A driver of an electric car doesn't have to learn new controls even though the car's engine is very different. Neither does the programmer who uses an object with an improved implementation—as long as the same methods are provided.

SELF CHECK

• **1.** Assuming that `name` is a `str` object, which of the following is a legal method call?

- [ ] `name.print()`
- [ ] `name.input()`
- [x] `name.upper()`
- [ ] `name.pop()`

One correct, 0 errors, 100%

•• **2.** Complete the following table.

GOOD JOB! ✓

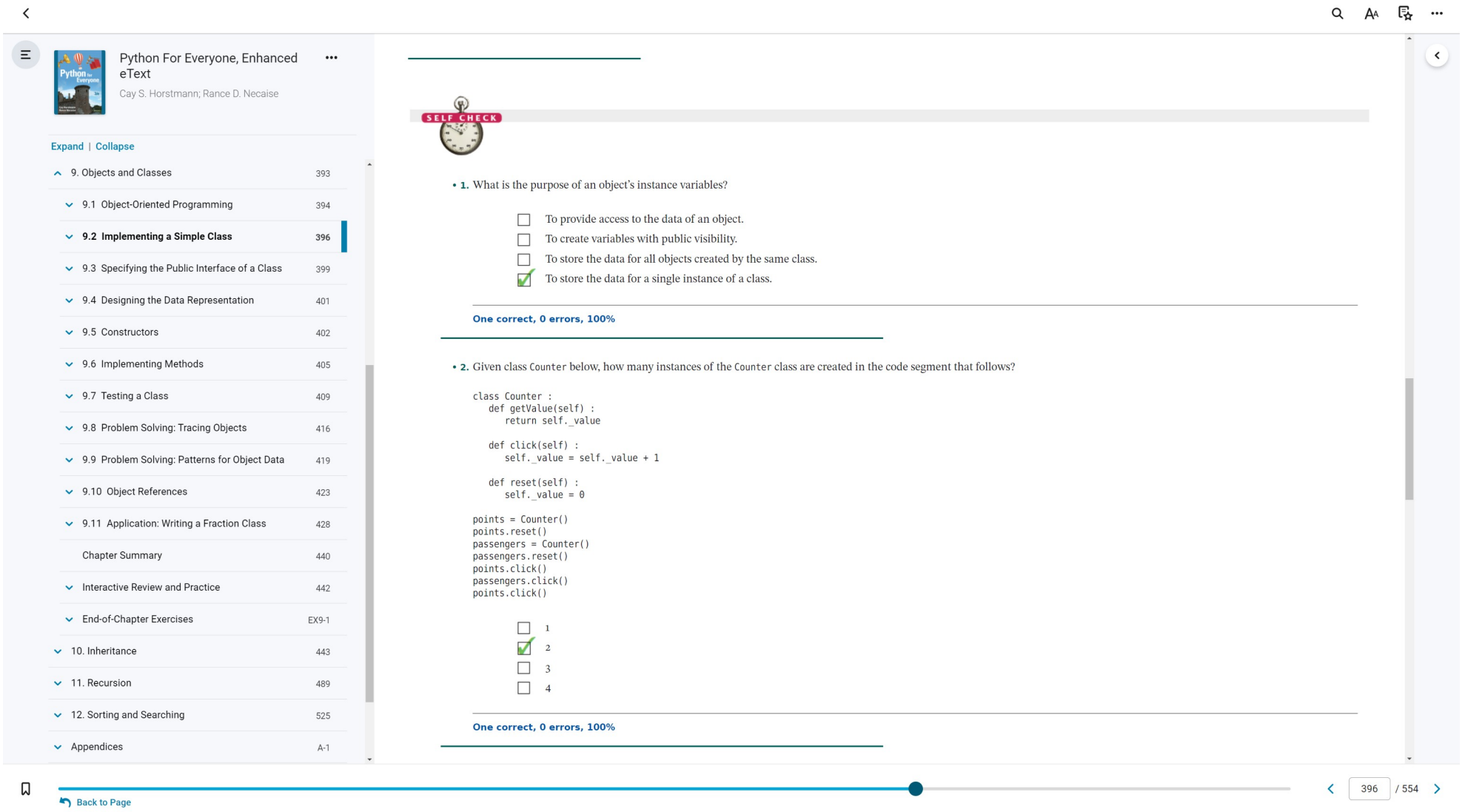
`"Hello"` and `["World"]` are examples of ___.	objects	In an object-oriented program, objects are asked to carry out the tasks that are necessary to achieve a goal.
`upper` and `pop` are examples of ___.	methods	In order to get an object to carry out a task, you invoke a method on it.
`str` and `list` are examples of ___.	classes	A class describes a set of objects with the same behavior.
`upper` and `lower` are methods of the ___ class, but `pop` is not.	`str`	You can invoke the `upper` and `lower` methods on any object of the `str` class, and you apply the `pop` method to objects of the `list` class.
The names and behavior of the methods of a class form the ___ ___ of the class.	public interface	The public interface of a class tells you what you can do with objects of the class.
The user of a class only knows the public interface, not the implementation details, of a class. This is called ___.	encapsulation	The public interface tells you the "what", not the "how". This gives the implementor of the class the freedom to change the implementation.

