Step 6 Test your function.

Supply a program file for testing this function only:

```
from random import randint

def main() :
   for i in range(10) :
      print(insertAtRandom("arxcsw", "8")

def insertAtRandom(string, toInsert) :
   n = len(string) :
   r = randint(0, n)
   result = ""

   for i in range(r) :
      result = result + string[i]
   result = result + toInsert
   for i in range(r, n) :
      result = result + string[i]

   return result

main()
```

When you run this program, you might get an output such as

```
arxcsw8
ar8xcsw
arxc8sw
a8rxcsw
arxcsw8
ar8xcsw
arxcsw8
a8rxcsw
8arxcsw
8arxcsw
```

The output shows that the second string is being inserted at an arbitrary position, including the beginning and end of the first string.

See `password.py` in your source code for the complete program.

5.5 Functions Without Return Values

Some functions may not return a value, but they can produce output.

Sometimes, you need to carry out a sequence of instructions that does not yield a value. If that instruction sequence occurs multiple times, you will want to package it into a function.

Here is a typical example: Your task is to print a string in a box, like this:

```
-------
!Hello!
-------
```

Some functions are called because they produce output, even though they don't return a value.

However, different strings can be substituted for `Hello`. A function for this task can be defined as follows:

```
def boxString(contents) :
```

Now you develop the body of the function in the usual way, by formulating a general algorithm for solving the task.

Print a line that contains the - character n + 2 times, where n is the length of the string.
Print a line containing the contents, surrounded with a ! to the left and right.
Print another line containing the - character n + 2 times.

Here is the function implementation:

```
## Prints a string in a box.
#  @param contents the string to enclose in a box
#
def boxString(contents) :
   n = len(contents) :
   print("-" * (n + 2))
   print("!" + contents + "!")
   print("-" * (n + 2))
```

Note that this function doesn't compute any value. It performs some actions and then returns to the caller. Actually, the function returns a special value, called `None`, but there is nothing that you can do with that value.

Because there is no useful return value, don't use `boxString` in an expression. You can call

```
boxString("Hello")
```

but don't call

```
result = boxString("Hello")   # No—boxString doesn't return a useful result.
```

If you want to return from a function that does not compute a value before reaching the end, you use a `return` statement without a value. For example,

```
def boxString(contents) :
   n = len(contents)
   if n == 0 :
      return   # Return immediately
   print("-" * (n + 2))
   print("!" + contents + "!")
   print("-" * (n + 2))
```

16. How do you generate the following printout, using the `boxString` function?

```
-------
!Hello!
-------
-------
!World!
-------
```

17. What is wrong with the following statement?

```
print(boxString("Hello"))
```

18. Implement a function `shout` that prints a line consisting of a string followed by three exclamation marks. For example, `shout("Hello")` should print `Hello!!!`. The function should not return a value.

19. How would you modify the `boxString` function to leave a space around the string that is being boxed, like this:

```
---------
! Hello !
---------
```

Practice It Now you can try these exercises at the end of the chapter: R5.6, P5.25.

5.6 Problem Solving: Reusable Functions

Eliminate replicated code or pseudocode by defining a function.

You have used many Python functions, both built-in and from the standard library. These functions have been provided as a part of the Python platform so that programmers need not recreate them. Of course, the Python library doesn't cover every conceivable need. You will often be able to save yourself time by designing your own functions that can be used for multiple problems.

When you write nearly identical code or pseudocode multiple times, either in the same program or in separate programs, consider introducing a function. Here is a typical example of code replication:

```
hours = int(input("Enter a value between 0 and 23: "))
while hours < 0 or hours > 23 :
   print("Error: value out of range.")
   hours = int(input("Enter a value between 0 and 23: "))

minutes = int(input("Enter a value between 0 and 59: "))
while minutes < 0 or minutes > 59 :
   print("Error: value out of range.")
   minutes = int(input("Enter a value between 0 and 59: "))
```

This program segment reads two variables, making sure that each of them is within a certain range. It is easy to extract the common behavior into a function:

```
## Prompts a user to enter a value up to a given maximum until the user provides
#  a valid input.
#  @param high an integer indicating the largest allowable input
#  @return the integer value provided by the user (between 0 and high, inclusive)
#
def readIntUpTo(high) :
   value = int(input("Enter a value between 0 and " + str(high) + ": "))
   while value < 0 or value > high :
      print("Error: value out of range.")
      value = int(input("Enter a value between 0 and " + str(high) + ": "))

   return value
```

Then use this function twice:

```
hours = readIntUpTo(23)
minutes = readIntUpTo(59)
```

We have now removed the replication of the loop—it only occurs once, inside the function.

Note that the function can be reused in other programs that need to read integer values. However, we should consider the possibility that the smallest value need not always be zero.