Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10

© yenwen/iStockphoto.

Although a thermostat is usually white, you can think of it as a black box. The input is the desired temperature, and the output is a signal to the heater or air conditioner.

SELF CHECK

• **1.** The value passed back from a function is called a(n) _____.

- [ ] argument
- [x] return value
- [ ] input
- [ ] output

One correct, 0 errors, 100%

• **2.** Assuming variable x contains a value of 3.1415926, what value is returned by the function call `round(x, 3)`?

- [ ] 3
- [ ] 3.1
- [ ] 3.14
- [x] 3.142

One correct, 0 errors, 100%

•• **3.** Test your understanding of function arguments with the activity below.

GOOD JOB! ✓

Question	Answer	Explanation
How many arguments does the function call `round(10.2468, 3)` have?	two	Each value in the parentheses is an argument.
What is the first argument of the function call `min(56, 17, -3)`?	56	This is the first value that is passed to the function.
What is the first argument of the function call `min(5 + 5, 3, 17)`?	10	When an argument is an expression, it is first computed before it is passed to the function.
What is the return value of the function call `min(5 + 5, 3, 17)`?	3	This value is passed from the function back to the caller.
How many arguments does a call to the `sqrt` function have?	one	A typical call is `sqrt(4)`. You pass one argument to this function.

5 correct, 0 errors, 100%, 27 seconds

Start over

5.1

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Back to Page

186 / 554

5.2.1 Implementing a Function

We will start with a very simple example: a function to compute the volume of a cube with a given side length.

© studioaraminta/iStockphoto.

The `cubeVolume` function uses a given side length to compute the volume of a cube.

When writing this function, you need to

- Pick a name for the function (`cubeVolume`).
- Define a variable for each argument (`sideLength`). These variables are called the **parameter variables**.

Put all this information together along with the `def` reserved word to form the first line of the function's definition:

```
def cubeVolume(sideLength) :
```

When defining a function, you provide a name for the function and a variable for each argument.

This line is called the **header** of the function. Next, specify the **body** of the function. The body contains the statements that are executed when the function is called.

The volume of a cube of side length s is $s \times s \times s = s^3$. However, for greater clarity, our parameter variable has been called `sideLength`, not s, so we need to compute `sideLength ** 3`.

We will store this value in a variable called `volume`:

```
volume = sideLength ** 3
```

In order to return the result of the function, use the `return` statement:

```
return volume
```

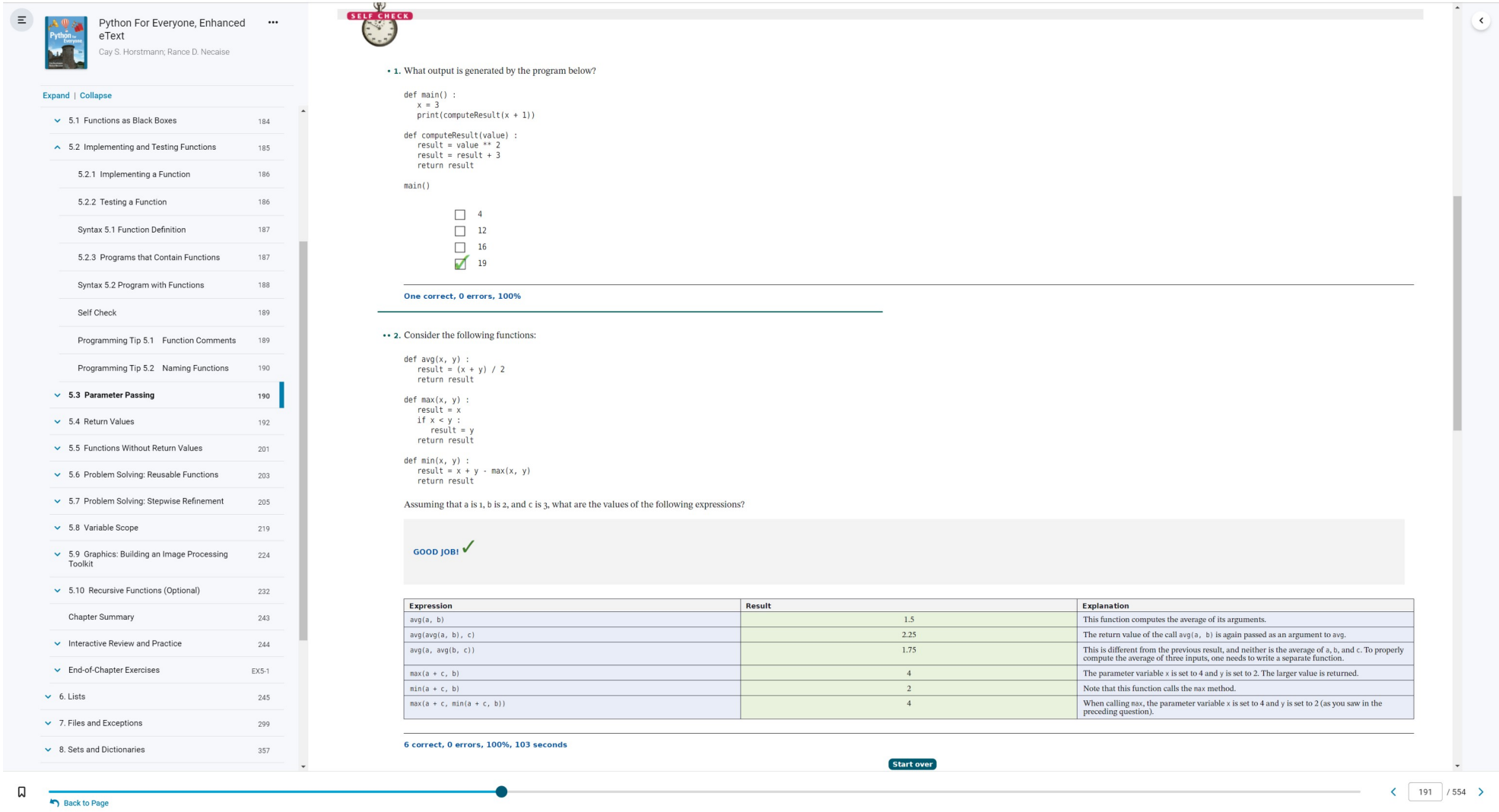
© princessdlaf/iStockphoto.

The `return` statement gives the function's result to the caller.

A function is a compound statement, which requires the statements in the body to be indented to the same level. Here is the complete function:

```
def cubeVolume(sideLength) :
   volume = sideLength ** 3
   return volume
```

5.2



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

SELF CHECK

• **1.** What output is generated by the program below?

```
def main() :
   x = 3
   print(computeResult(x + 1))

def computeResult(value) :
   result = value ** 2
   result = result + 3
   return result

main()
```

- ☐ 4
- ☐ 12
- ☐ 16
- ☑ 19

One correct, 0 errors, 100%

•• **2.** Consider the following functions:

```
def avg(x, y) :
   result = (x + y) / 2
   return result

def max(x, y) :
   result = x
   if x < y :
      result = y
   return result

def min(x, y) :
   result = x + y - max(x, y)
   return result
```

Assuming that a is 1, b is 2, and c is 3, what are the values of the following expressions?

GOOD JOB! ✓

Expression	Result	Explanation
`avg(a, b)`	1.5	This function computes the average of its arguments.
`avg(avg(a, b), c)`	2.25	The return value of the call `avg(a, b)` is again passed as an argument to `avg`.
`avg(a, avg(b, c))`	1.75	This is different from the previous result, and neither is the average of `a`, `b`, and `c`. To properly compute the average of three inputs, one needs to write a separate function.
`max(a + c, b)`	4	The parameter variable `x` is set to 4 and `y` is set to 2. The larger value is returned.
`min(a + c, b)`	2	Note that this function calls the `max` method.
`max(a + c, min(a + c, b))`	4	When calling `max`, the parameter variable `x` is set to 4 and `y` is set to 2 (as you saw in the preceding question).