6 correct, 0 errors, 100%

Start over

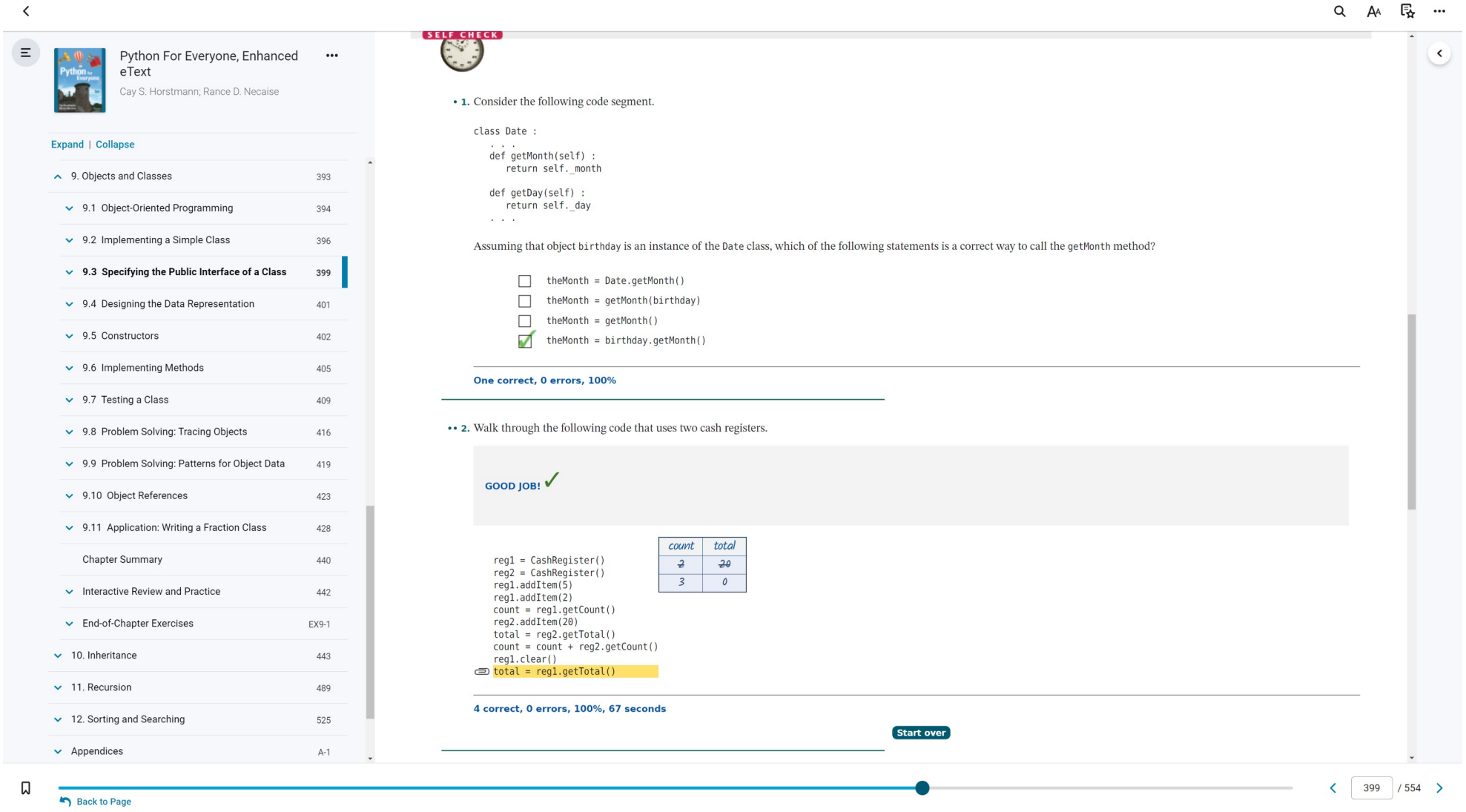
9.1

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 9 Homework – 9.1 – 9.11



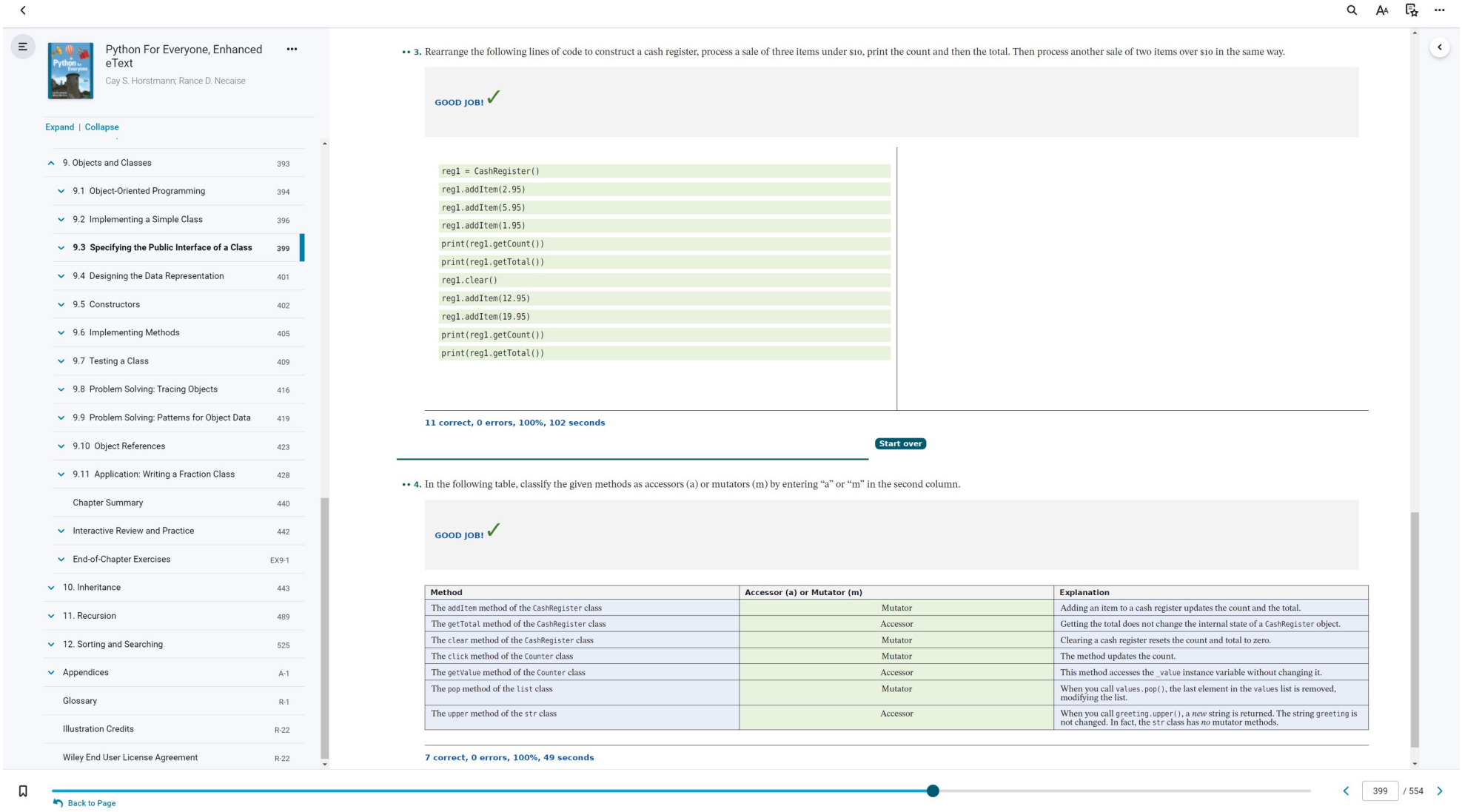
9.2

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 9 Homework – 9.1 – 9.11



9.3

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 9 Homework – 9.1 – 9.11



•• 3. Rearrange the following lines of code to construct a cash register, process a sale of three items under $10, print the count and then the total. Then process another sale of two items over $10 in the same way.

GOOD JOB! ✓

```
reg1 = CashRegister()
reg1.addItem(2.95)
reg1.addItem(5.95)
reg1.addItem(1.95)
print(reg1.getCount())
print(reg1.getTotal())
reg1.clear()
reg1.addItem(12.95)
reg1.addItem(19.95)
print(reg1.getCount())
print(reg1.getTotal())
```

11 correct, 0 errors, 100%, 102 seconds

Start over

•• 4. In the following table, classify the given methods as accessors (a) or mutators (m) by entering "a" or "m" in the second column.

GOOD JOB! ✓

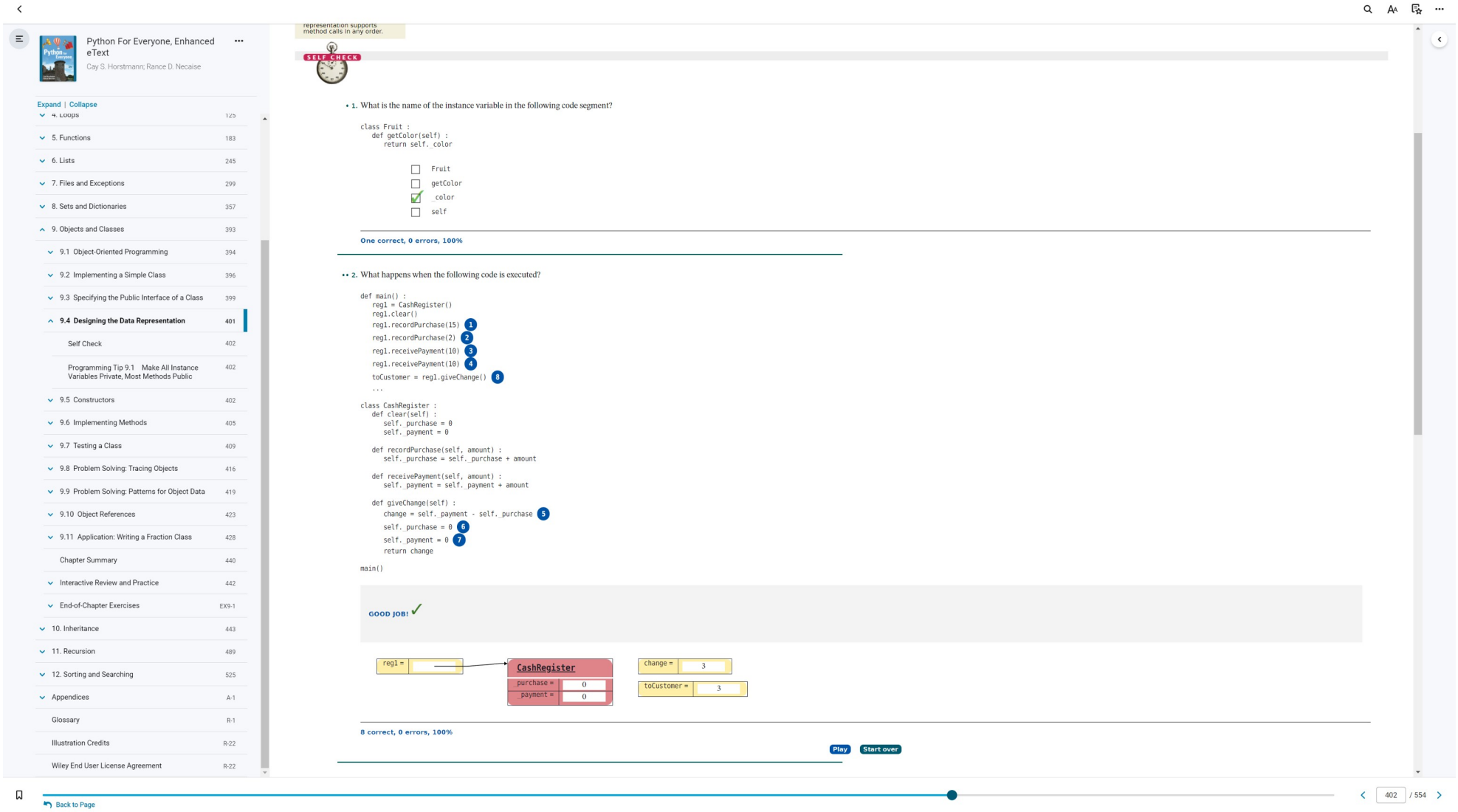
Method	Accessor (a) or Mutator (m)	Explanation
The `addItem` method of the `CashRegister` class	Mutator	Adding an item to a cash register updates the count and the total.
The `getTotal` method of the `CashRegister` class	Accessor	Getting the total does not change the internal state of a `CashRegister` object.
The `clear` method of the `CashRegister` class	Mutator	Clearing a cash register resets the count and total to zero.
The `click` method of the `Counter` class	Mutator	The method updates the count.
The `getValue` method of the `Counter` class	Accessor	This method accesses the `_value` instance variable without changing it.
The `pop` method of the `list` class	Mutator	When you call `values.pop()`, the last element in the `values` list is removed, modifying the list.
The `upper` method of the `str` class	Accessor	When you call `greeting.upper()`, a *new* string is returned. The string `greeting` is not changed. In fact, the `str` class has *no* mutator methods.

