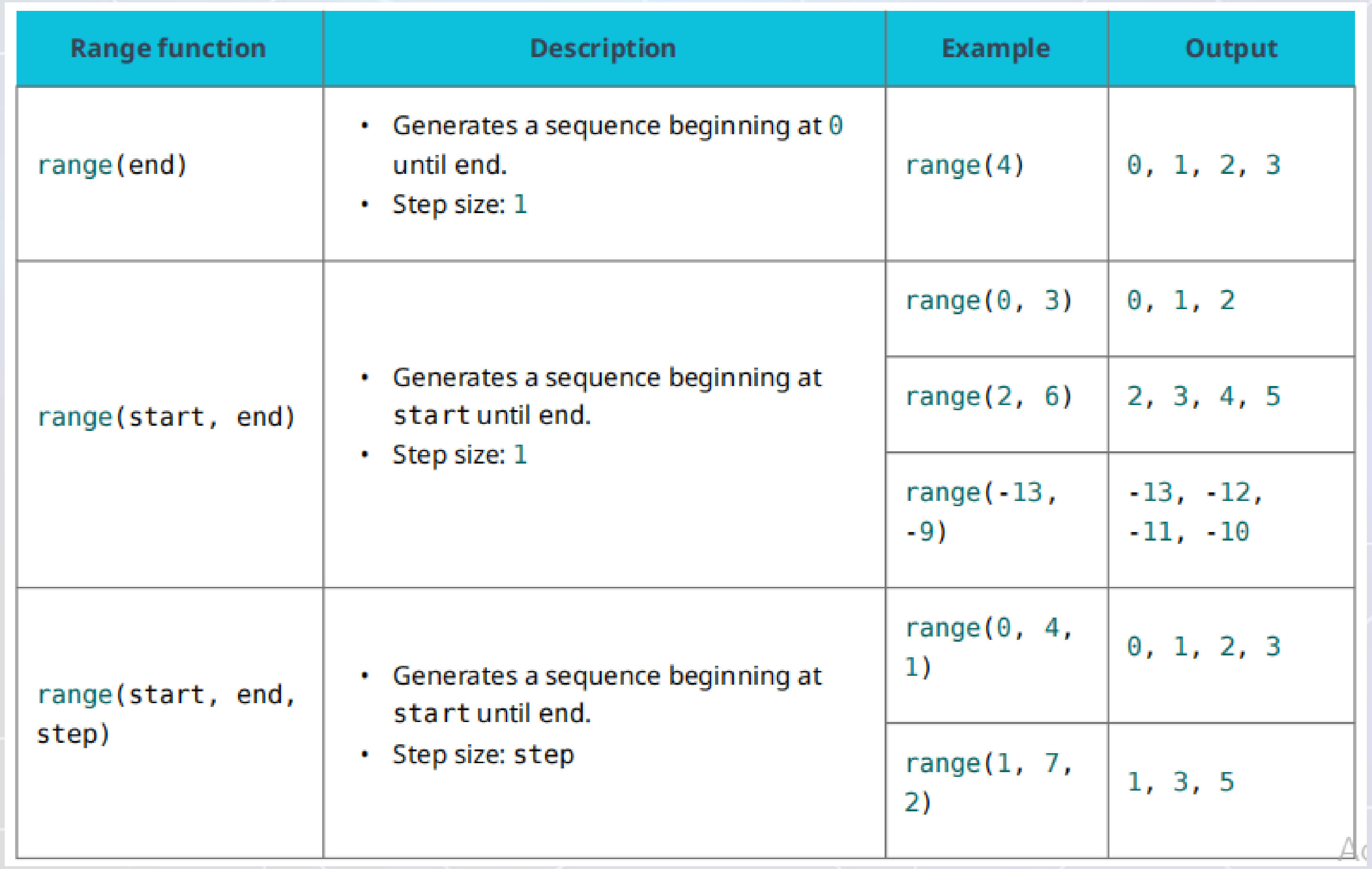


```
Enter a text: Memory
Your text vertically is
M
e
m
o
r
y
```

The range() function

- A for loop can be used for iteration and counting.
- The range() function is a common approach for implementing counting in a for loop.
- A **range()** function generates a sequence of integers between the two numbers given a step size.
- This integer sequence is inclusive of the start and exclusive of the end of the sequence.
- The range() function can take up to three input values.