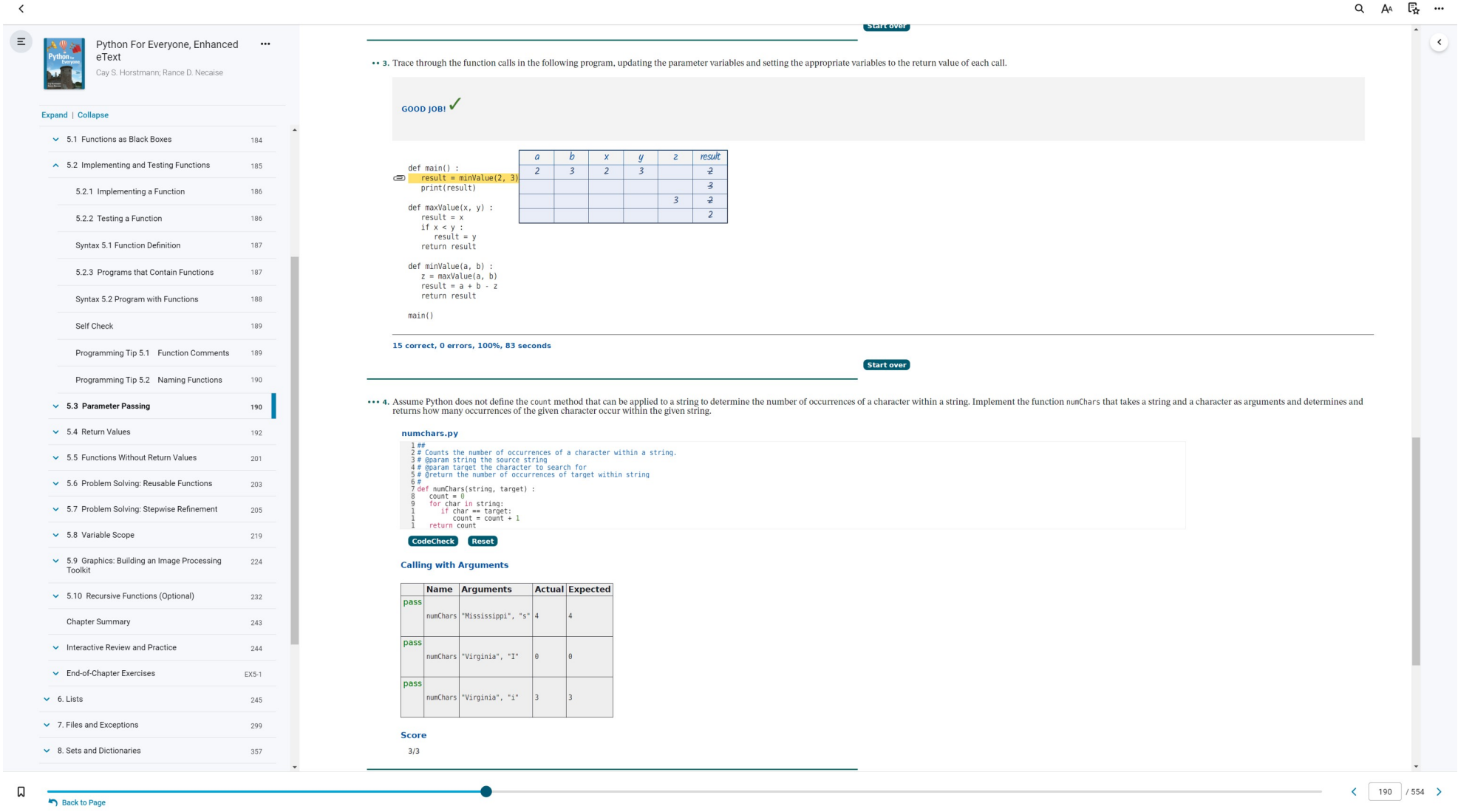
6 correct, 0 errors, 100%, 103 seconds

Start over

Back to Page

191 / 554

5.3 (p.1)



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

Start over

•• 3. Trace through the function calls in the following program, updating the parameter variables and setting the appropriate variables to the return value of each call.

GOOD JOB! ✓

```
def main() :
   result = minValue(2, 3)
   print(result)

def maxValue(x, y) :
   result = x
   if x < y :
      result = y
   return result

def minValue(a, b) :
   z = maxValue(a, b)
   result = a + b - z
   return result

main()
```

a	b	x	y	z	result
2	3	2	3		~~2~~
					~~3~~
				3	~~2~~
					2

15 correct, 0 errors, 100%, 83 seconds

Start over

••• 4. Assume Python does not define the `count` method that can be applied to a string to determine the number of occurrences of a character within a string. Implement the function `numChars` that takes a string and a character as arguments and determines and returns how many occurrences of the given character occur within the given string.

numchars.py

```
##
# Counts the number of occurrences of a character within a string.
# @param string the source string
# @param target the character to search for
# @return the number of occurrences of target within string
#
def numChars(string, target) :
   count = 0
   for char in string:
      if char == target:
         count = count + 1
   return count
```

CodeCheck Reset

Calling with Arguments

	Name	Arguments	Actual	Expected
pass	numChars	"Mississippi", "s"	4	4
pass	numChars	"Virginia", "I"	0	0
pass	numChars	"Virginia", "i"	3	3

Score

3/3

Back to Page

190 / 554

5.3 (p.2)

Score

3/3

•• **5.** Trace through the following walkthrough of calls to the `cubeVolume` function, updating parameter variables and setting the `result` variable to the return value of each call. When you are asked to indicate which variables are "out of scope" (that is, removed when the function exits), click on the variable names in the table.

GOOD JOB! ✓

```
def main() :
   len = 2
   result = cubeVolume(len)
   print(result)
   result = cubeVolume(5 + 5)
   print(result)

def cubeVolume(sideLength) :
   volume = sideLength ** 3
   return volume

main()
```

len	sideLength	volume	return value	result
2	~~2~~	~~8~~	~~8~~	
		~~X~~		
	~~X~~			~~8~~
	~~10~~	~~1000~~	1000	
		X		
	X			1000