7 correct, 0 errors, 100%, 49 seconds

Back to Page

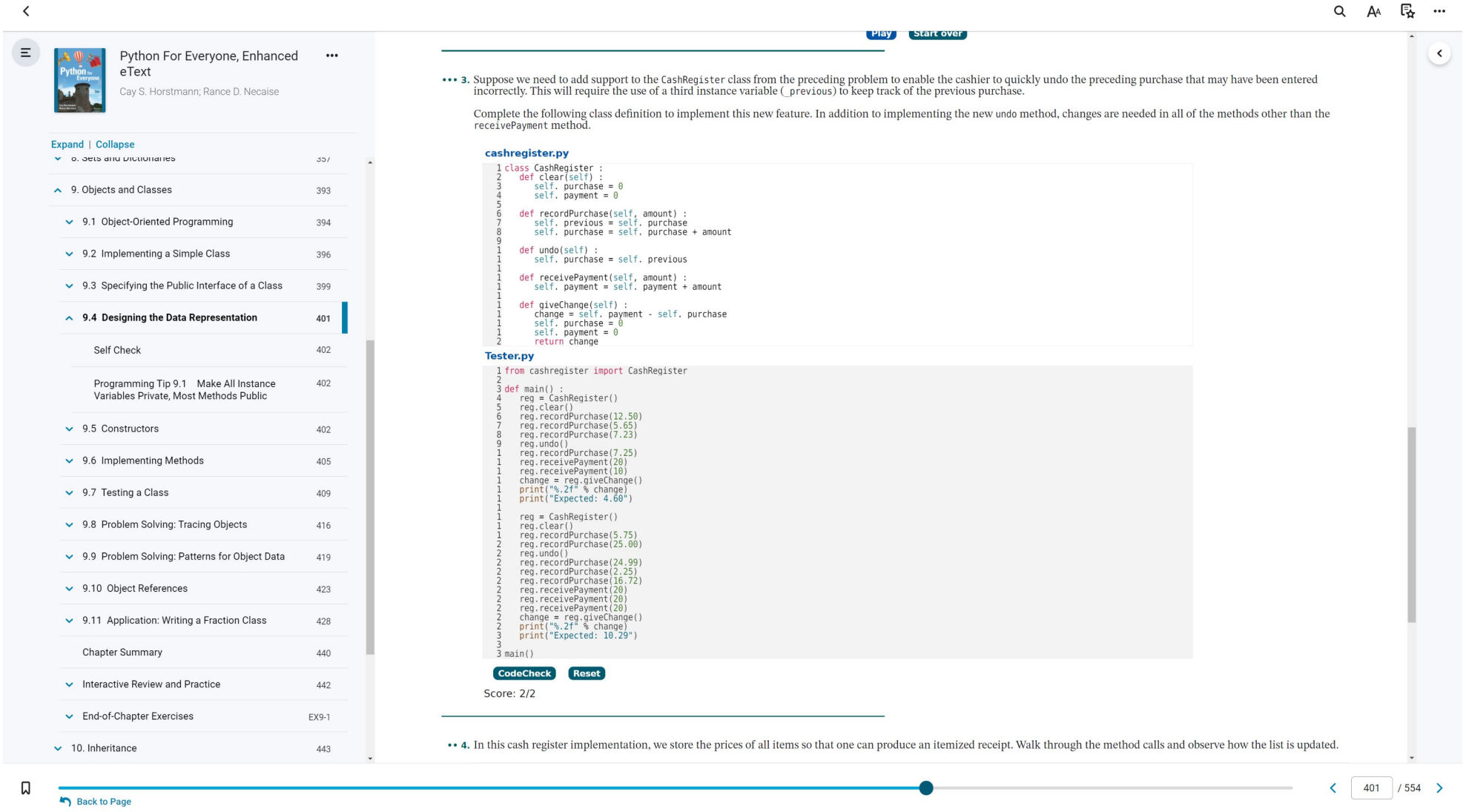
399 / 554

9.3


Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise
Expand | Collapse
5. Functions 183
6. Lists 245
7. Files and Exceptions 299
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
9.11 Application: Writing a Fraction Class 428
Chapter Summary 440
Interactive Review and Practice 442
End-of-Chapter Exercises EX9-1
10. Inheritance 443
11. Recursion 489
12. Sorting and Searching 525
Appendices A-1
Glossary R-1
Illustration Credits R-22
Wiley End User License Agreement R-22
SELF CHECK
• 1. What is the name of the instance variable in the following code segment?
class Fruit :
def getColor(self) :
return self._color
Fruit
getColor
_color
self
One correct, 0 errors, 100%
•• 2. What happens when the following code is executed?
def main() :
reg1 = CashRegister()
reg1.clear()
reg1.recordPurchase(15) 1
reg1.recordPurchase(2) 2
reg1.receivePayment(10) 3
reg1.receivePayment(10) 4
toCustomer = reg1.giveChange() 8
...
class CashRegister :
def clear(self) :
self._purchase = 0
self._payment = 0
def recordPurchase(self, amount) :
self._purchase = self._purchase + amount
def receivePayment(self, amount) :
self._payment = self._payment + amount
def giveChange(self) :
change = self._payment - self._purchase 5
self._purchase = 0 6
self._payment = 0 7
return change
main()
GOOD JOB!
reg1 =
CashRegister
purchase = 0
payment = 0
change = 3
toCustomer = 3
8 correct, 0 errors, 100%
Play
Start over
Back to Page
402 / 554

9.4

Albert Nguyen
CS131-47853 MW 06:00 PM – 08:30 PM
Chapter 9 Homework – 9.1 – 9.11



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

Expand | Collapse

Play Start over

••• 3. Suppose we need to add support to the `CashRegister` class from the preceding problem to enable the cashier to quickly undo the preceding purchase that may have been entered incorrectly. This will require the use of a third instance variable (`_previous`) to keep track of the previous purchase.

Complete the following class definition to implement this new feature. In addition to implementing the new `undo` method, changes are needed in all of the methods other than the `receivePayment` method.

cashregister.py

```
class CashRegister :
   def clear(self) :
      self. purchase = 0
      self. payment = 0

   def recordPurchase(self, amount) :
      self. previous = self. purchase
      self. purchase = self. purchase + amount

   def undo(self) :
      self. purchase = self. previous

   def receivePayment(self, amount) :
      self. payment = self. payment + amount

   def giveChange(self) :
      change = self. payment - self. purchase
      self. purchase = 0
      self. payment = 0
      return change
```

Tester.py

```
from cashregister import CashRegister

def main() :
   reg = CashRegister()
   reg.clear()
   reg.recordPurchase(12.50)
   reg.recordPurchase(5.65)
   reg.recordPurchase(7.23)
   reg.undo()
   reg.recordPurchase(7.25)
   reg.receivePayment(20)
   reg.receivePayment(10)
   change = reg.giveChange()
   print("%.2f" % change)
   print("Expected: 4.60")

   reg = CashRegister()
   reg.clear()
   reg.recordPurchase(5.75)
   reg.recordPurchase(25.00)
   reg.undo()
   reg.recordPurchase(24.99)
   reg.recordPurchase(2.25)
   reg.recordPurchase(16.72)
   reg.receivePayment(20)
   reg.receivePayment(20)
   reg.receivePayment(20)
   change = reg.giveChange()
   print("%.2f" % change)
   print("Expected: 10.29")

main()
```

CodeCheck Reset

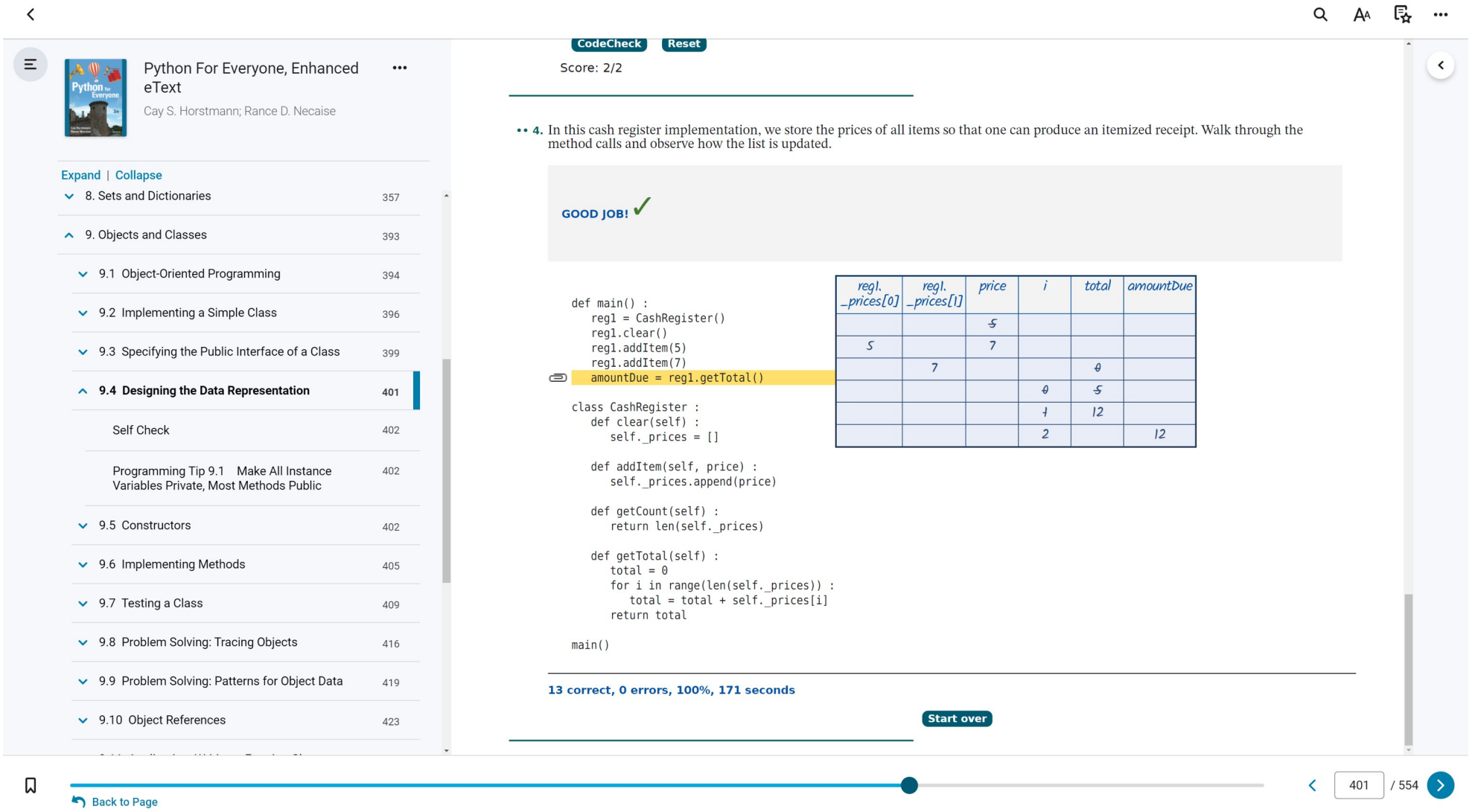
Score: 2/2

•• 4. In this cash register implementation, we store the prices of all items so that one can produce an itemized receipt. Walk through the method calls and observe how the list is updated.