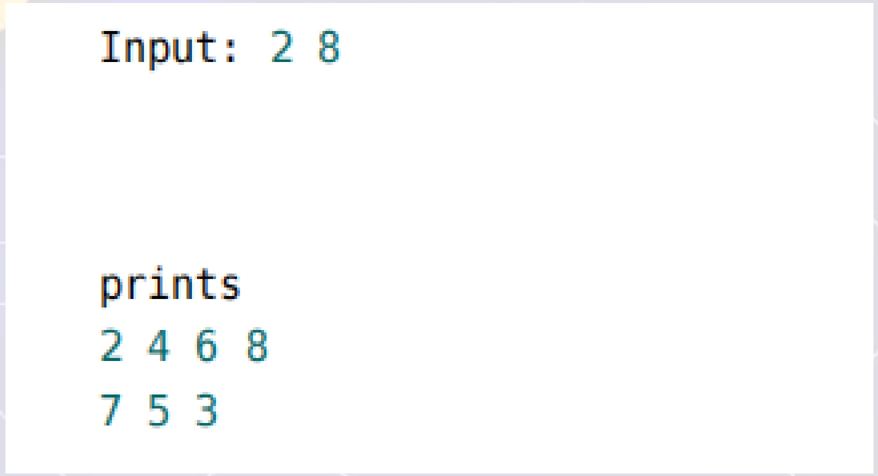
Use of the range() function



Range function	Description	Example	Output
`range(end)`	• Generates a sequence beginning at `0` until end. • Step size: `1`	`range(4)`	`0, 1, 2, 3`
`range(start, end)`	• Generates a sequence beginning at `start` until end. • Step size: `1`	`range(0, 3)`	`0, 1, 2`
		`range(2, 6)`	`2, 3, 4, 5`
		`range(-13, -9)`	`-13, -12, -11, -10`
`range(start, end, step)`	• Generates a sequence beginning at `start` until end. • Step size: `step`	`range(0, 4, 1)`	`0, 1, 2, 3`
