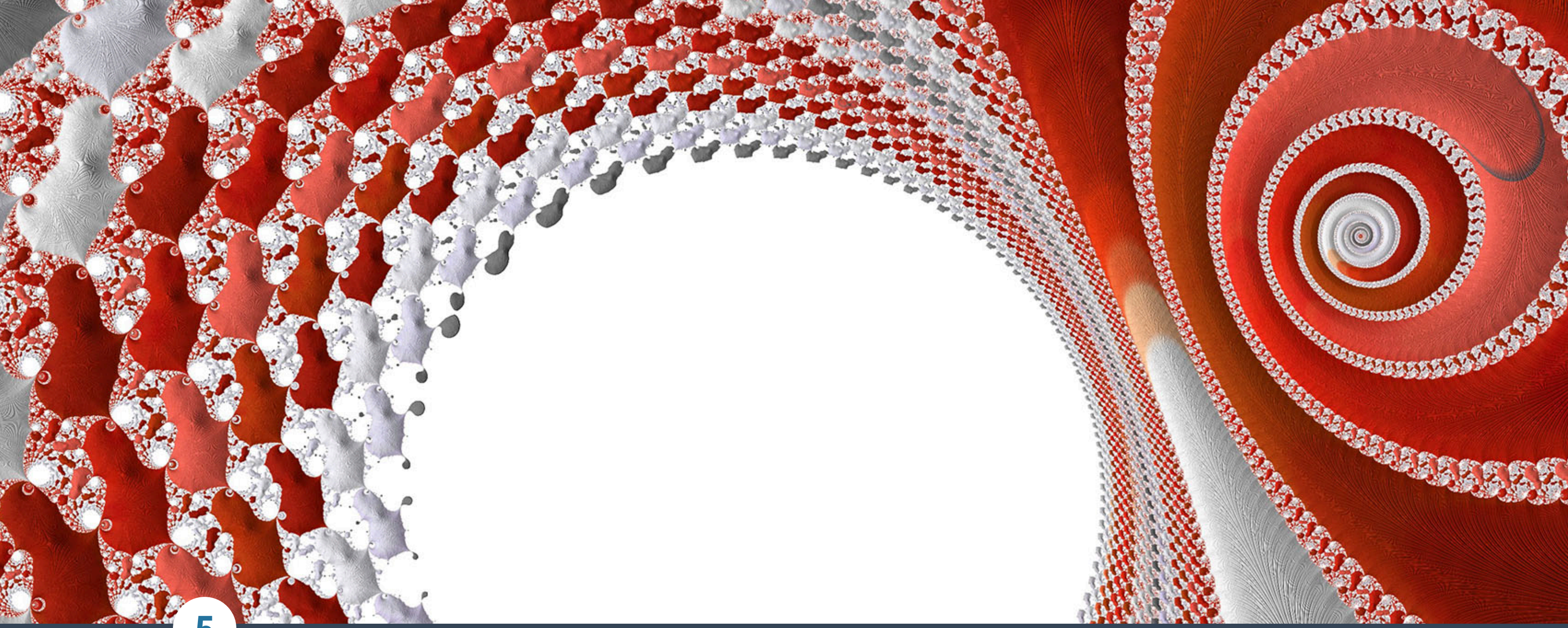

5 Loops

Figure 5.1 credit: modification of work "Quantum Computing", by Kevin Dooley/Flickr, CC BY 2.0

Chapter Outline

Introduction

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. Ex: The software on a phone repeatedly checks to see if the phone is idle. Once the time set by a user is reached, the phone is locked. Loops can also be used for iterating over lists like student names in a roster, and printing the names one at a time.

In this chapter, two types of loops, `for` loop and `while` loop, are introduced. This chapter also introduces `break` and `continue` statements for controlling a loop's execution.

5.1 While loop

Learning objectives

By the end of this section you should be able to

- Explain the loop construct in Python.
- Use a `while` loop to implement repeating tasks.

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.