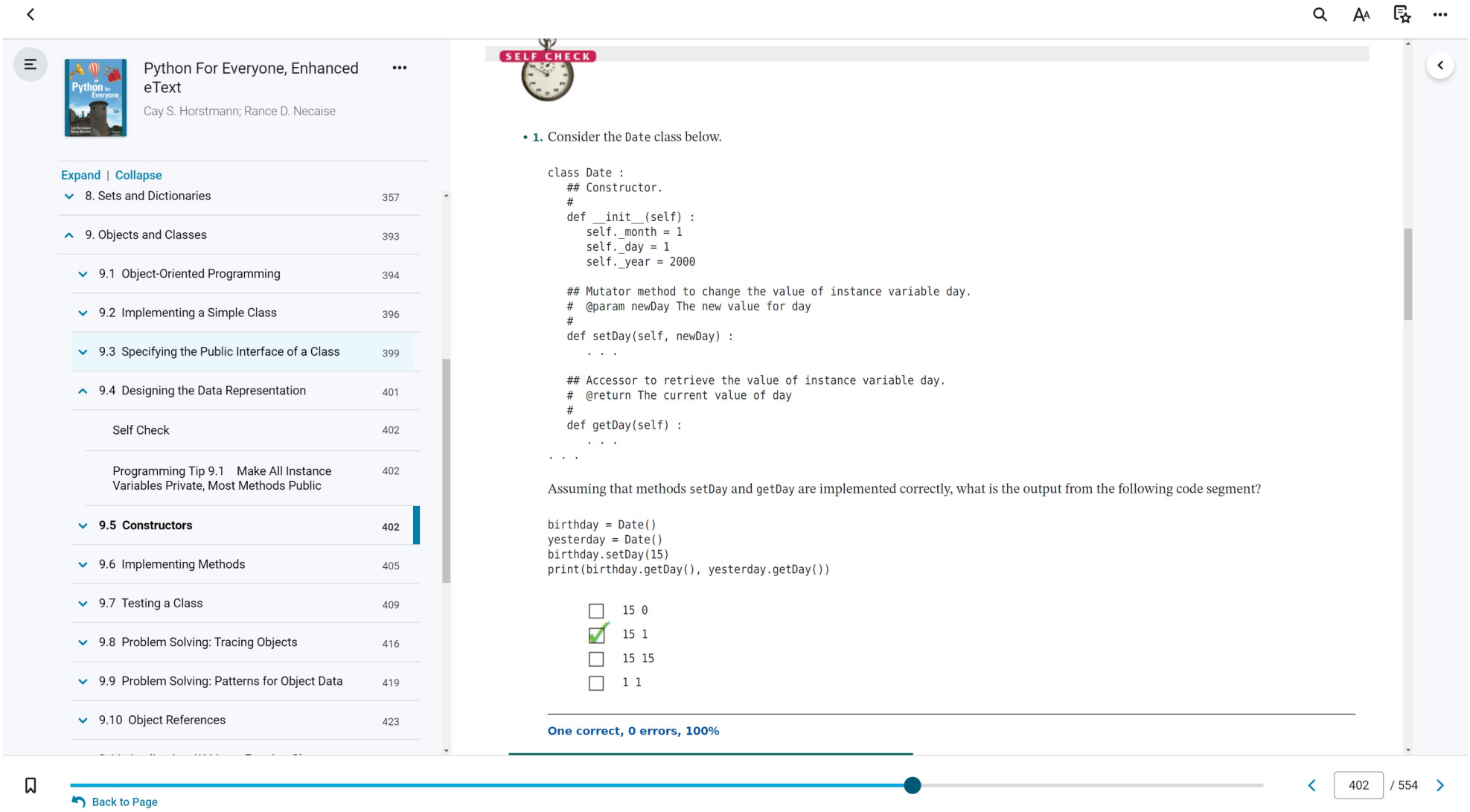
Back to Page

401 / 554

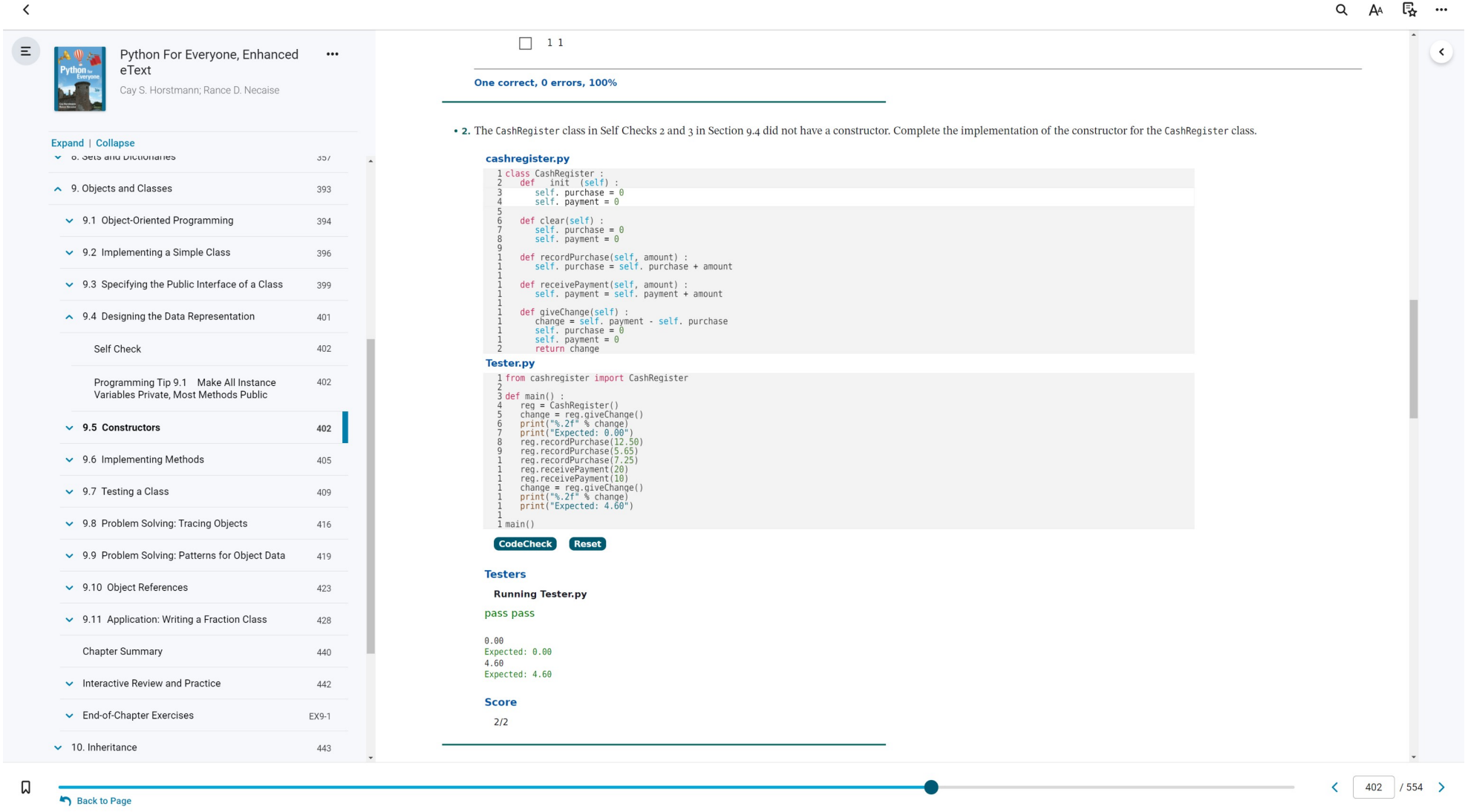
9.4


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
CodeCheck
Reset
Score: 2/2
•• 4. In this cash register implementation, we store the prices of all items so that one can produce an itemized receipt. Walk through the method calls and observe how the list is updated.
GOOD JOB!
def main() :
reg1 = CashRegister()
reg1.clear()
reg1.addItem(5)
reg1.addItem(7)
amountDue = reg1.getTotal()
class CashRegister :
def clear(self) :
self._prices = []
def addItem(self, price) :
self._prices.append(price)
def getCount(self) :
return len(self._prices)
def getTotal(self) :
total = 0
for i in range(len(self._prices)) :
total = total + self._prices[i]
return total
main()
reg1._prices[0]
reg1._prices[1]
price
i
total
amountDue
13 correct, 0 errors, 100%, 171 seconds
Start over
Back to Page
401 / 554