		`range(1, 7, 2)`	`1, 3, 5`

Activity 4.3: Sequences

- Write a program that reads two integer values, n1 and n2, with n1 < n2, and performs the following tasks:
- Prints all even numbers between the two provided numbers (inclusive of both), in ascending order.
- Prints all odd numbers between the two provided numbers (exclusive of both), in descending order.
- See sample output



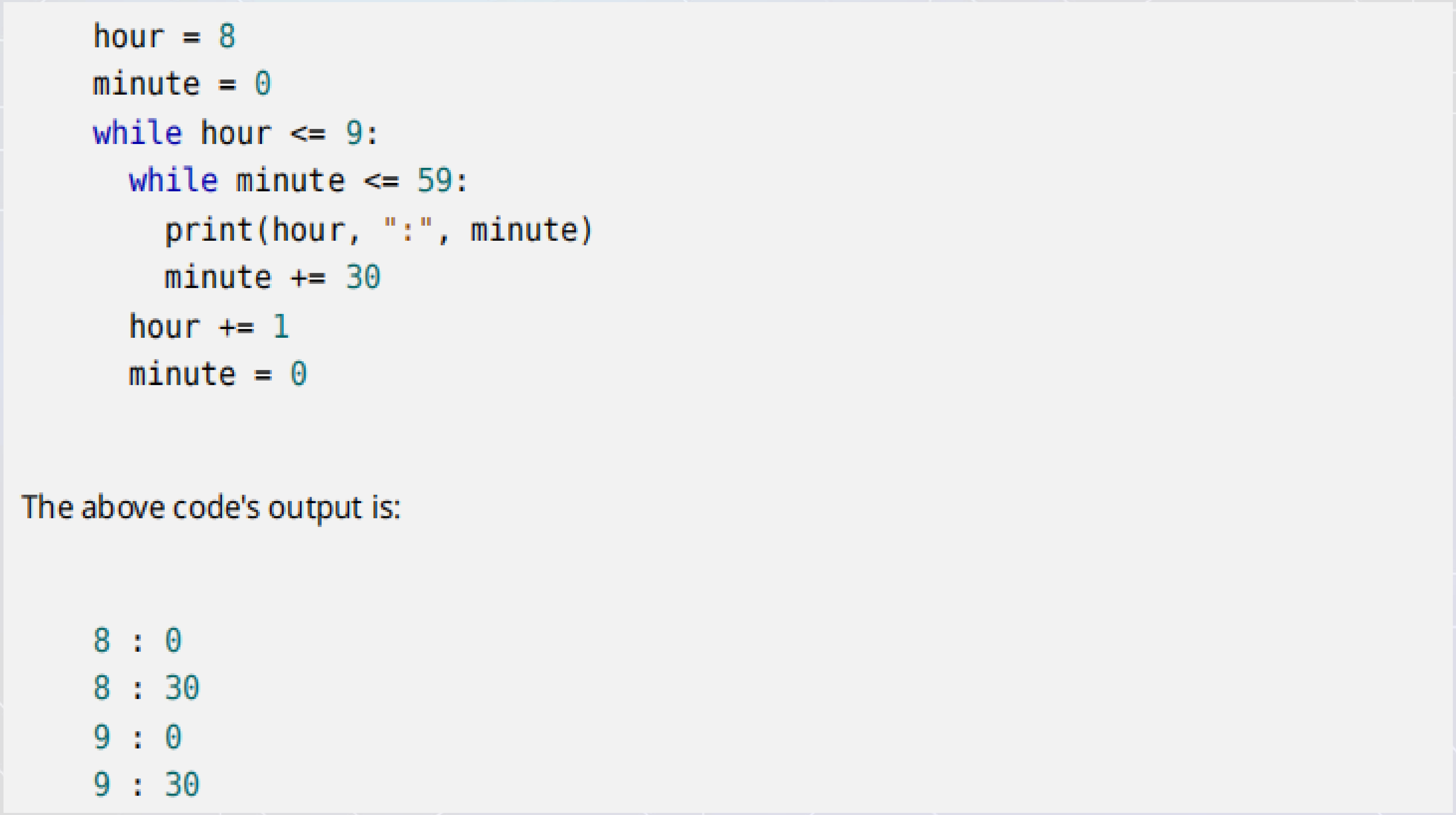
```
Input: 2 8



prints
2 4 6 8
7 5 3
```