CHECKPOINT

While loop

[Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-1-while-loop)](https://openstax.org/books/introduction-python-programming/pages/5-1-while-loop)

CONCEPTS IN PRACTICE

While loop example

Fibonacci is a series of numbers in which each number is the sum of the two preceding numbers. The Fibonacci sequence starts with two ones: 1, 1, 2, 3, ... Consider the following code that prints all Fibonacci numbers less than 20, and answer the following questions.

```
    # Initializing the first two Fibonacci numbers
    f = 1
    g = 1
    print (f, end = ' ')

    # Running the loop while the last Fibonacci number is less than 20
    while g < 20:
      print(g, end = ' ')
      # Calculating the next Fibonacci number and updating the last two sequence
numbers
      temp = f
      f = g
      g = temp + g
```

1. How many times does the loop execute?
   a. 5
   b. 6
   c. 7

2. What is the variable g's value when the while loop condition evaluates to `False`?
   a. `13`
   b. `20`
   c. `21`

3. What are the printed values in the output?
   a. `1 1 2 3 5 8 13`
   b. `1 2 3 5 8 13`
   c. `1 1 2 3 5 8 13 21`

Counting with a 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. Ex: 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. 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 5.1

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
```

CHECKPOINT

Counting with while loop

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-1-while-loop)

CONCEPTS IN PRACTICE

while loop counting examples

Given the code, answer the following questions.

```
n = 4
while n > 0:
  print(n)
  n = n - 1

print("value of n after the loop is", n)
```

4. How many times does the loop execute?
   a. 3
   b. 4
   c. 5

5. Which line is printed as the last line of output?
   a. `value of n after the loop is -1.`
   b. `value of n after the loop is 0.`
   c. `value of n after the loop is 1.`

6. What happens if the code is changed as follows?

```
n = 4
while n > 0:
  print(n)
  # Modified line
  n = n + 1

print("value of n after the loop is", n)
```

   a. The code will not run.
   b. The code will run for one additional iteration.
   c. The code will never terminate.

TRY IT

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.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-1-while-loop)

TRY IT

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`.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-1-while-loop)

5.2 For loop

Learning objectives

By the end of this section you should be able to

- Explain the `for` loop construct.
- Use a `for` loop to implement repeating tasks.

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. Ex: Iterating over a class roster and printing students' names.

CHECKPOINT

For loop example for iterating over a container object

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-2-for-loop)

CONCEPTS IN PRACTICE

For loop over a string container

A string variable can be considered a container of multiple characters, and hence can be iterated on. Given the following code, answer the questions.

```
str_var = "A string"

count = 0
for c in str_var:
  count += 1

print(count)
```

1. What is the program's output?
   a. `7`
   b. `8`
   c. `9`

2. What's the code's output if the line `count += 1` is replaced with `count *= 2`?
   a. `0`
   b. `16`
   c. 2^8

3. What is printed if the code is changed as follows?

```
str_var = "A string"

count = 0
for c in str_var:
  count += 1
  # New line
  print(c, end = '*')

print(count)
```

a. `A string*`
b. `A*s*t*r*i*n*g*`
c. `A* *s*t*r*i*n*g*`

Range() function in for loop

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. Examples are provided in the table below.

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`

Table 5.1 Using the range() function.

Range function	Description	Example	Output
		`range(3, -2, -1)`	`3, 2, 1, 0, -1`
		`range(10, 0, -4)`	`10, 6, 2`

Table 5.1 Using the range() function.

EXAMPLE 5.2

Two programs printing all integer multiples of 5 less than 50 (Notice the compactness of the `for` construction compared to the `while`)

```
# For loop condition using
# range() function to print
# all multiples of 5 less than 50
for i in range(0, 50, 5):
  print(i)
```
	```
	# While loop implementation of printing
	# multiples of 5 less than 50