9.4


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
SELF CHECK
1. Consider the Date class below.
class Date :
Constructor.
#
def __init__(self) :
self._month = 1
self._day = 1
self._year = 2000
Mutator method to change the value of instance variable day.
@param newDay The new value for day
#
def setDay(self, newDay) :
. . .
Accessor to retrieve the value of instance variable day.
@return The current value of day
#
def getDay(self) :
. . .
. . .
Assuming that methods setDay and getDay are implemented correctly, what is the output from the following code segment?
birthday = Date()
yesterday = Date()
birthday.setDay(15)
print(birthday.getDay(), yesterday.getDay())
15 0
15 1
15 15
1 1
One correct, 0 errors, 100%
Back to Page
402 / 554

9.5



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

Expand | Collapse

1 1

One correct, 0 errors, 100%

• **2.** The `CashRegister` class in Self Checks 2 and 3 in Section 9.4 did not have a constructor. Complete the implementation of the constructor for the `CashRegister` class.

cashregister.py

```
class CashRegister :
   def __init__(self) :
      self.purchase = 0
      self.payment = 0

   def clear(self) :
      self.purchase = 0
      self.payment = 0

   def recordPurchase(self, amount) :
      self.purchase = self.purchase + amount

   def receivePayment(self, amount) :
      self.payment = self.payment + amount

   def giveChange(self) :
      change = self.payment - self.purchase
      self.purchase = 0
      self.payment = 0
      return change
```

Tester.py

```
from cashregister import CashRegister

def main() :
   reg = CashRegister()
   change = reg.giveChange()
   print("%.2f" % change)
   print("Expected: 0.00")
   reg.recordPurchase(12.50)
   reg.recordPurchase(5.65)
   reg.recordPurchase(7.25)
   reg.receivePayment(20)
   reg.receivePayment(10)
   change = reg.giveChange()
   print("%.2f" % change)
   print("Expected: 4.60")

main()
```

CodeCheck Reset

Testers

Running Tester.py

pass pass

```
0.00
Expected: 0.00
4.60
Expected: 4.60
```

Score

2/2

Back to Page

9.5

•• **3.** Walk through the following program segment and observe how constructors are invoked.

GOOD JOB! ✓

```
def main() :
   acct1 = BankAccount(0)
   acct2 = BankAccount(1000)
   acct1.deposit(500)
   acct2.withdraw(200)

class BankAccount :
   def __init__(self, initialBalance) :
      self._balance = initialBalance

   def deposit(self, amount) :
      self._balance = self._balance + amount

   def withdraw(self, amount) :
      self._balance = self._balance - amount

   def getBalance(self) :
      return self._balance

main()
```

acct1._balance	acct2._balance	initialBalance	amount
		~~0~~	
~~0~~		1000	
	~~1000~~		~~500~~
500			200
	800		

15 correct, 0 errors, 100%, 90 seconds

Start over

•• **4.** Suppose we change our bank account class to store a sequence of all deposits and withdrawals, like this:

```
class BankAccount :
   # Constructor goes here . . .

   def deposit(self, amount) :
      self._amounts.append(amount)
```


9.5

•• **4.** Suppose we change our bank account class to store a sequence of all deposits and withdrawals, like this:

```
class BankAccount :
   # Constructor goes here . . .

   def deposit(self, amount) :
      self._amounts.append(amount)

   def withdraw(self, amount) :
      self._amounts.append(-amount)

   def getBalance(self) :
      total = 0
      for amount in amounts :
         total = total + amount
      return total
```

Rearrange the code below to provide the implementation of the constructor for the `BankAccount` class. The constructor receives the initial balance as an argument and uses that value to initialize a list. Not all lines are useful.

GOOD JOB! ✓

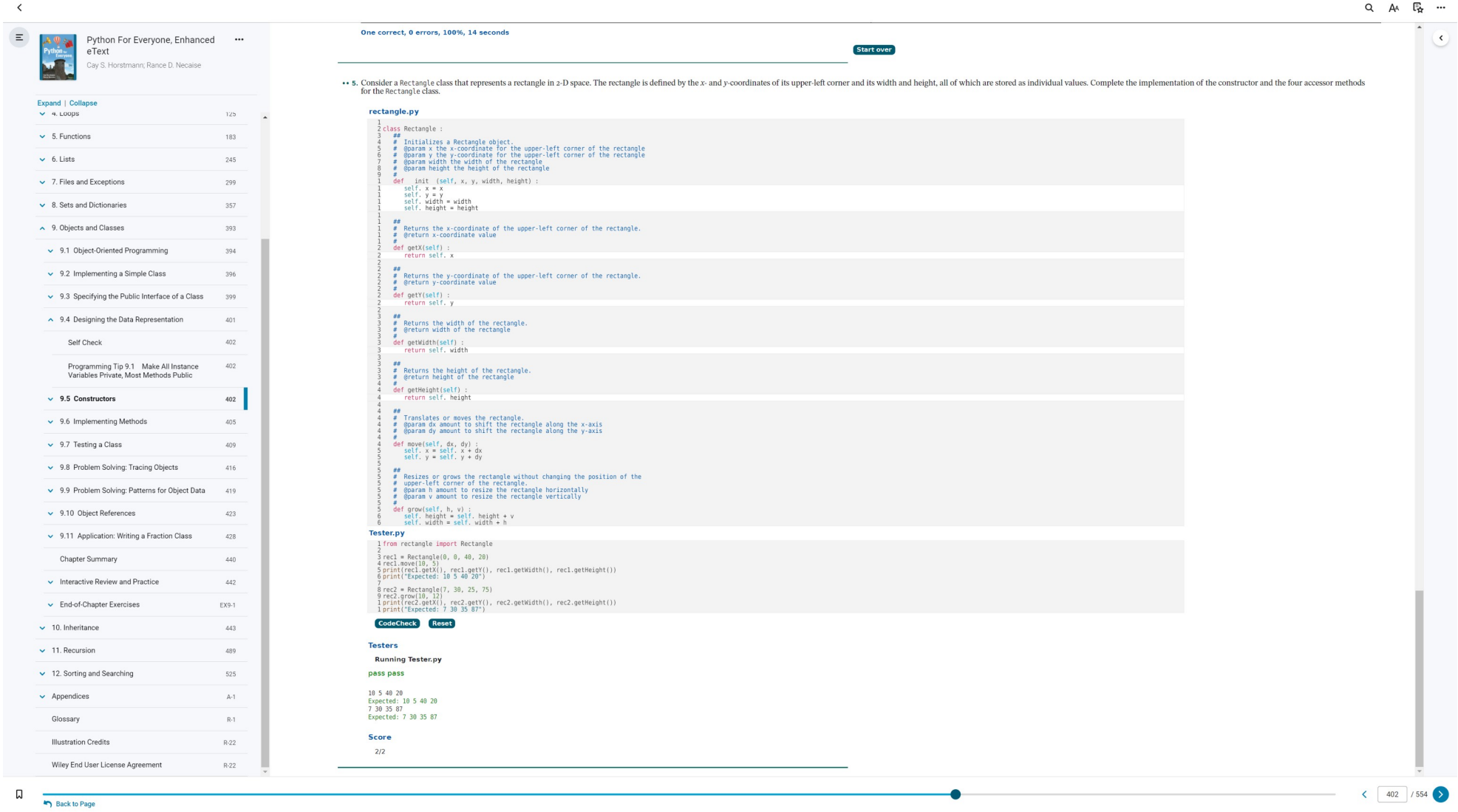
```
def __init__(self, initialBalance) :
   self._amounts = [initialBalance]
```

```
self._amounts = initialBalance
_balance = initialBalance
_amounts = []
self._amounts = []
_amounts = initialBalance
self._balance = initialBalance
_amounts = [initialBalance]
self._amounts.append(initialBalance)
```