Nested Loops

- A **nested loop** has one or more loops within the body of another loop.
- The two loops are referred to as **outer loop** and **inner loop**.
- The outer loop controls the number of the inner loop's full execution.
- More than one inner loop can exist in a nested loop.

Example 4.3: Available appointments

- Consider a doctor's office schedule.
- Each appointment is 30 minutes long.
- A program to print available appointments can use a nested for loop where the outer loop iterates over the hours, and the inner loop iterates over the minutes.
- This example prints time in hours and minutes in the range between 8:00am and 10:00am.
- In this example, the outer loop iterates over the time's hour portion between 8 and 9, and the inner loop iterates over the time's minute portion between 0 and 59.



```
hour = 8
minute = 0
while hour <= 9:
  while minute <= 59:
    print(hour, ":", minute)
    minute += 30
  hour += 1
  minute = 0
```

The above code's output is:

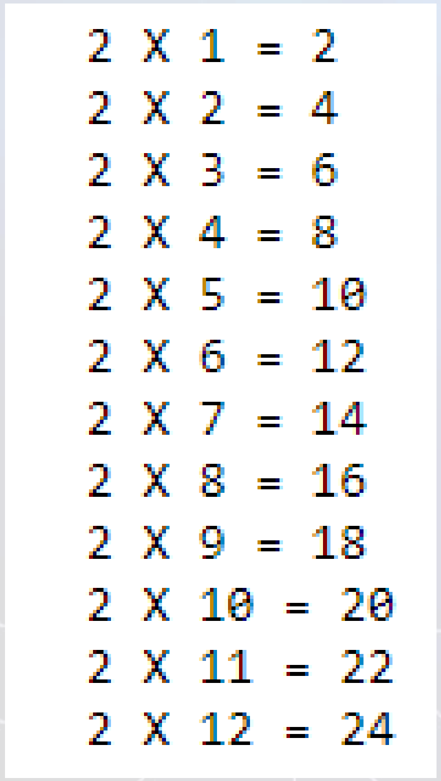
```
8 : 0
8 : 30
9 : 0
9 : 30
```

Nested for loop

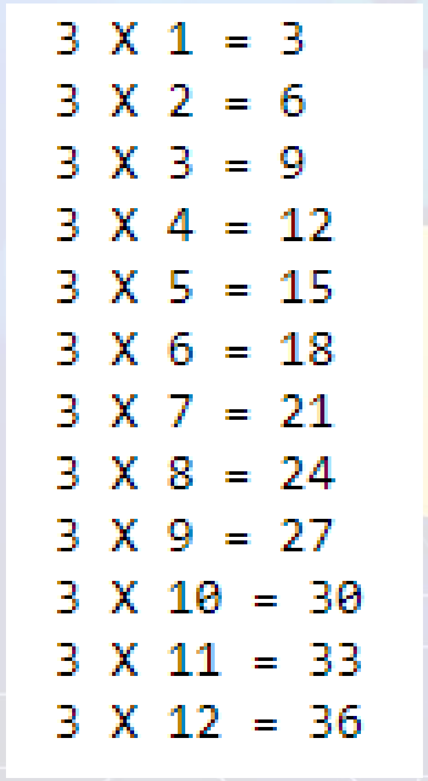
- A nested for loop can be implemented and used in the same way as a nested while loop.
- A for loop is a preferable option in cases where a loop is used for counting purposes using a range() function, or when iterating over a container object, including nested situations.
- Ex: Iterating over multiple course rosters.
- The outer loop iterates over different courses, and the inner loop iterates over the names in each course roster.

Example 4.4: Multiplication table

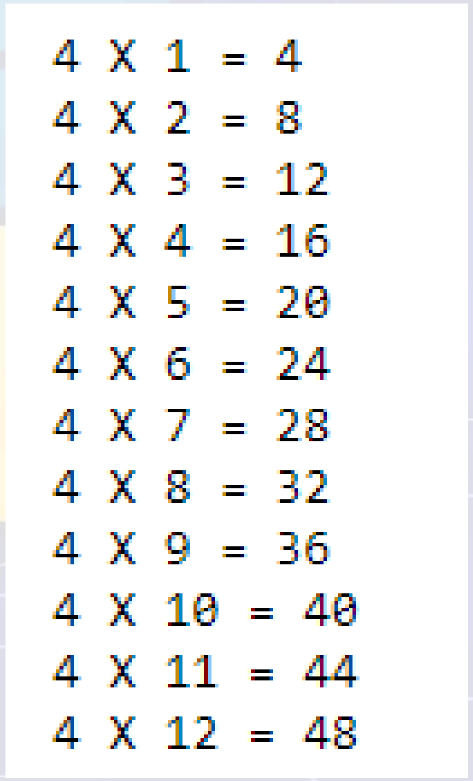
- Write a program to print the multiplication tables of 2,3,4 and 5.



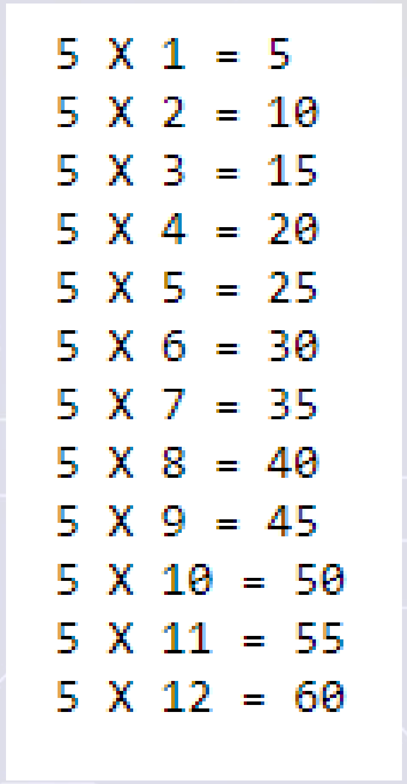
```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
2 X 11 = 22
2 X 12 = 24
```



```
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
3 X 11 = 33
3 X 12 = 36
```