16 correct, 0 errors, 100%, 57 seconds

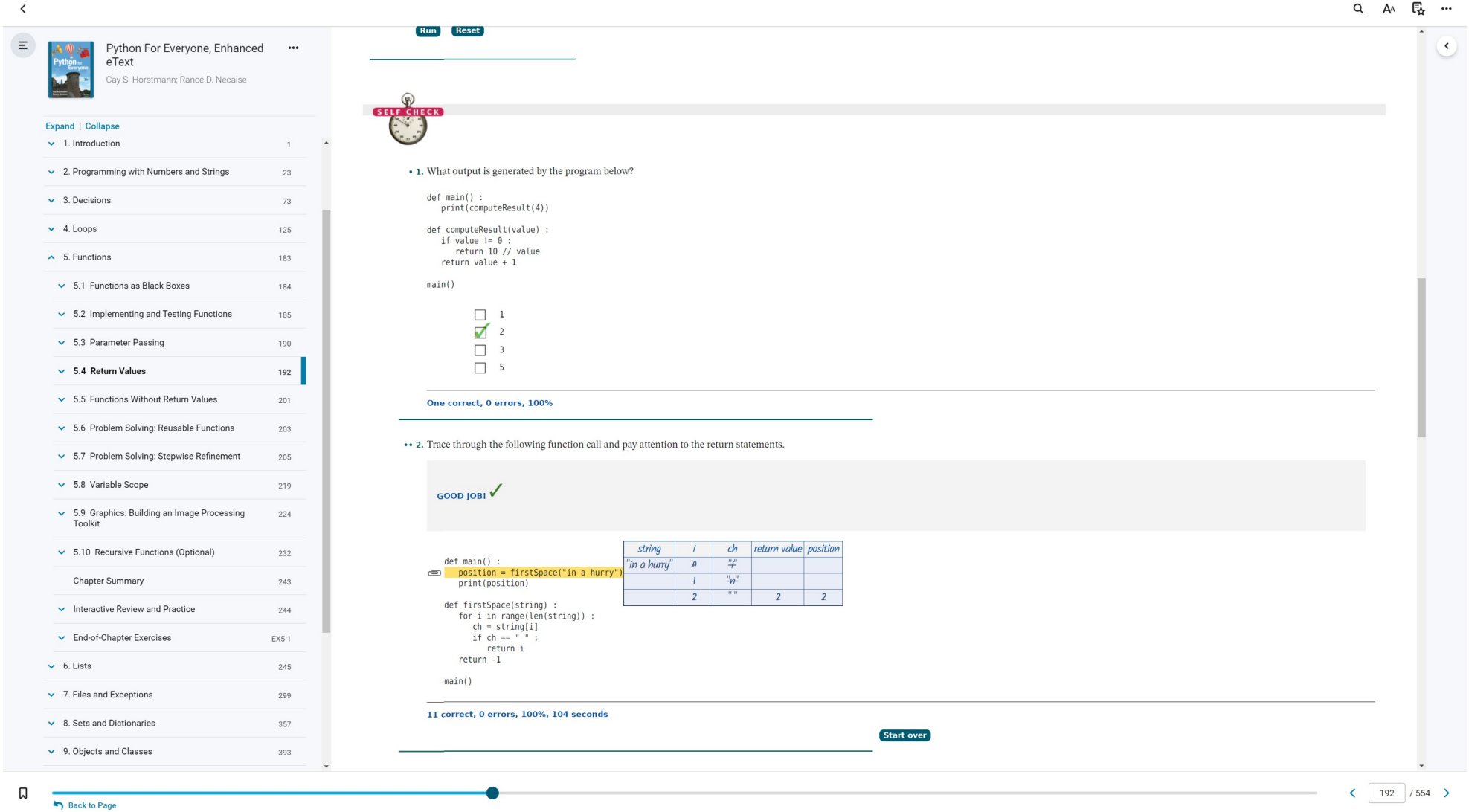
Start over

Back to Page

190 / 554

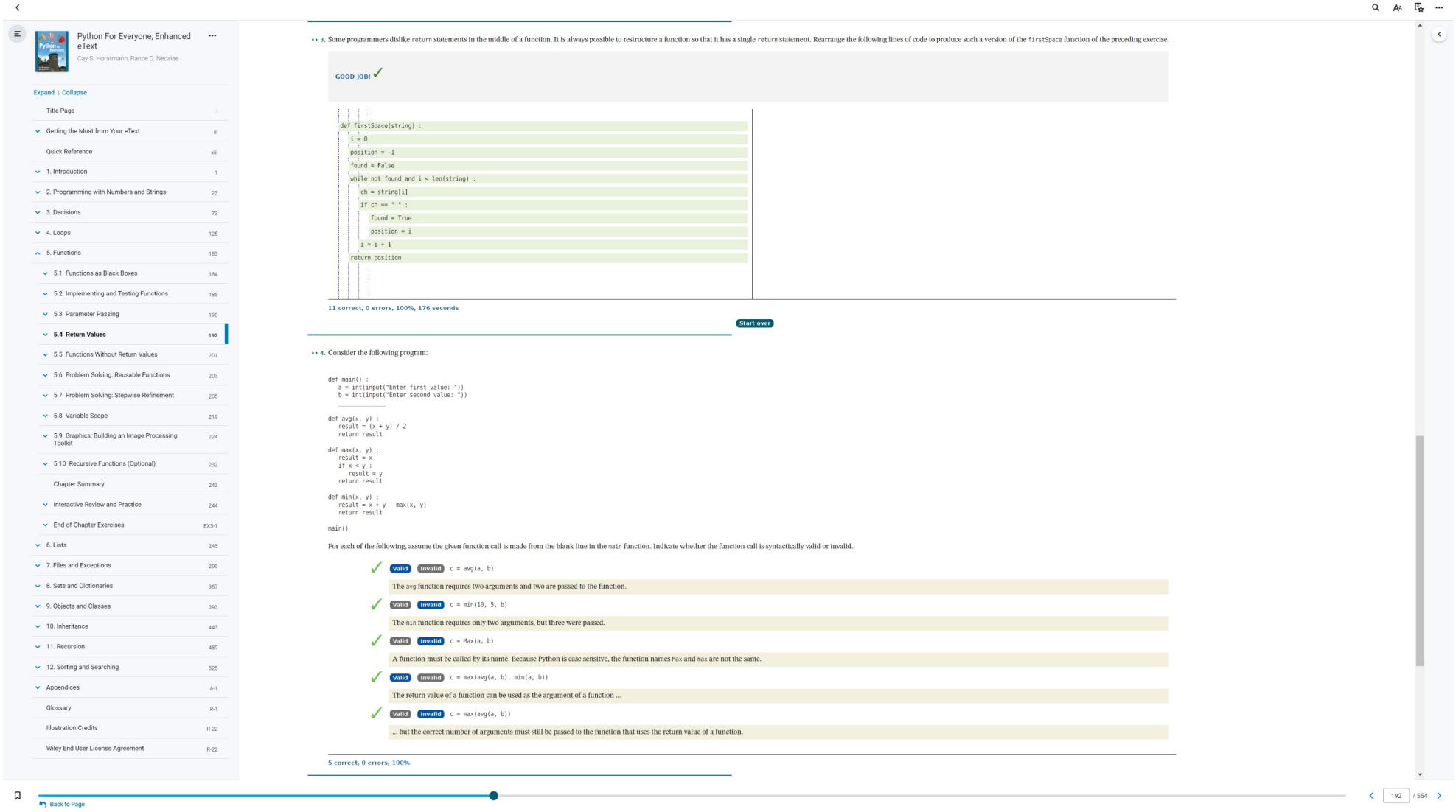
5.3 (p.3)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



5.4 (p.1)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

•• 3. Some programmers dislike `return` statements in the middle of a function. It is always possible to restructure a function so that it has a single `return` statement. Rearrange the following lines of code to produce such a version of the `firstSpace` function of the preceding exercise.

GOOD JOB! ✓

```
def firstSpace(string) :
    i = 0
    position = -1
    found = False
    while not found and i < len(string) :
        ch = string[i]
        if ch == " " :
            found = True
            position = i
        i = i + 1
    return position
```

11 correct, 0 errors, 100%, 176 seconds

Start over

•• 4. Consider the following program:

```
def main() :
   a = int(input("Enter first value: "))
   b = int(input("Enter second value: "))
   ____

def avg(x, y) :
   result = (x + y) / 2
   return result

def max(x, y) :
   result = x
   if x < y :
      result = y
   return result

def min(x, y) :
   result = x + y - max(x, y)
   return result

main()
```

For each of the following, assume the given function call is made from the blank line in the `main` function. Indicate whether the function call is syntactically valid or invalid.

✓ Valid Invalid `c = avg(a, b)`

The `avg` function requires two arguments and two are passed to the function.

✓ Valid Invalid `c = min(10, 5, b)`

The `min` function requires only two arguments, but three were passed.

✓ Valid Invalid `c = Max(a, b)`

A function must be called by its name. Because Python is case sensitve, the function names `Max` and `max` are not the same.

✓ Valid Invalid `c = max(avg(a, b), min(a, b))`

The return value of a function can be used as the argument of a function ...

✓ Valid Invalid `c = max(avg(a, b))`

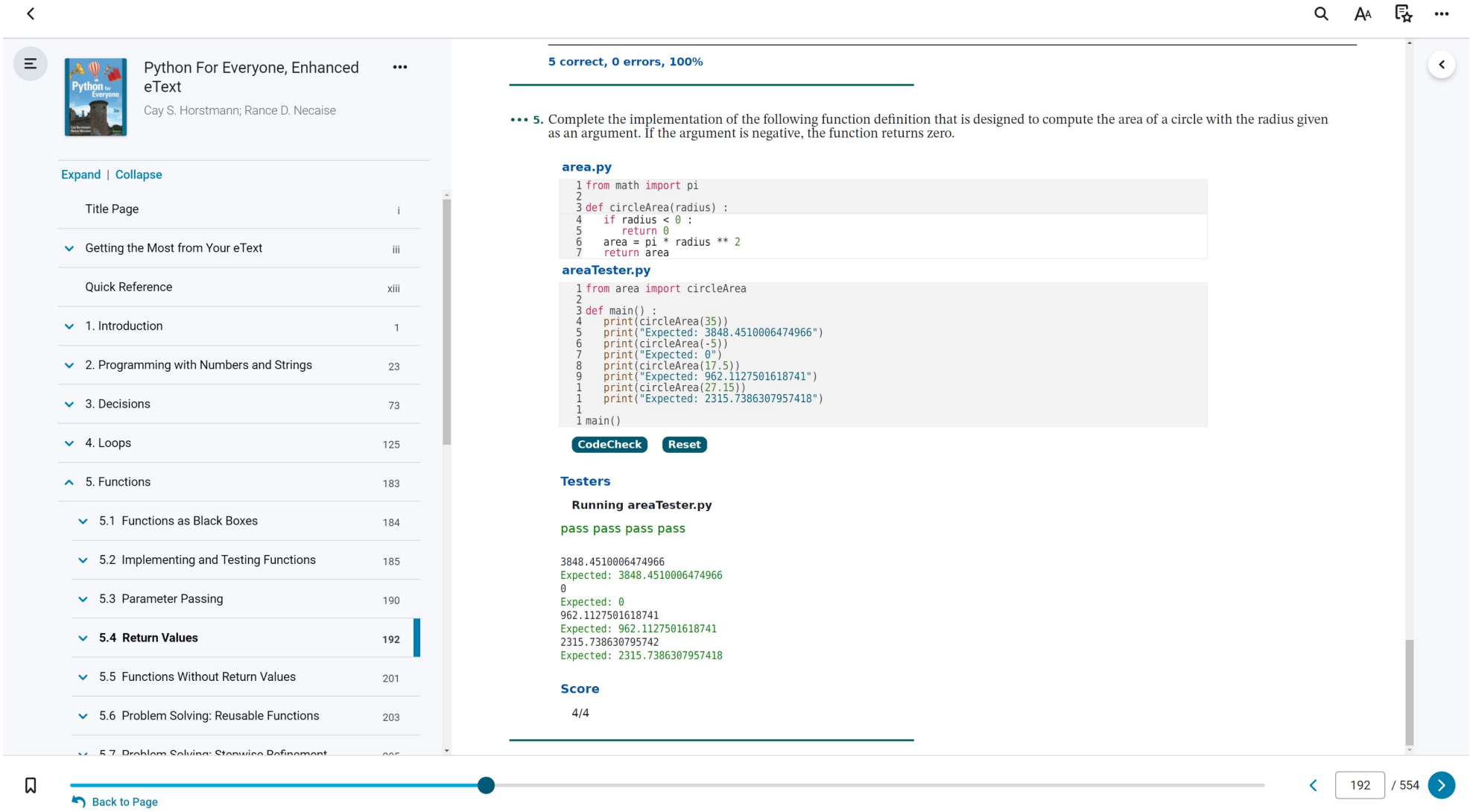
... but the correct number of arguments must still be passed to the function that uses the return value of a function.