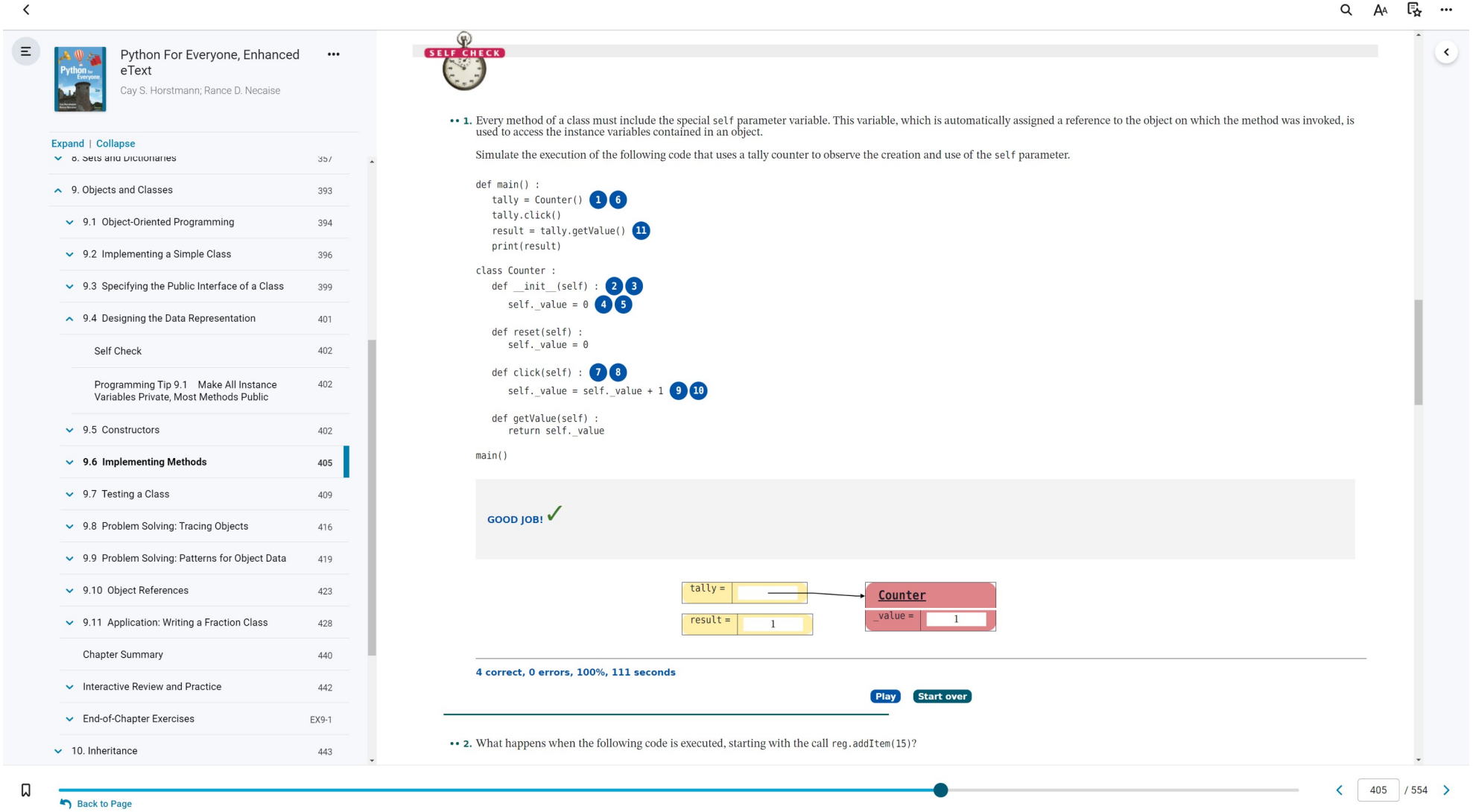
One correct, 0 errors, 100%, 14 seconds

Start over

9.5


Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise
Expand | Collapse
4. Loops 125
5. Functions 183
6. Lists 245
7. Files and Exceptions 299
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
9.11 Application: Writing a Fraction Class 428
Chapter Summary 440
Interactive Review and Practice 442
End-of-Chapter Exercises EX9-1
10. Inheritance 443
11. Recursion 489
12. Sorting and Searching 525
Appendices A-1
Glossary R-1
Illustration Credits R-22
Wiley End User License Agreement R-22
One correct, 0 errors, 100%, 14 seconds
Start over
•• 5. Consider a Rectangle class that represents a rectangle in 2-D space. The rectangle is defined by the x- and y-coordinates of its upper-left corner and its width and height, all of which are stored as individual values. Complete the implementation of the constructor and the four accessor methods for the Rectangle class.
rectangle.py
class Rectangle :
Initializes a Rectangle object.
@param x the x-coordinate for the upper-left corner of the rectangle
@param y the y-coordinate for the upper-left corner of the rectangle
@param width the width of the rectangle
@param height the height of the rectangle
#
def __init__(self, x, y, width, height) :
self.x = x
self.y = y
self.width = width
self.height = height
Returns the x-coordinate of the upper-left corner of the rectangle.
@return x-coordinate value
#
def getX(self) :
return self.x
Returns the y-coordinate of the upper-left corner of the rectangle.
@return y-coordinate value
#
def getY(self) :
return self.y
Returns the width of the rectangle.
@return width of the rectangle
#
def getWidth(self) :
return self.width
Returns the height of the rectangle.
@return height of the rectangle
#
def getHeight(self) :
return self.height
Translates or moves the rectangle.
@param dx amount to shift the rectangle along the x-axis
@param dy amount to shift the rectangle along the y-axis
#
def move(self, dx, dy) :
self.x = self.x + dx
self.y = self.y + dy
Resizes or grows the rectangle without changing the position of the
upper-left corner of the rectangle.
@param h amount to resize the rectangle horizontally
@param v amount to resize the rectangle vertically
#
def grow(self, h, v) :
self.height = self.height + v
self.width = self.width + h
Tester.py
from rectangle import Rectangle
rec1 = Rectangle(0, 0, 40, 20)
rec1.move(10, 5)
print(rec1.getX(), rec1.getY(), rec1.getWidth(), rec1.getHeight())
print("Expected: 10 5 40 20")
rec2 = Rectangle(7, 30, 25, 75)
rec2.grow(10, 12)
print(rec2.getX(), rec2.getY(), rec2.getWidth(), rec2.getHeight())
print("Expected: 7 30 35 87")
CodeCheck
Reset
Testers
Running Tester.py
pass pass
10 5 40 20
Expected: 10 5 40 20
7 30 35 87
Expected: 7 30 35 87
Score
2/2
Back to Page
402 / 554

9.5

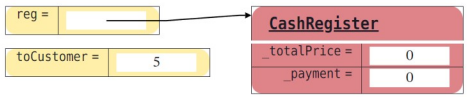

Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
9.11 Application: Writing a Fraction Class 428
Chapter Summary 440
Interactive Review and Practice 442
End-of-Chapter Exercises EX9-1
10. Inheritance 443
SELF CHECK
•• 1. Every method of a class must include the special self parameter variable. This variable, which is automatically assigned a reference to the object on which the method was invoked, is used to access the instance variables contained in an object.
Simulate the execution of the following code that uses a tally counter to observe the creation and use of the self parameter.
def main() :
tally = Counter() 1 6
tally.click()
result = tally.getValue() 11
print(result)
class Counter :
def __init__(self) : 2 3
self._value = 0 4 5
def reset(self) :
self._value = 0
def click(self) : 7 8
self._value = self._value + 1 9 10
def getValue(self) :
return self._value
main()
GOOD JOB!
tally =
result = 1
Counter
_value = 1
4 correct, 0 errors, 100%, 111 seconds
Play
Start over
•• 2. What happens when the following code is executed, starting with the call reg.addItem(15)?
Back to Page
405 / 554

9.6

•• **2.** What happens when the following code is executed, starting with the call `reg.addItem(15)`?

```
def main() :
   reg = CashRegister()
   reg.addItem(15)
   reg.enterPayment(20)
   toCustomer = reg.giveChange() 17
   . . .

class CashRegister :
   def __init__(self) :
      self._totalPrice = 0
      self._payment = 0

   def addItem(self, amount) : 1 2 3
      self._totalPrice = self._totalPrice + amount 4 5

   def enterPayment(self, amount2) : 6 7 8
      self._payment = self._payment + amount2 9 10

   def giveChange(self) : 11
      change = self._payment - self._totalPrice 12 13
      self._totalPrice = 0 14
      self._payment = 0 15
      return change 16

main()
```