	# Initialization
	i = 0
	# Limiting the range to be less than 50
	while i < 50:
	    print(i)
	    i+=5
	```

Table 5.2

CONCEPTS IN PRACTICE

For loop using a range() function

4. What are the arguments to the `range()` function for the increasing sequence of every 3rd integer from 10 to 22 (inclusive of both ends)?
   a. `range(10, 23, 3)`
   b. `range(10, 22, 3)`
   c. `range(22, 10, -3)`

5. What are the arguments to the `range()` function for the decreasing sequence of every integer from 5 to 1 (inclusive of both ends)?
   a. `range(5, 1, 1)`
   b. `range(5, 1, -1)`
   c. `range(5, 0, -1)`

6. What is the sequence generated from `range(-1, -2, -1)`?
   a. `1`
   b. `-1, -2`
   c. `-2`

7. What is the output of the `range(1, 2, -1)`?
   a. `1`
   b. `1, 2`
   c. empty sequence

8. What is the output of `range(5, 2)`?
   a. `0, 2, 4`
   b. `2, 3, 4`
   c. empty sequence

TRY IT

Counting spaces

Write a program using a `for` loop that takes in a string as input and counts the number of spaces in the provided string. The program must print the number of spaces counted. Ex: If the input is `"Hi everyone"`, the program outputs `1`.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-2-for-loop)

TRY IT

Sequences

Write a program that reads two integer values, n1 and n2, with `n1 < n2`, and performs the following tasks:

1. Prints all even numbers between the two provided numbers (inclusive of both), in ascending order.
2. Prints all odd numbers between the two provided numbers (exclusive of both), in descending order.

```
Input: 2 8


prints
2 4 6 8
7 5 3
```

Note: the program should return an error message if the second number is smaller than the first.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-2-for-loop)

5.3 Nested loops

Learning objectives

By the end of this section you should be able to

- Implement nested `while` loops.
- Implement nested `for` loops.

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.

CHECKPOINT

Nested while loops

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-3-nested-loops)

EXAMPLE 5.3

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
```

CONCEPTS IN PRACTICE

Nested while loop question set

1. Given the following code, how many times does the print statement execute?

```
i = 1
while i <= 5:
  j = 1
  while i + j <= 5:
    print(i, j)
    j += 1
  i += 1
```

   a. 5
   b. 10
   c. 25

2. What is the output of the following code?

```
i = 1

while i <= 2:
  j = 1
  while j <= 3:
    print('*', end = '')
    j += 1
  print()
  i += 1
```

   a.
   ```
   ******
   ```
   b.
   ```
   ***
   ***
   ```
   c.
   ```
   **
   **
   **
   ```

3. Which program prints the following output?

```
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
```

   a.
   ```
   i = 1
   while i <= 4:
     while j <= 4:
       print(i * j, end = ' ')
       j += 1
     print()
   ```

```
        i += 1
b.  i = 1
    while i <= 4:
      j = 1
      while j <= 4:
        print(i * j, end = ' ')
        j += 1
      print()
      i += 1
c.  i = 1
    while i <= 4:
      j = 1
      while j <= 4:
        print(i * j, end = ' ')
        j += 1
      i += 1
```

Nested for loops

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.

CHECKPOINT

Nested for loops

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-3-nested-loops)

CONCEPTS IN PRACTICE

Nested loop practices

4. Given the following code, how many times does the outer loop execute?

```
for i in range(3):
  for j in range(4):
    print(i, ' ', j)
```

a. 3
b. 4
c. 12

5. Given the following code, how many times does the inner loop execute?

```
for i in range(3):
  for j in range(4):
    print(i, ' ', j)
```

a. 4
b. 12
c. 20

6. Which program prints the following output?

```
0 1 2 3
0 2 4 6
0 3 6 9
```

a.
```
for i in range(4):
  for j in range(4):
    print(i * j, end = ' ')
  print()
```
b.
```
for i in range(1, 4):
  for j in range(4):
    print(i * j)
```
c.
```
for i in range(1, 4):
  for j in range(4):
    print(i * j, end = ' ')
  print()
```

MIXED LOOPS

The two `for` and `while` loop constructs can also be mixed in a nested loop construct. Ex: Printing even numbers less than a given number in a list. The outer loop can be implemented using a `for` loop iterating over the provided list, and the inner loop iterates over all even numbers less than a given number from the list using a `while` loop.

```
numbers = [12, 5, 3]

i = 0
for n in numbers:
  while i < n:
    print (i, end = " ")
    i += 2
  i = 0
  print()
```

TRY IT

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!
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-3-nested-loops)

TRY IT

Printing two triangles

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

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

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-3-nested-loops)

5.4 Break and continue

Learning objectives

By the end of this section you should be able to

- Analyze a loop's execution with `break` and `continue` statements.
- Use `break` and `continue` control statements in `while` and `for` loops.