5 correct, 0 errors, 100%

Back to Page

192 / 554

5.4 (p.2)



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

5 correct, 0 errors, 100%

••• 5. Complete the implementation of the following function definition that is designed to compute the area of a circle with the radius given as an argument. If the argument is negative, the function returns zero.

area.py

```
from math import pi

def circleArea(radius) :
   if radius < 0 :
      return 0
   area = pi * radius ** 2
   return area
```

areaTester.py

```
from area import circleArea

def main() :
   print(circleArea(35))
   print("Expected: 3848.4510006474966")
   print(circleArea(-5))
   print("Expected: 0")
   print(circleArea(17.5))
   print("Expected: 962.1127501618741")
   print(circleArea(27.15))
   print("Expected: 2315.7386307957418")

main()
```

CodeCheck Reset

Testers

Running areaTester.py

pass pass pass pass

```
3848.4510006474966
Expected: 3848.4510006474966
0
Expected: 0
962.1127501618741
Expected: 962.1127501618741
2315.738630795742
Expected: 2315.7386307957418
```

Score

4/4

Back to Page

192 / 554

5.4 (p.3)

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

• **1.** Which statement correctly calls function `displayLines` below so that `##########` is printed ?

```
def displayLines() :
   for k in range(5) :
      print("##########")
```

- ☐ `print(displayLines())`
- ☐ `displayLines(5)`
- ☑ `displayLines()`
- ☐ `result = displayLines(5)`

One correct, 0 errors, 100%

• **2.** What output is generated by the program below?

```
def main() :
   showStars(3)

def showStars(number) :
   for k in range(number) :
      print("*", end="")

main()
```

- ☐ `*`
- ☑ `***`
- ☐ `3`
- ☐ `333`

One correct, 0 errors, 100%

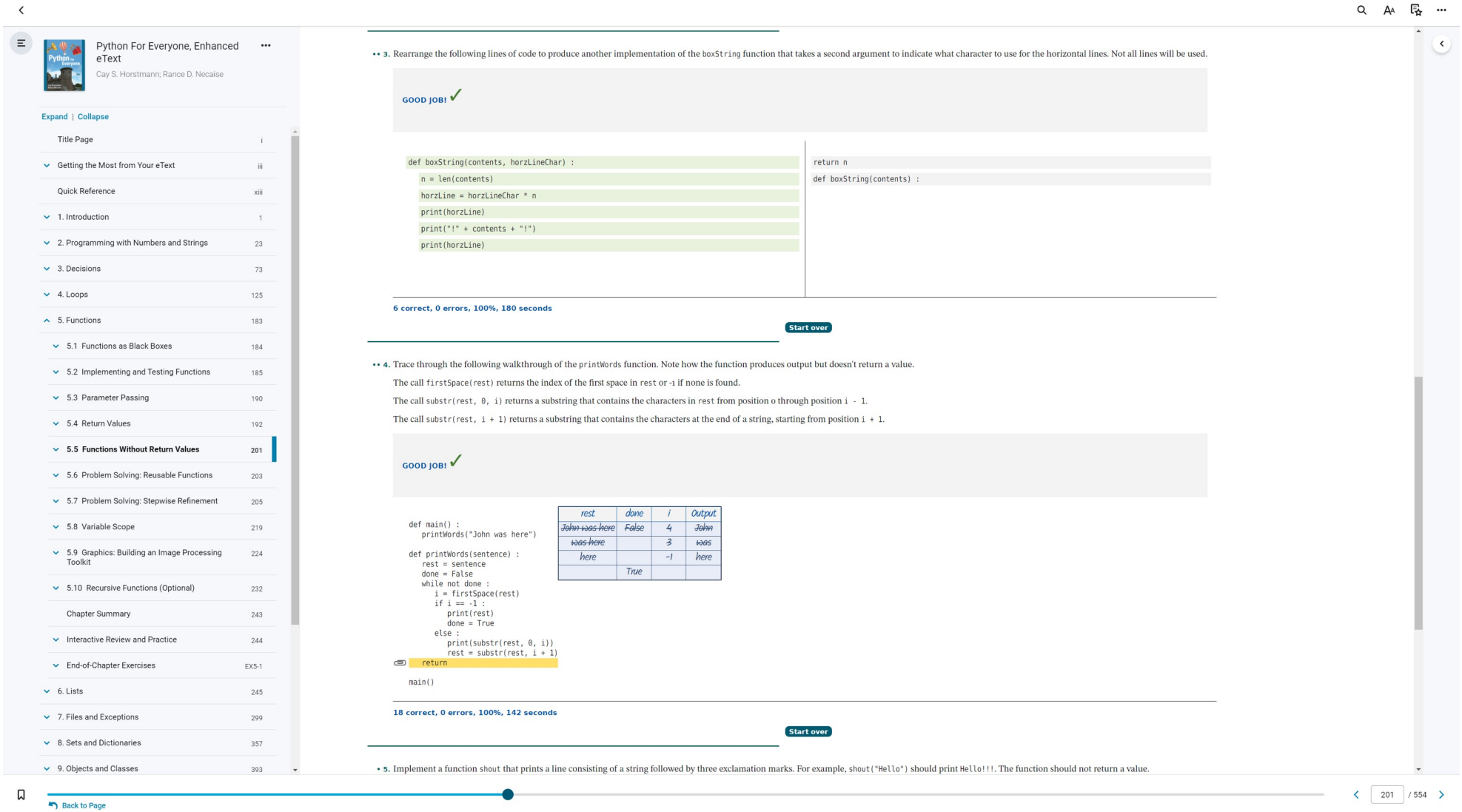
•• **3.** Rearrange the following lines of code to produce another implementation of the `boxString` function that takes a second argument to indicate what character to use for the horizontal lines. Not all lines will be used.

• **3.** How could one improve the user experience of the unit conversion program so that users are not frustrated by incompatible units?

- ☐ Terminate the program when two units are incompatible.
- ☐ Return a result of 0 when two units are incompatible: *30 in = 0 oz*
- ☐ Only provide conversions between compatible units.
- ☑ List compatible units in the *To unit* prompt: *To unit (ft, mi, mm, cm, m, km):*