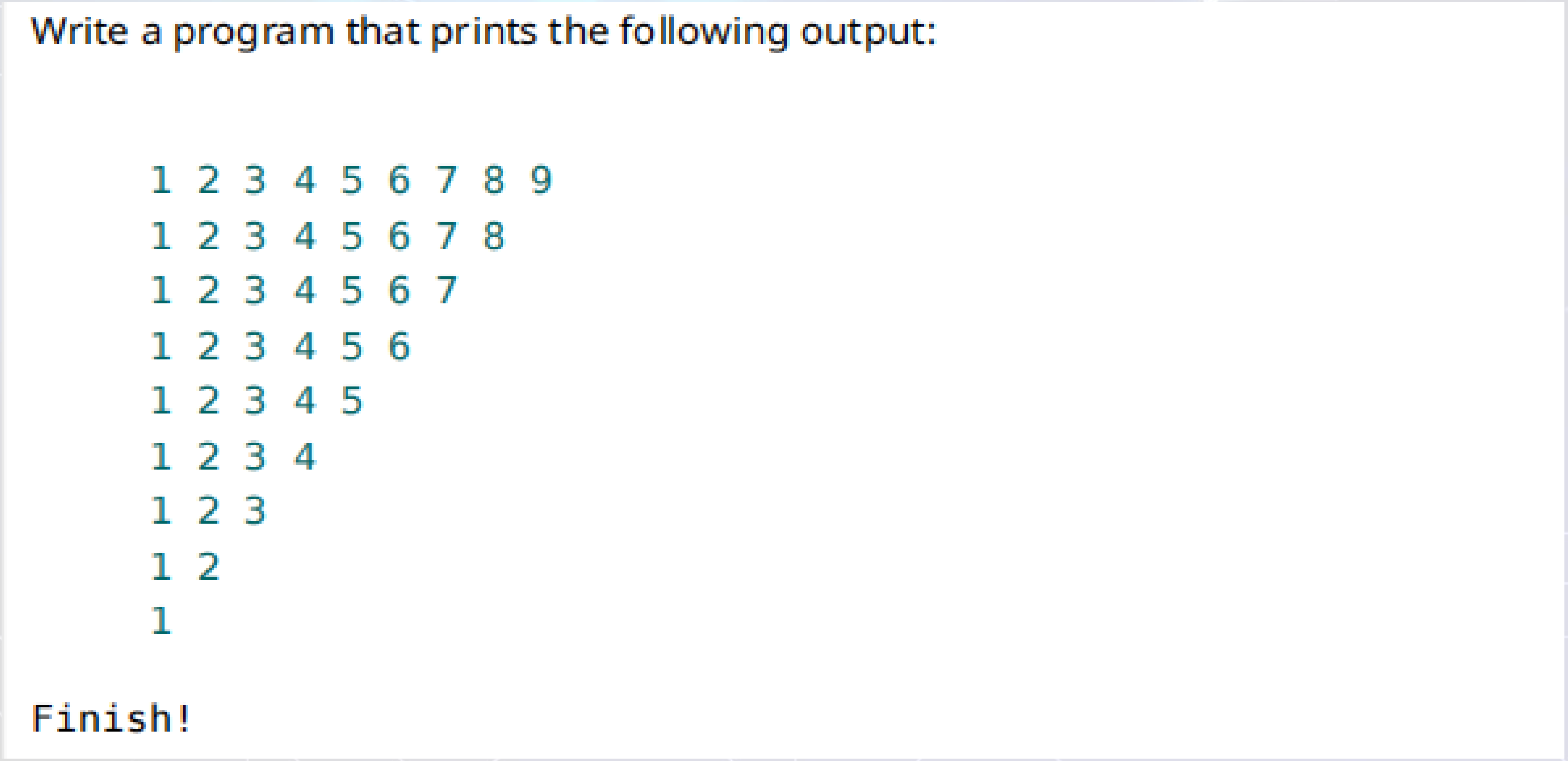


```
4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
4 X 11 = 44
4 X 12 = 48
```



```
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
5 X 4 = 20
5 X 5 = 25
5 X 6 = 30
5 X 7 = 35
5 X 8 = 40
5 X 9 = 45
5 X 10 = 50
5 X 11 = 55
5 X 12 = 60
```