Break

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.

Ex: 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
```

CHECKPOINT

Break statement in a while loop

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-4-break-and-continue)

INFINITE LOOP

A break statement is an essential part of a loop that does not have a termination condition. A loop without a termination condition is known as an **infinite loop**. Ex: An infinite loop that counts up starting from 1 and prints the counter's value while the counter's value is less than 10. A break condition is triggered when the counter's value is equal to 10, and hence the program execution exits.

```
counter = 1
while True:
  if counter >= 10:
    break
  print(counter)
  counter += 1
```

CONCEPTS IN PRACTICE

Using a break statement

1. What is the following code's output?

```
string_val = "Hello World"
for c in string_val:
  if c == " ":
    break
  print(c)
```

a. `Hello`
b. `Hello World`
c.
```
H
e
l
l
o
```

2. Given the following code, how many times does the print statement get executed?

```
i = 1
while True:
  if i%3 == 0 and i%5 == 0:
    print(i)
    break
  i += 1
```

a. 0
b. 1
c. 15

3. What is the final value of i?

```
i = 1
count = 0
while True:
  if i%2 == 0 or i%3 == 0:
    count += 1
  if count >= 5:
    print(i)
    break
  i += 1
```

a. `5`
b. `8`
c. `30`

Continue

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.

CHECKPOINT

Continue statement in a while loop

[Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-4-break-and-continue)](https://openstax.org/books/introduction-python-programming/pages/5-4-break-and-continue)

CONCEPTS IN PRACTICE

Using a continue statement

4. Given the following code, how many times does the print statement get executed?

```
i = 10
while i >= 0:
  i -= 1
  if i%3 != 0:
    continue
  print(i)
```

a. 3
b. 4
c. 11

5. What is the following code's output?

```
for c in "hi Ali":
  if c == " ":
    continue
  print(c)
```

a.
```
h
i

A
l
i
```
b.
```
h
i
A
l
i
```
c.
```
hi
Ali
```

TRY IT

Using break control statement in a loop

Write a program that reads a string value and prints `"Found"` if the string contains a space character. Else, prints `"Not found"`.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-4-break-and-continue)

TRY IT

Using a continue statement in a loop

Complete the following code so that the program calculates the sum of all the numbers in list `my_list` that are greater than or equal to 5.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-4-break-and-continue)

5.5 Loop else

Learning objectives

By the end of this section you should be able to

- Use loop `else` statement to identify when the loop execution is interrupted using a `break` statement.
- Implement a loop `else` statement with a `for` or a `while` loop.

Loop else

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.

Ex: A `for` loop that iterates over a list of numbers to find if the value `10` is in the list. In each iteration, if `10` is observed, the statement `"Found 10!"` is printed and the execution can terminate using a break statement. If `10` is not in the list, the loop terminates when all integers in the list are evaluated, and hence the `else` statement will run and print `"10 is not in the list."` Alternatively, a Boolean variable can be used to track whether number `10` is found after loop's execution terminates.

EXAMPLE 5.4

Finding the number 10 in a list

In the code examples below, the code on the left prints `"Found 10!"` if the variable `i`'s value is `10`. If the value `10` is not in the list, the code prints `"10 is not in the list."`. The code on the right uses the `seen` Boolean variable to track if the value `10` is in the list. After loop's execution, if `seen`'s value is still false, the code prints "10 is not in the list.".

```
numbers = [2, 5, 7, 11, 12]
for i in numbers:
  if i == 10:
    print("Found 10!")
    break
else:
  print("10 is not in the list.")
```
	```
	numbers = [2, 5, 7, 11, 12]
	seen = False
	for i in numbers:
	  if i == 10:
	    print("Found 10!")
	    seen = True
	if seen == False:
	  print("10 is not in the list.")
	```

Table 5.3

CHECKPOINT

Loop else template

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-5-loop-else)

CONCEPTS IN PRACTICE

Loop else practices

1. What is the output?