One correct, 0 errors, 100%

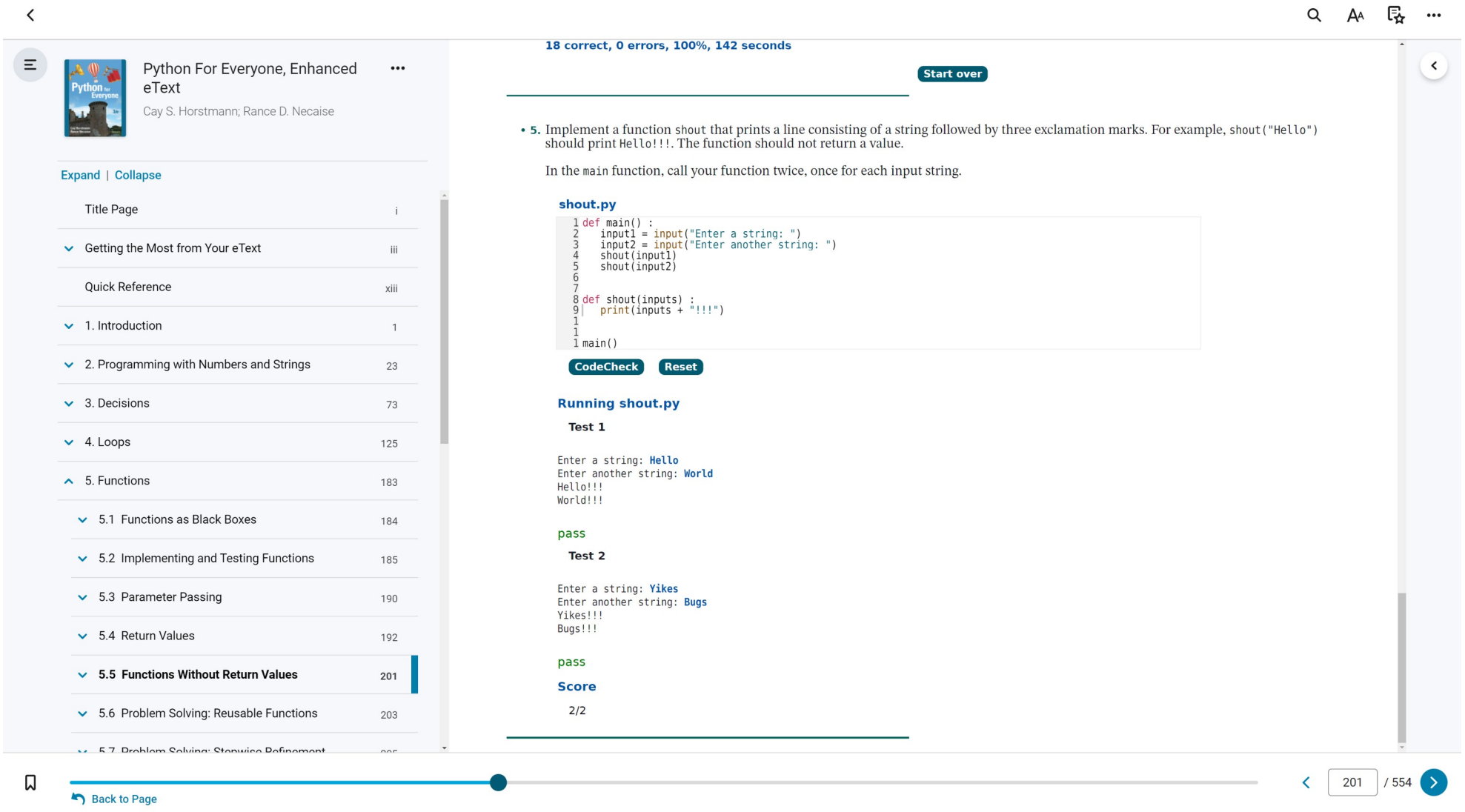
Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



5.5 (p.2)

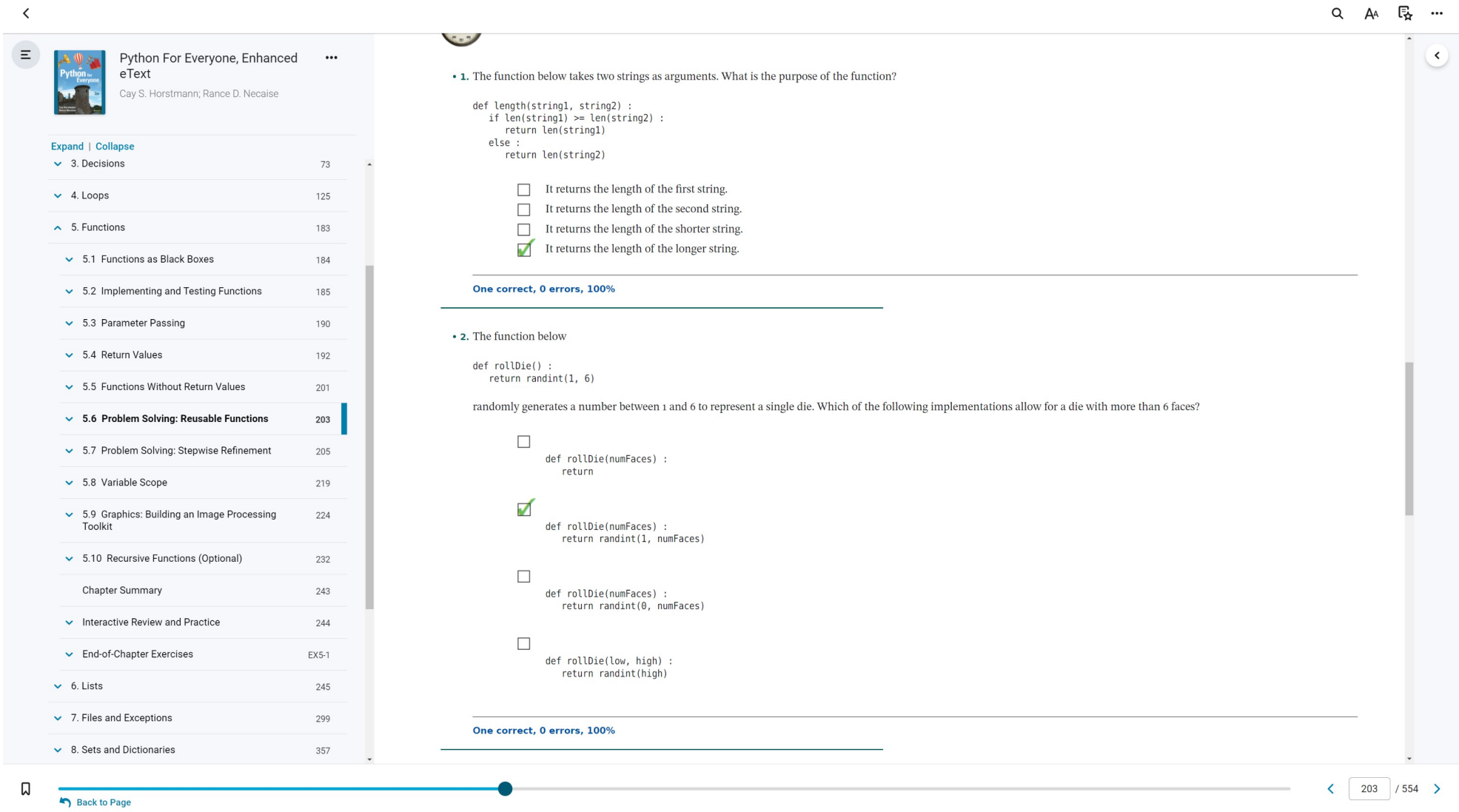
5.5 (p.2) - Functions Without Return Values (3, 4)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



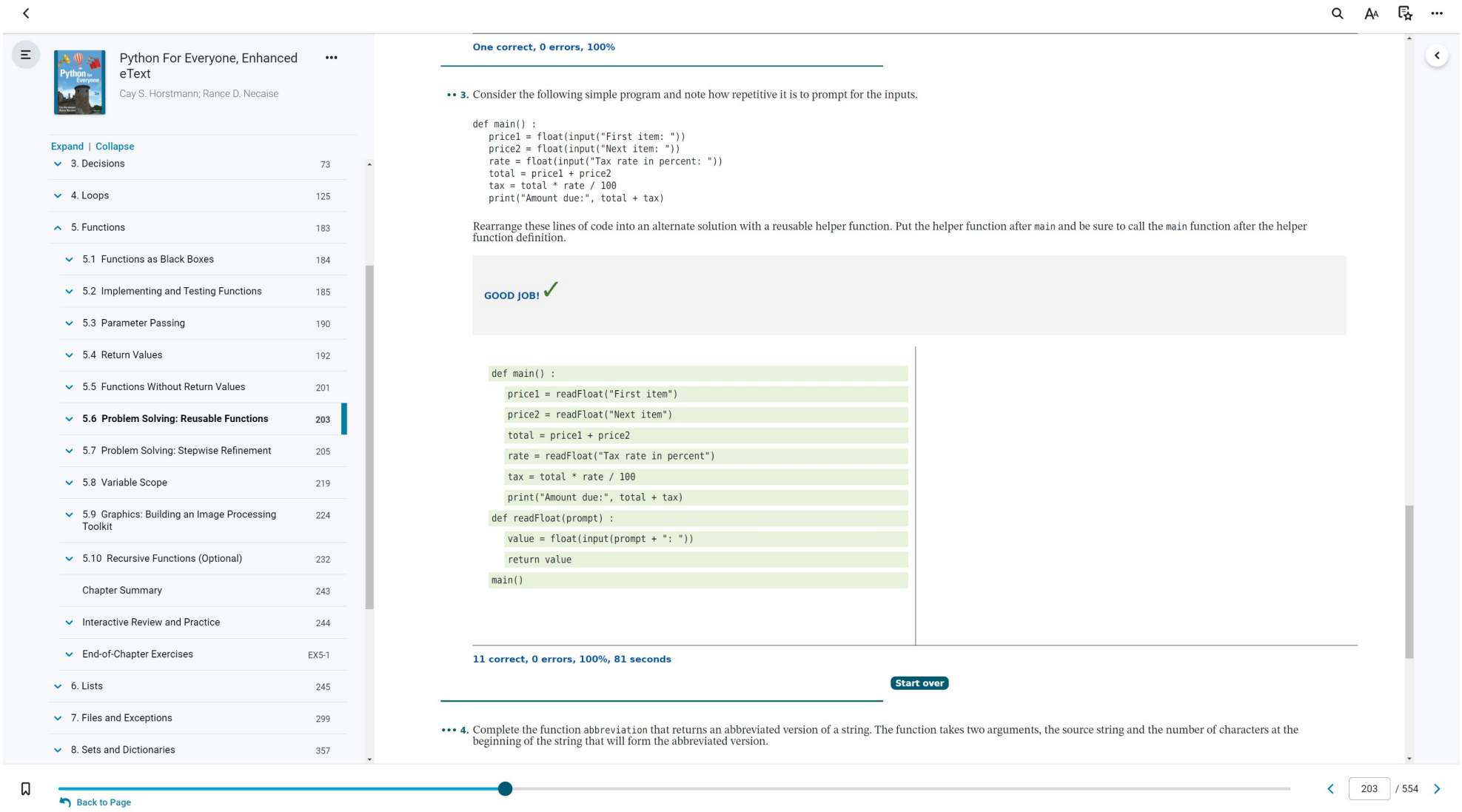
5.5 (p.3)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