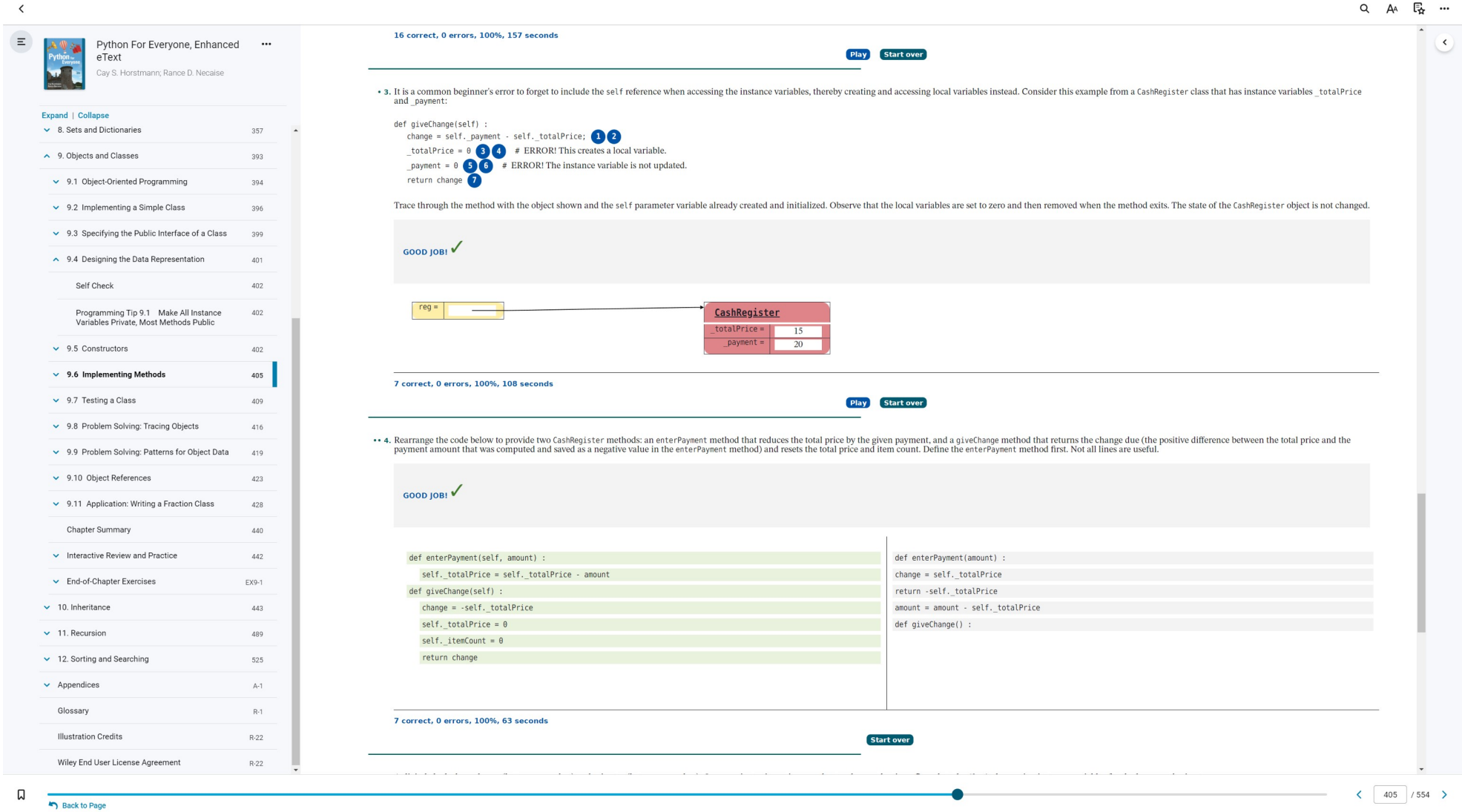
GOOD JOB! ✓



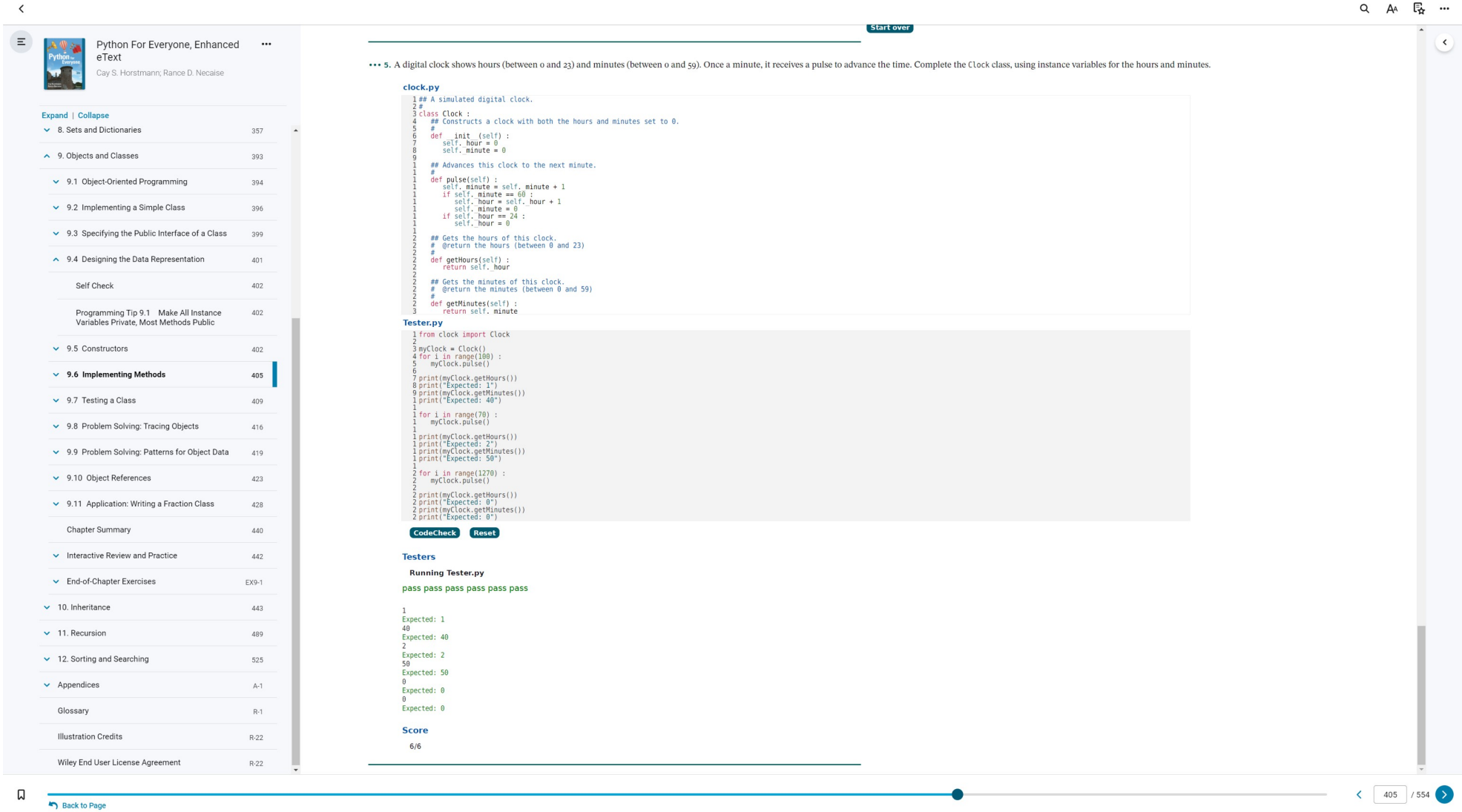
16 correct, 0 errors, 100%, 157 seconds

Play Start over

• **3.** It is a common beginner's error to forget to include the `self` reference when accessing the instance variables, thereby creating and accessing local variables instead. Consider this example from a `CashRegister` class that has instance variables `_totalPrice` and `_payment`:


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
8. Sets and Dictionaries 357
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
9.7 Testing a Class 409
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.10 Object References 423
9.11 Application: Writing a Fraction Class 428
Chapter Summary 440
Interactive Review and Practice 442
End-of-Chapter Exercises EX9-1
10. Inheritance 443
11. Recursion 489
12. Sorting and Searching 525
Appendices A-1
Glossary R-1
Illustration Credits R-22
Wiley End User License Agreement R-22
16 correct, 0 errors, 100%, 157 seconds
Play
Start over
• 3. It is a common beginner's error to forget to include the self reference when accessing the instance variables, thereby creating and accessing local variables instead. Consider this example from a CashRegister class that has instance variables _totalPrice and _payment:
def giveChange(self) :
change = self._payment - self._totalPrice;
_totalPrice = 0
# ERROR! This creates a local variable.
_payment = 0
# ERROR! The instance variable is not updated.
return change
Trace through the method with the object shown and the self parameter variable already created and initialized. Observe that the local variables are set to zero and then removed when the method exits. The state of the CashRegister object is not changed.
GOOD JOB!
reg =
CashRegister
_totalPrice = 15
_payment = 20
7 correct, 0 errors, 100%, 108 seconds
Play
Start over
•• 4. Rearrange the code below to provide two CashRegister methods: an enterPayment method that reduces the total price by the given payment, and a giveChange method that returns the change due (the positive difference between the total price and the payment amount that was computed and saved as a negative value in the enterPayment method) and resets the total price and item count. Define the enterPayment method first. Not all lines are useful.
GOOD JOB!
def enterPayment(self, amount) :
self._totalPrice = self._totalPrice - amount
def giveChange(self) :
change = -self._totalPrice
self._totalPrice = 0
self._itemCount = 0
return change
def enterPayment(amount) :
change = self._totalPrice
return -self._totalPrice
amount = amount - self._totalPrice
def giveChange() :
7 correct, 0 errors, 100%, 63 seconds
Start over
Back to Page
405 / 554

9.6



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

Expand | Collapse

Start over

••• 5. A digital clock shows hours (between 0 and 23) and minutes (between 0 and 59). Once a minute, it receives a pulse to advance the time. Complete the `Clock` class, using instance variables for the hours and minutes.

clock.py

```
## A simulated digital clock.
#
class Clock :
   ## Constructs a clock with both the hours and minutes set to 0.
   #
   def __init__(self) :
      self._hour = 0
      self._minute = 0

   ## Advances this clock to the next minute.
   #
   def pulse(self) :
      self._minute = self._minute + 1
      if self._minute == 60 :
         self._hour = self._hour + 1
         self._minute = 0
      if self._hour == 24 :
         self._hour = 0

   ## Gets the hours of this clock.
   #  @return the hours (between 0 and 23)
   #
   def getHours(self) :
      return self._hour

   ## Gets the minutes of this clock.
   #  @return the minutes (between 0 and 59)
   #
   def getMinutes(self) :
      return self._minute
```

Tester.py

```
from clock import Clock

myClock = Clock()
for i in range(100) :
   myClock.pulse()

print(myClock.getHours())
print("Expected: 1")
print(myClock.getMinutes())
print("Expected: 40")

for i in range(70) :
   myClock.pulse()

print(myClock.getHours())
print("Expected: 2")
print(myClock.getMinutes())
print("Expected: 50")

for i in range(1270) :
   myClock.pulse()

print(myClock.getHours())
print("Expected: 0")
print(myClock.getMinutes())
print("Expected: 0")
```

CodeCheck Reset

Testers

Running Tester.py

pass pass pass pass pass pass

```
1
Expected: 1
40
Expected: 40
2
Expected: 2
50
Expected: 50
0
Expected: 0
0
Expected: 0
```