```
n = 16
exp = 0
i = n
while i > 1:
  if n%2 == 0:
    i = i/2
    exp += 1
  else:
    break
else:
  print(n,"is 2 to the", exp)
```

   a. `16 is 2 to the 3`
   b. `16 is 2 to the 4`
   c. no output

2. What is the output?

```
n = 7
exp = 0
i = n
while i > 1:
  if n%2 == 0:
    i = i//2
    exp += 1
  else:
    break
else:
  print(n,"is 2 to the", exp)
```

a. no output
b. `7 is 2 to the 3`
c. `7 is 2 to the 2`

3. What is the output?

```
numbers = [1, 2, 2, 6]
for i in numbers:
  if i >= 5:
    print("Not all numbers are less than 5.")
    break
  else:
    print(i)
    continue
else:
  print("all numbers are less than 5.")
```

a.
```
1
2
2
6
Not all numbers are less than 5.
```
b.
```
1
2
2
Not all numbers are less than 5.
```
c.
```
1
2
2
6
all numbers are less than 5.
```

TRY IT

Sum of values less than 10

Write a program that, given a list, calculates the sum of all integer values less than 10. If a value greater than or equal to 10 is in the list, no output should be printed. Test the code for different values in the list.

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-5-loop-else)

5.6 Chapter summary

Highlights from this chapter include:

- A `while` loop runs a set of statements, known as the loop body, while a given condition, known as the loop expression, is true.
- A `for` loop can be used to iterate over elements of a container object.
- A `range()` function generates a sequence of integers between the two numbers given a step size.
- A nested loop has one or more loops within the body of another loop.
- 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 `continue` statement allows for skipping the execution of the remainder of the loop without exiting the loop entirely.
- A loop `else` statement runs after the loop's execution is completed without being interrupted by a `break` statement.

At this point, you should be able to write programs with loop constructs. The programming practice below ties together most topics presented in the chapter.

Function	Description
`range(end)`	Generates a sequence beginning at 0 until end with step size of 1.
`range(start, end)`	Generates a sequence beginning at `start` until end with step size of 1.
`range(start, end, s)`	Generates a sequence beginning at `start` until end with the step size of `s`.
Loop constructs	**Description**

Table 5.4 Chapter 5 reference.

Function	Description

`while` loop	`# initialization`
	`while expression:`
	`  # loop body`
	
	`# statements after the loop`

`for` loop	`# initialization`
	`for loop_variable in container:`
	`  # loop body`
	
	`# statements after the loop`

Nested `while` loop	`while outer_loop_expression:`
	`  # outer loop body (1)`
	`  while inner_loop_expression:`
	`    # inner loop body`
	`  # outer loop body (2)`
	
	`# statements after the loop`

`break` statement	`# initialization`
	`while loop_expression:`
	`  # loop body`
	`  if break_condition:`
	`    break`
	`  # remaining body of loop`
	
	`# statements after the loop`

Table 5.4 Chapter 5 reference.

Function	Description

`continue` statement	
	```
	# initialization
	while loop_expression:
	  # loop body
	  if continue_condition:
	    continue
	  # remaining body of loop

	# statements after the loop
	```

Loop `else` statement	
	```
	# initialization
	for loop_expression:
	  # loop body
	  if break_condition:
	    break
	  # remaining body of loop
	else:
	  # loop else statement

	# statements after the loop
	```

Table 5.4 Chapter 5 reference.

TRY IT

Prime numbers

Write a program that takes in a positive integer number (`N`) and prints out the first `N` prime numbers on separate lines.

Note: A prime number is a number that is not divisible by any positive number larger than 1. To check whether a `number` is prime, the condition of `number % i != 0` can be checked for `i` greater than `1` and less than `number`.

Ex: if `N = 6`, the output is:

```
2
3
5
7
```

```
11
13
```

Access multimedia content (https://openstax.org/books/introduction-python-programming/pages/5-6-chapter-summary)