5.6 (p.1)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



5.6 (p.2)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

11 correct, 0 errors, 100%, 81 seconds

Start over

••• 4. Complete the function `abbreviation` that returns an abbreviated version of a string. The function takes two arguments, the source string and the number of characters at the beginning of the string that will form the abbreviated version.

abbrv.py

```
##
# Returns the abbreviated version of a string.
# @param string the string from which the abbreviation is constructed
# @param size the maximum number of characters in the abbreviation
# @returns the abbreviated version of the string, which is formed from
#       size number of characters at the front of of the string
#
def abbreviation(string, size) :
   stringAbbreviated = ""
   if size > len(string) :
      size = len(string)
   for char in range(0, size) :
      stringAbbreviated =  stringAbbreviated + string[char]
   return stringAbbreviated
```

CodeCheck Reset

Calling with Arguments

	Name	Arguments	Actual	Expected
pass	abbreviation	"Mississippi", 6	Missis	Missis
pass	abbreviation	"Hello", 6	Hello	Hello
pass	abbreviation	"Hello", 4	Hell	Hell

Score

3/3

Back to Page

203 / 554

5.6 (p.3)

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

• **1.** When implementing a program designed using the process known as *stepwise refinement*, it is common to write temporary functions that are incomplete but allow the testing of other functions. The incomplete functions are known as:

- [x] stubs
- [ ] functions without return values
- [ ] helper functions
- [ ] reusable functions

One correct, 0 errors, 100%

• **2.** What is stepwise refinement?

- [ ] The process of unit testing.
- [ ] The design of pseudocode for black-box functions.
- [x] The process of breaking complex problems down into smaller, manageable steps.
- [ ] The use of a temporary implementation of a function that can be improved later.

One correct, 0 errors, 100%

5.7 (p.1)

•• 3. Trace through the following program. (The `intName` function calls the `digitName`, `teenName`, and `tensName` functions from this section.)

GOOD JOB! ✓

```
def main() :
   print(intName(314))
   print(intName(42))

def intName(number) :
   part = number  # The part that still needs to be converted
   name = ""      # The name of the number

   if part >= 100 :
      name = digitName(part // 100) + " hundred"
      part = part % 100

   if part >= 20 :
      name = name + " " + tensName(part)
      part = part % 10
   elif part >= 10 :
      name = name + " " + teenName(part)
      part = 0

   if part > 0 :
      name = name + " " + digitName(part)

   return name

main()
```

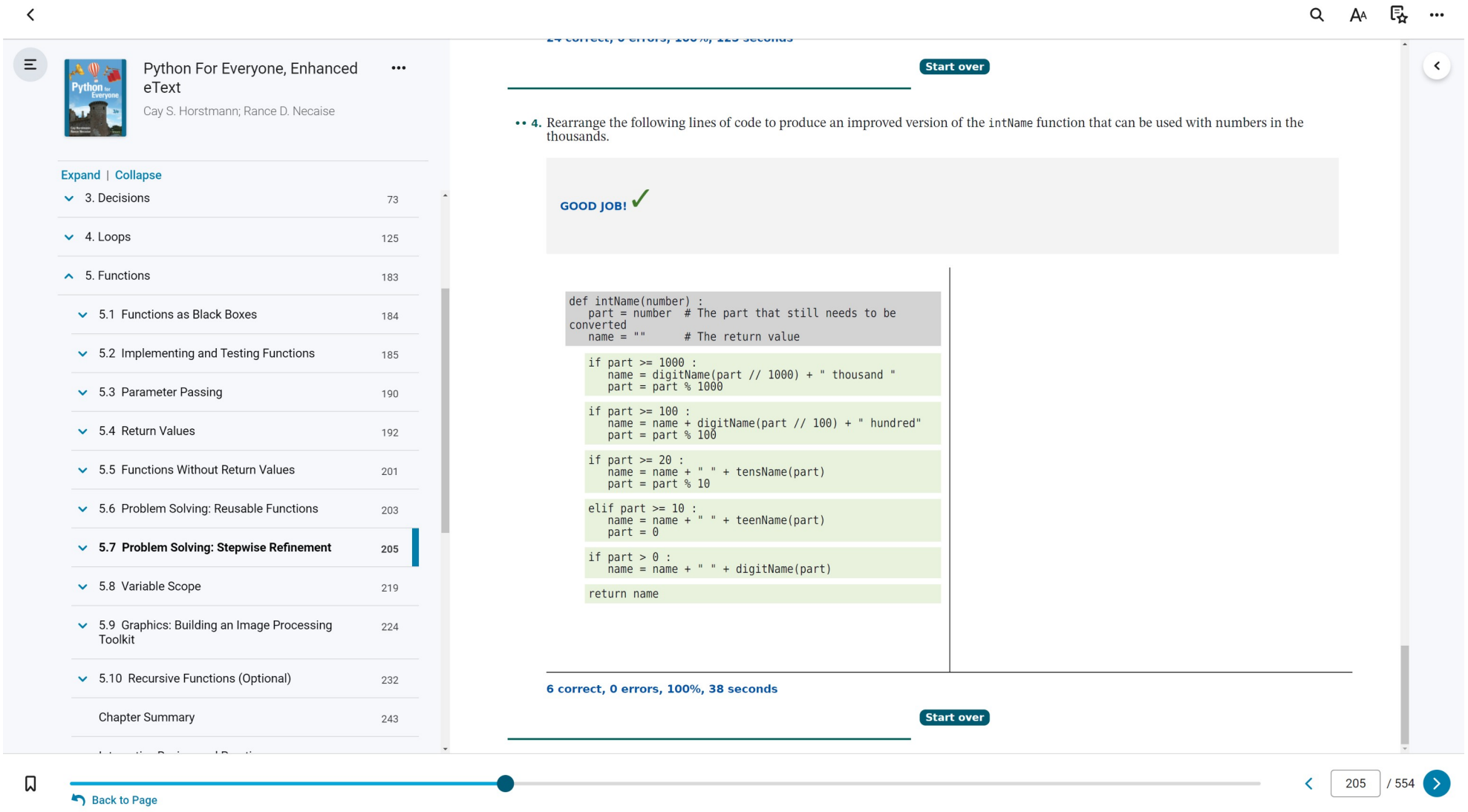
number	part	name	Output
~~314~~	~~314~~	~~(empty)~~	
		~~three hundred~~	
	~~14~~	~~three hundred fourteen~~	
	~~0~~		~~three hundred fourteen~~
~~42~~	~~42~~	~~(empty)~~	
		~~forty~~	
	2	forty two	forty two