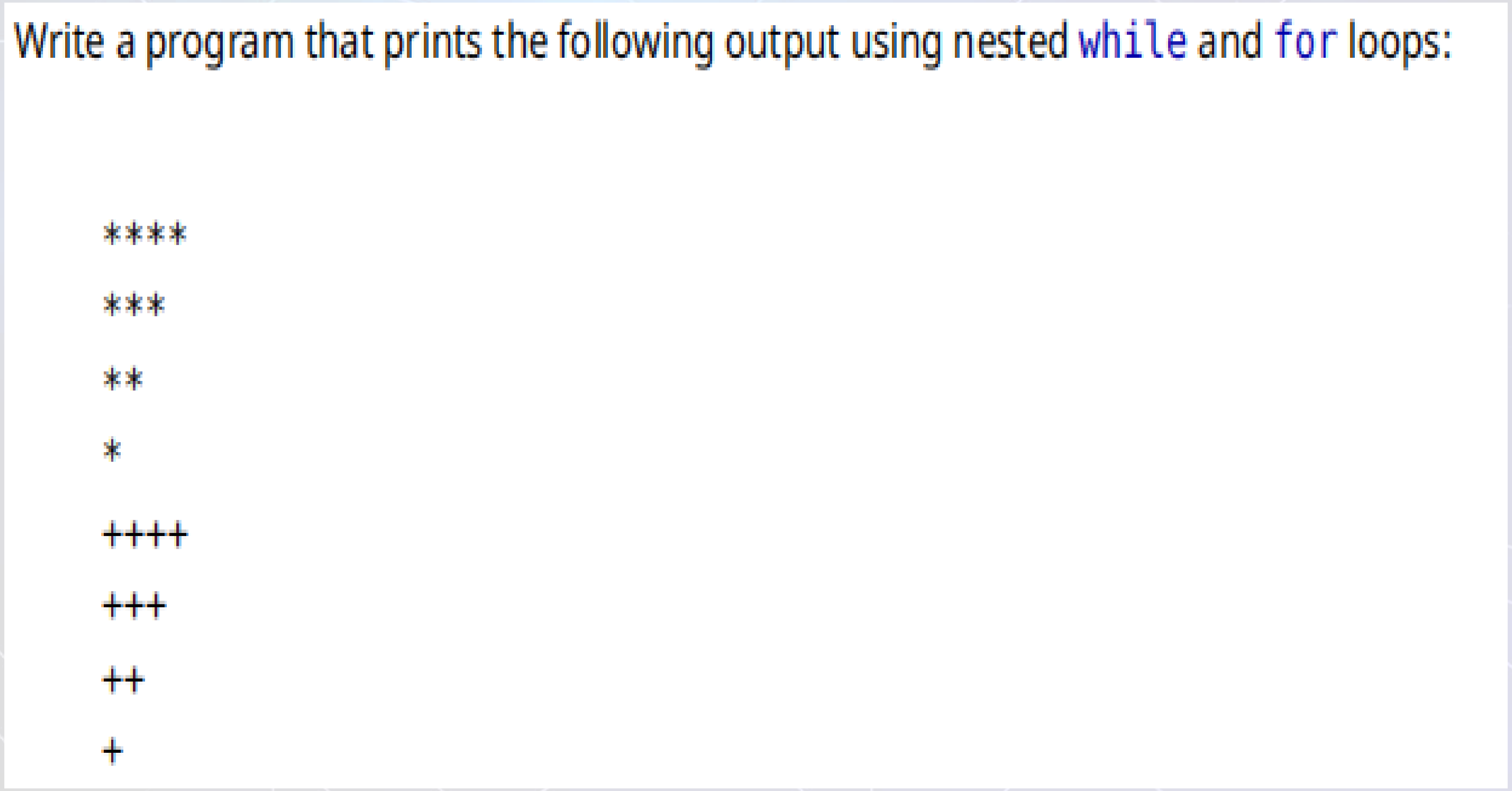
Activity 4.4: Printing a triangle of numbers



Write a program that prints the following output:

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

Finish!
```

Activity: 4.5: Printing two triangles



Write a program that prints the following output using nested while and for loops:

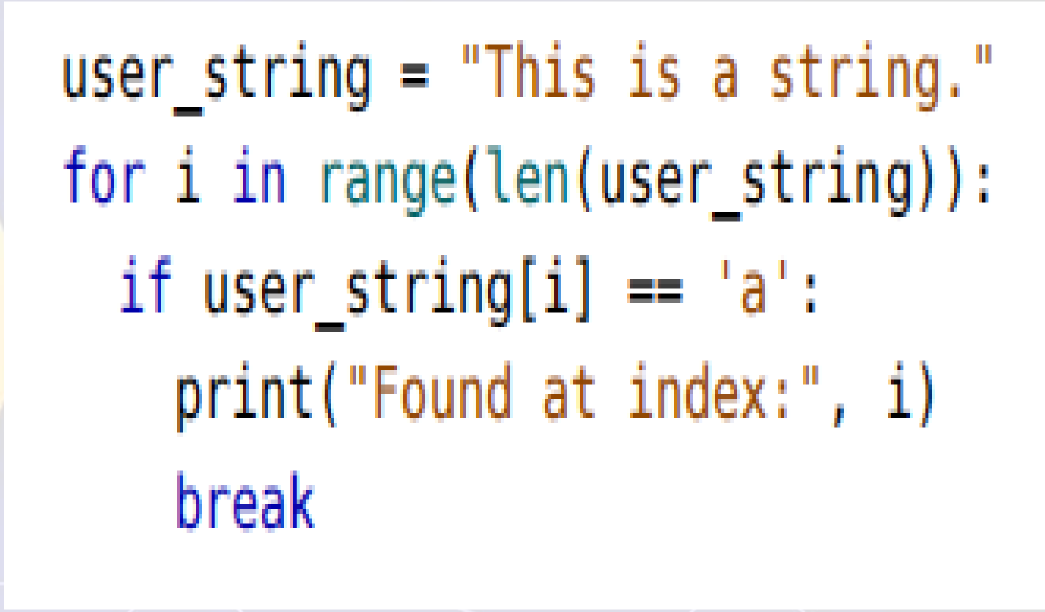
```
****
***
**
*
++++
+++
++
+
```

The break statement

- A **break** statement is used within a for or a while loop to allow the program execution to exit the loop once a given condition is triggered.
- A break statement can be used to improve runtime efficiency when further loop execution is not required.

Example:

- A loop that looks for the character "a" in a given string called user_string.
- The loop below is a regular for loop for going through all the characters of user_string.
- If the character is found, the break statement takes execution out of the for loop.
- Since the task has been accomplished, the rest of the for loop execution is bypassed.



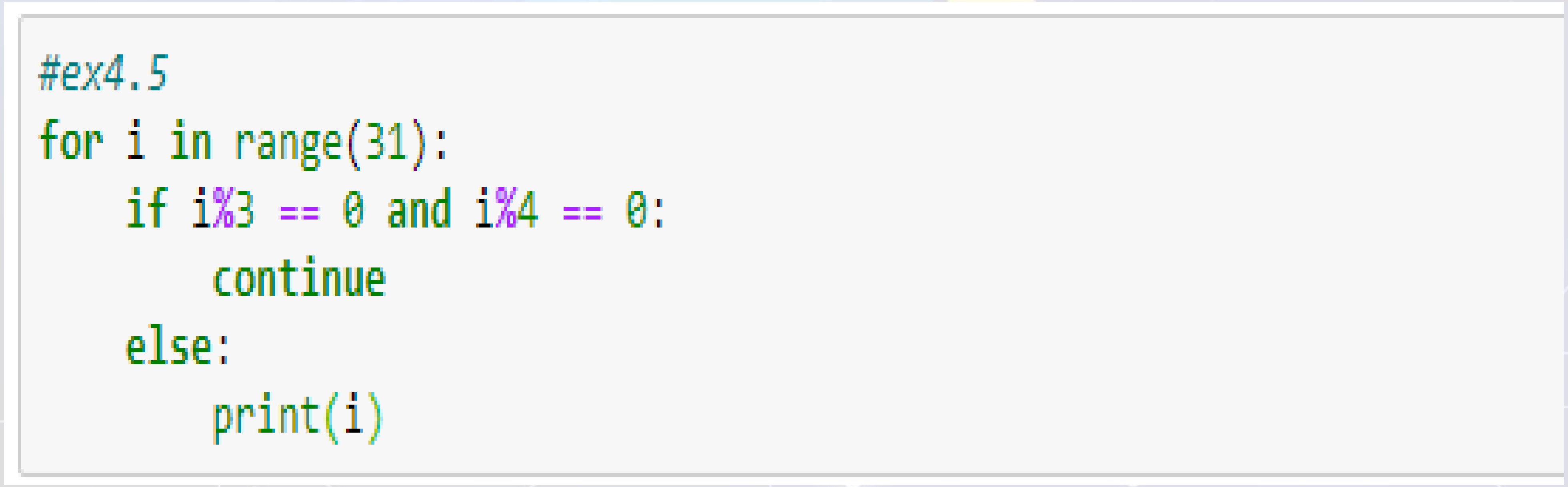
```
user_string = "This is a string."
for i in range(len(user_string)):
  if user_string[i] == 'a':
    print("Found at index:", i)
    break
```

The continue statement

- A **continue** statement allows for skipping the execution of the remainder of the loop without exiting the loop entirely.
- A continue statement can be used in a for or a while loop.
- After the continue statement's execution, the loop expression will be evaluated again and the loop will continue from the loop's expression.