24 correct, 0 errors, 100%, 125 seconds

Start over

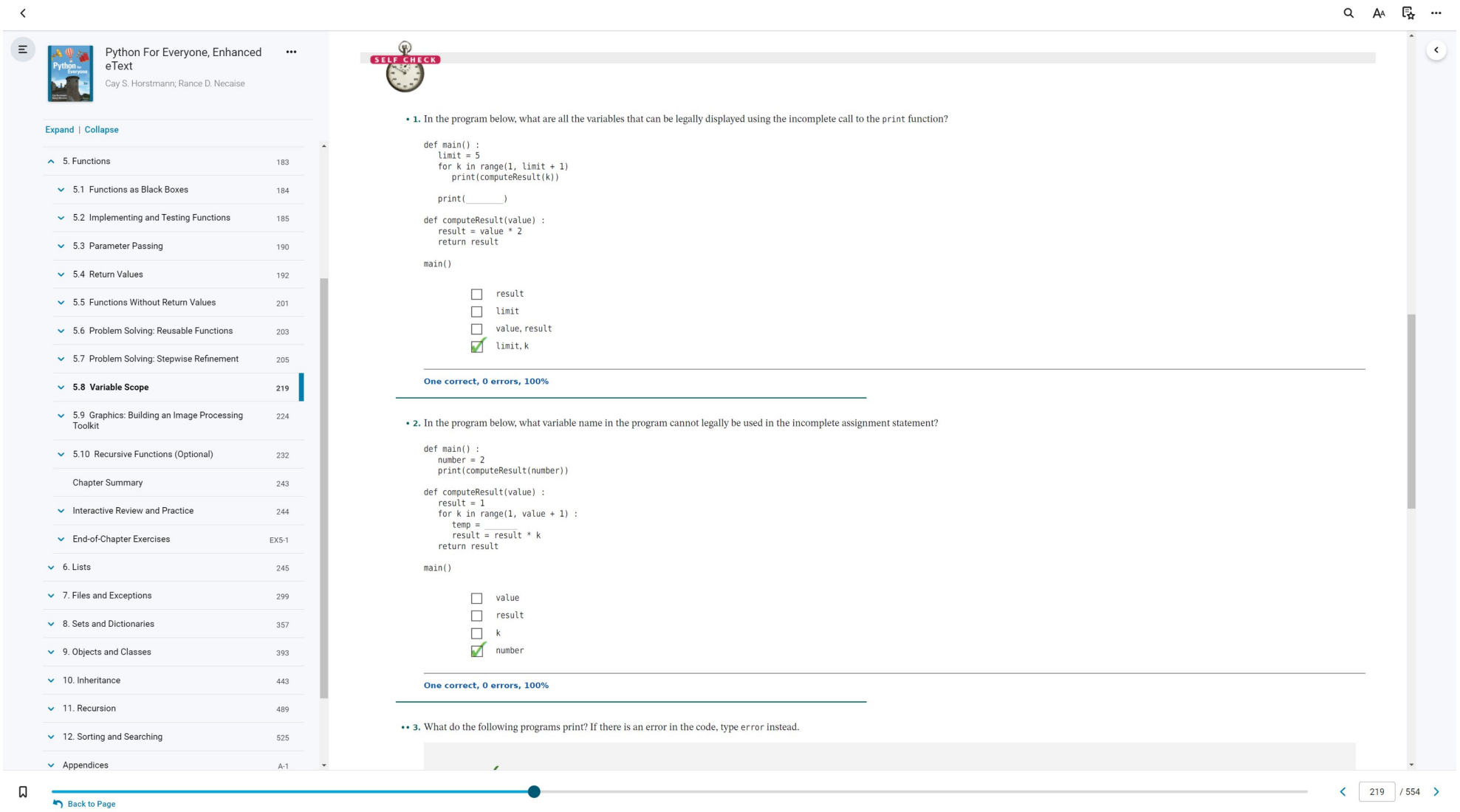
•• 4. Rearrange the following lines of code to produce an improved version of the `intName` function that can be used with numbers in the thousands.

5.7 (p.2)

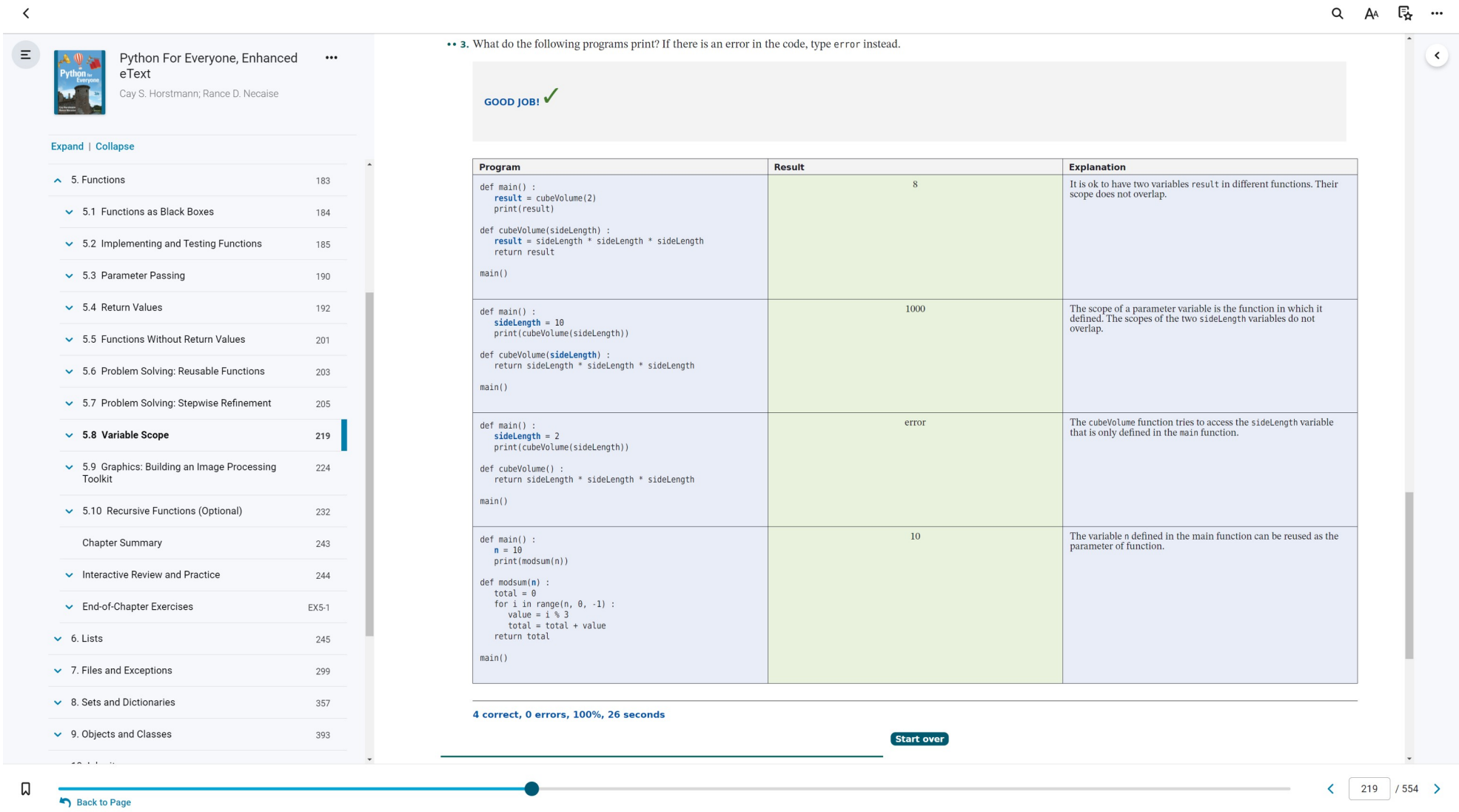

Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
3. Decisions 73
4. Loops 125
5. Functions 183
5.1 Functions as Black Boxes 184
5.2 Implementing and Testing Functions 185
5.3 Parameter Passing 190
5.4 Return Values 192
5.5 Functions Without Return Values 201
5.6 Problem Solving: Reusable Functions 203
5.7 Problem Solving: Stepwise Refinement 205
5.8 Variable Scope 219
5.9 Graphics: Building an Image Processing Toolkit 224
5.10 Recursive Functions (Optional) 232
Chapter Summary 243
Start over
•• 4. Rearrange the following lines of code to produce an improved version of the intName function that can be used with numbers in the thousands.
GOOD JOB!
def intName(number) :
part = number # The part that still needs to be converted
name = "" # The return value
if part >= 1000 :
name = digitName(part // 1000) + " thousand "
part = part % 1000
if part >= 100 :
name = name + digitName(part // 100) + " hundred"
part = part % 100
if part >= 20 :
name = name + " " + tensName(part)
part = part % 10
elif part >= 10 :
name = name + " " + teenName(part)
part = 0
if part > 0 :
name = name + " " + digitName(part)
return name
6 correct, 0 errors, 100%, 38 seconds
Start over
Back to Page
205 / 554

5.7 (p.3)

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 4 Homework – 5.1 – 5.8, 5.10



5.8 (p.1)



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

•• 3. What do the following programs print? If there is an error in the code, type `error` instead.

GOOD JOB! ✓

Program	Result	Explanation

`def main() :`
`  result = cubeVolume(2)`
`  print(result)`
 	
`def cubeVolume(sideLength) :`
`  result = sideLength * sideLength * sideLength`
`  return result`
 	
`main()`
	8	It is ok to have two variables `result` in different functions. Their scope does not overlap.

`def main() :`
`  sideLength = 10`
`  print(cubeVolume(sideLength))`
 	
`def cubeVolume(sideLength) :`
`  return sideLength * sideLength * sideLength`
 	
`main()`
	1000	The scope of a parameter variable is the function in which it defined. The scopes of the two `sideLength` variables do not overlap.

`def main() :`
`  sideLength = 2`
`  print(cubeVolume(sideLength))`
 	
`def cubeVolume() :`
`  return sideLength * sideLength * sideLength`
 	
`main()`
	error	The `cubeVolume` function tries to access the `sideLength` variable that is only defined in the `main` function.

`def main() :`
`  n = 10`
`  print(modsum(n))`
 	
`def modsum(n) :`
`  total = 0`
`  for i in range(n, 0, -1) :`
`  value = i % 3`
`  total = total + value`
`  return total`
 	
`main()`
	10	The variable `n` defined in the main function can be reused as the parameter of function.

4 correct, 0 errors, 100%, 26 seconds

Start over

Back to Page

219 / 554

5.8 (p.2)

4 correct, 0 errors, 100%, 26 seconds

Start over

•• **4.** Trace the following program, paying attention to the scope of each variable.

GOOD JOB! ✓

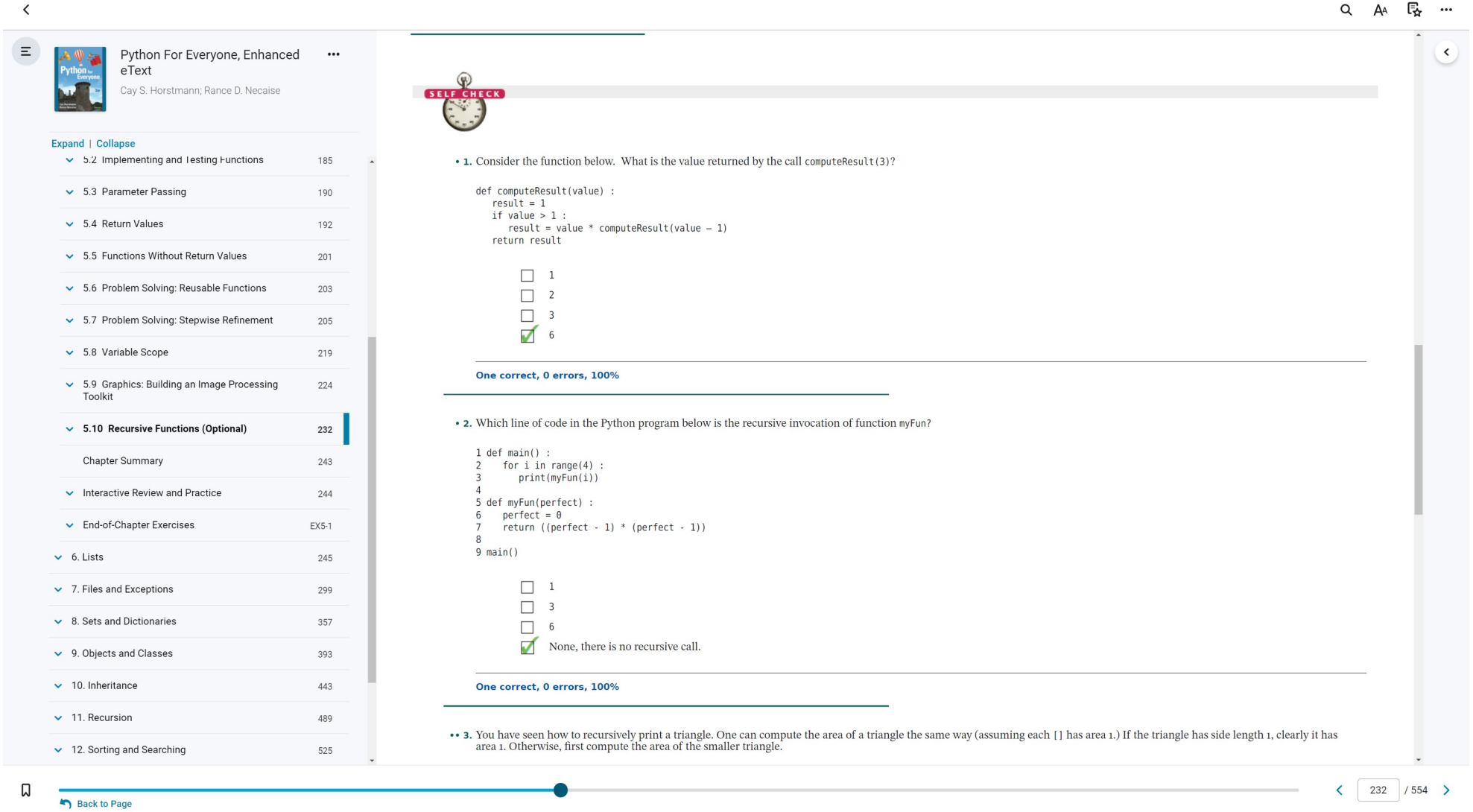
```
def main() :
   a = 2
   b = 6
   n = avg(b, a + b)
   print(n)

def avg(a, b) :
   result = 0
   for n in range(a, b) :
      result = result + n

   result = result / (b - a)
   return result

main()
```

a in main	b in main	n in main	a in avg	b in avg	n in avg	result
2	6		~~6~~	~~8~~		~~0~~
					~~6~~	~~6~~
					~~7~~	~~13~~
						~~6.5~~
			X	X	X	X
		6.5				

13 correct, 0 errors, 100%

Start over

5.8 (p.3)



Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise

Expand | Collapse

SELF CHECK

• 1. Consider the function below. What is the value returned by the call `computeResult(3)`?

```
def computeResult(value) :
   result = 1
   if value > 1 :
      result = value * computeResult(value – 1)
   return result
```

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 6

One correct, 0 errors, 100%

• 2. Which line of code in the Python program below is the recursive invocation of function `myFun`?

```
1 def main() :
2    for i in range(4) :
3       print(myFun(i))
4
5 def myFun(perfect) :
6    perfect = 0
7    return ((perfect - 1) * (perfect - 1))
8
9 main()
```

- [ ] 1
- [ ] 3
- [ ] 6
- [x] None, there is no recursive call.

One correct, 0 errors, 100%

•• 3. You have seen how to recursively print a triangle. One can compute the area of a triangle the same way (assuming each [] has area 1.) If the triangle has side length 1, clearly it has area 1. Otherwise, first compute the area of the smaller triangle.

Back to Page

232 / 554

5.10 (p.1)

•• 3. You have seen how to recursively print a triangle. One can compute the area of a triangle the same way (assuming each [] has area 1.) If the triangle has side length 1, clearly it has area 1. Otherwise, first compute the area of the smaller triangle.

```
[]
[][]
[][][]
[][][][]
```

Then add the side length to account for the bottom strip.

Trace the following code, paying attention to the recursive function calls. Here, `length (1)` and `area (1)` are the variables from the first call to the `triangleArea` function, `length (2)` and `area (2)` are the variables from the second call, and so on.

GOOD JOB! ✓

```
def main() :
   result = triangleArea(4)
   print(result)

def triangleArea(length) :
   if length == 1 :
      area = 1
   else :
      area = triangleArea(length - 1)
      area = area + length

   return area

main()
```

length(1)	area(1)	length(2)	area(2)	length(3)	area(3)	length(4)	area(4)
4		3		2		1	1
					1		
					3		
			3				
			6				
	6						
	10						

23 correct, 0 errors, 100%

Start over

•• 4. What is printed by each of the following programs?

5.10 (p.2)

Start over

•• 4. What is printed by each of the following programs?

GOOD JOB! ✓

Program	Result	Explanation

`def mystery(n) :`
`  if n <= 0 :`
`  return 0`
`  else :`
`  return n + mystery(n - 1)`
 	
`print(mystery(4))`
	10	The recursive function is called 5 times: 4 + 3 + 2 + 1 + 0 = 10

`def mystery(n) :`
`  if n <= 0 :`
`  return 0`
`  else :`
`  return n + mystery(n - 1)`
 	
`print(mystery(-1))`
	0	The recursive function is only called once and the base case is reached, which returns 0.

`def mystery(n) :`
`  if n <= 0 :`
`  return 0`
`  else :`
`  return mystery(n // 2) + 1`
 	
`print(mystery(20))`
	5	The recursive function is called five times: `mystery(10) + 1 = mystery(5) + 2 = mystery(2) + 3 = mystery(1) + 4 = mystery(0) + 5 = 5`

3 correct, 0 errors, 100%

Start over

5.10 (p.3)