- A continue statement facilitates the loop's control and readability.

Example 4.5: Skipping numbers in a sequence

- Write a program that prints positive integers less than 30, except numbers that are both divisible by 3 and 4.



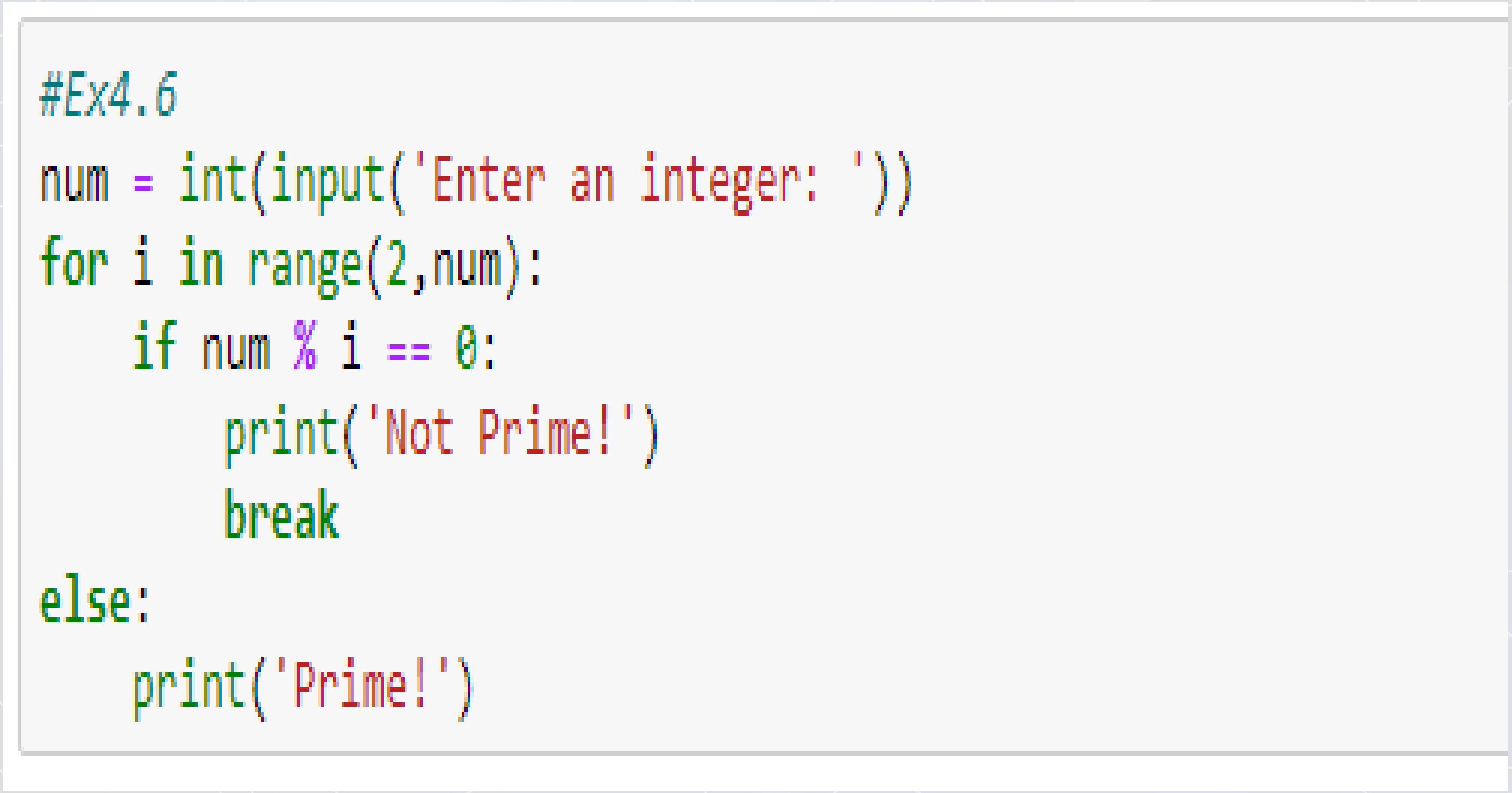
```
#ex4.5
for i in range(31):
    if i%3 == 0 and i%4 == 0:
        continue
    else:
        print(i)
```

The loop else statement

- A **loop else** statement runs after the loop's execution is completed without being interrupted by a break statement.
- A loop else is used to identify if the loop is terminated normally or the execution is interrupted by a break statement.

Example 4.6: Prime numbers

- A number is prime if it is not divisible by any number except 1 and itself.
- In determining if a number is prime, one needs to iterate through all integers from 2 to the given number, and if any of numbers is able to divide the given number without a remainder, then such ceases to be a prime number and the loop is terminated.
- If the loop runs successfully without any interruption by a break statement, then the number is prime.
- Write a program that will ask a user for a number and then state whether or not the number is prime.



```
#Ex4.6
num = int(input('Enter an integer: '))
for i in range(2,num):
    if num % i == 0:
        print('Not Prime!')
        break
else:
    print('Prime!')
```

Questions to attempt from reference text

Concept in Practice Questions

Pages 122 - 139