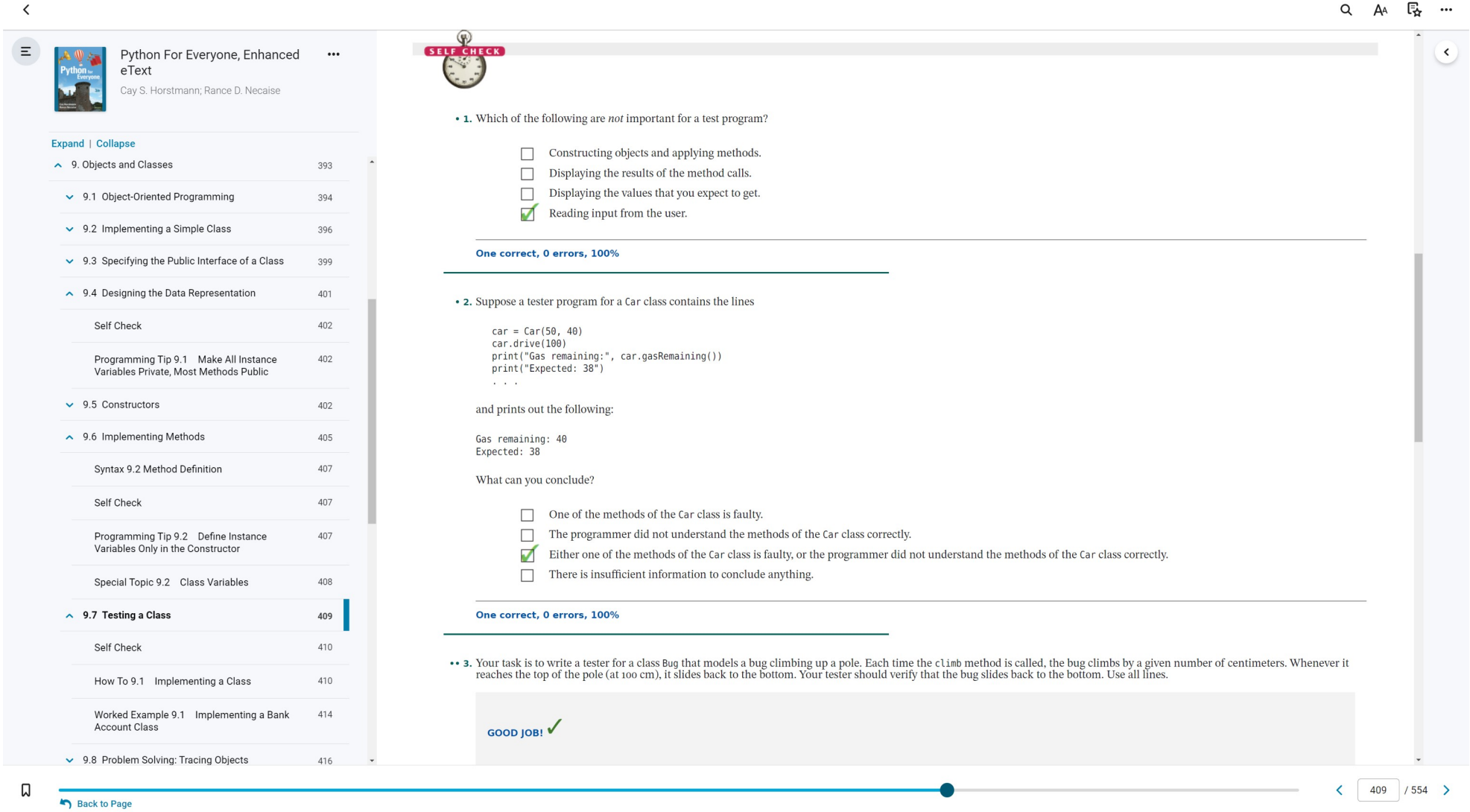
Score

6/6

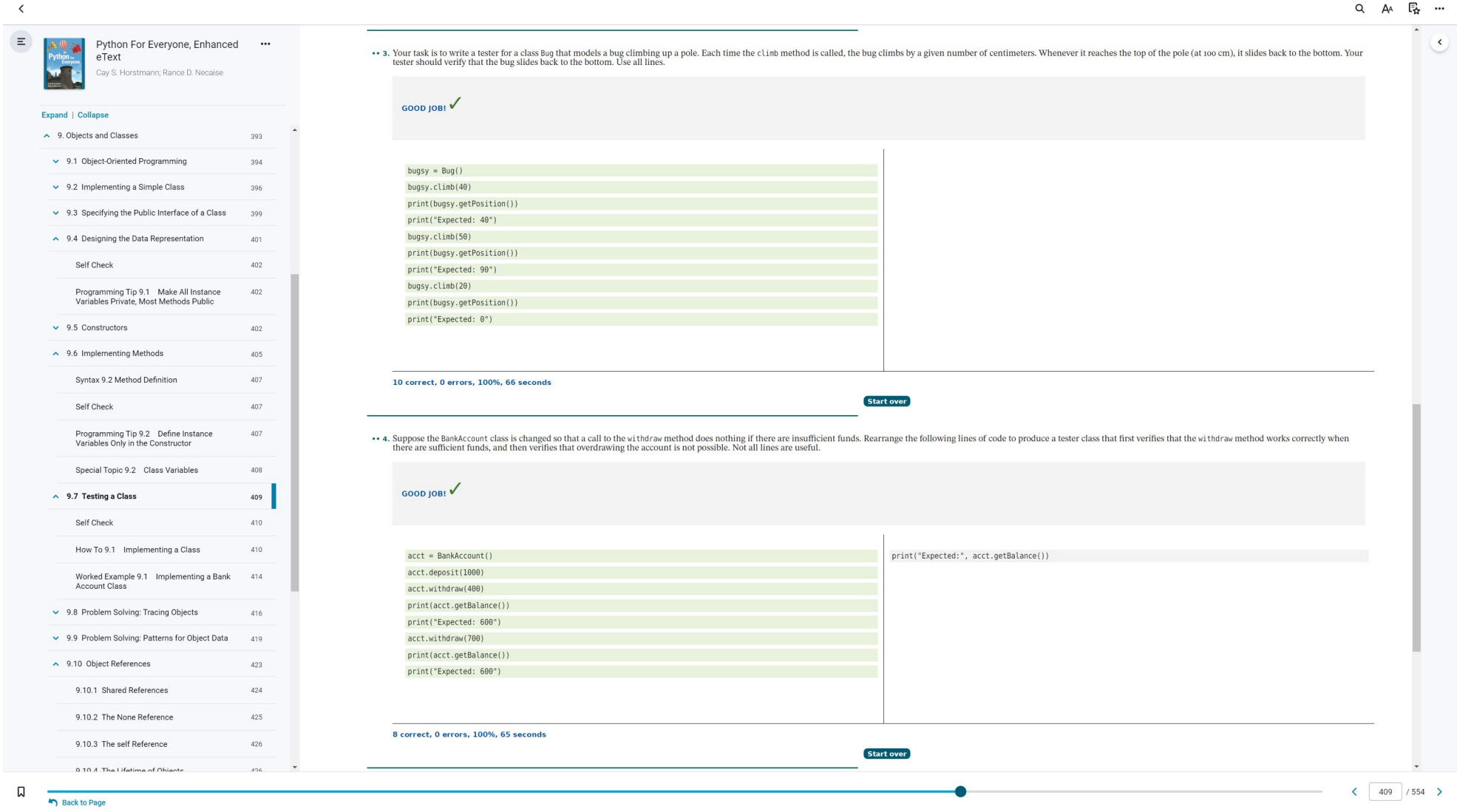
Back to Page

405 / 554

9.6


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
9. Objects and Classes 393
9.1 Object-Oriented Programming 394
9.2 Implementing a Simple Class 396
9.3 Specifying the Public Interface of a Class 399
9.4 Designing the Data Representation 401
Self Check 402
Programming Tip 9.1 Make All Instance Variables Private, Most Methods Public 402
9.5 Constructors 402
9.6 Implementing Methods 405
Syntax 9.2 Method Definition 407
Self Check 407
Programming Tip 9.2 Define Instance Variables Only in the Constructor 407
Special Topic 9.2 Class Variables 408
9.7 Testing a Class 409
Self Check 410
How To 9.1 Implementing a Class 410
Worked Example 9.1 Implementing a Bank Account Class 414
9.8 Problem Solving: Tracing Objects 416
SELF CHECK
• 1. Which of the following are not important for a test program?
Constructing objects and applying methods.
Displaying the results of the method calls.
Displaying the values that you expect to get.
Reading input from the user.
One correct, 0 errors, 100%
• 2. Suppose a tester program for a Car class contains the lines
car = Car(50, 40)
car.drive(100)
print("Gas remaining:", car.gasRemaining())
print("Expected: 38")
. . .
and prints out the following:
Gas remaining: 40
Expected: 38
What can you conclude?
One of the methods of the Car class is faulty.
The programmer did not understand the methods of the Car class correctly.
Either one of the methods of the Car class is faulty, or the programmer did not understand the methods of the Car class correctly.
There is insufficient information to conclude anything.
One correct, 0 errors, 100%
•• 3. Your task is to write a tester for a class Bug that models a bug climbing up a pole. Each time the climb method is called, the bug climbs by a given number of centimeters. Whenever it reaches the top of the pole (at 100 cm), it slides back to the bottom. Your tester should verify that the bug slides back to the bottom. Use all lines.
GOOD JOB!
Back to Page
409 / 554

9.7



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

Expand | Collapse

•• 3. Your task is to write a tester for a class Bug that models a bug climbing up a pole. Each time the climb method is called, the bug climbs by a given number of centimeters. Whenever it reaches the top of the pole (at 100 cm), it slides back to the bottom. Your tester should verify that the bug slides back to the bottom. Use all lines.

GOOD JOB! ✓

```
bugsy = Bug()
bugsy.climb(40)
print(bugsy.getPosition())
print("Expected: 40")
bugsy.climb(50)
print(bugsy.getPosition())
print("Expected: 90")
bugsy.climb(20)
print(bugsy.getPosition())
print("Expected: 0")
```

10 correct, 0 errors, 100%, 66 seconds

Start over

•• 4. Suppose the BankAccount class is changed so that a call to the withdraw method does nothing if there are insufficient funds. Rearrange the following lines of code to produce a tester class that first verifies that the withdraw method works correctly when there are sufficient funds, and then verifies that overdrawing the account is not possible. Not all lines are useful.

GOOD JOB! ✓

```
acct = BankAccount()
acct.deposit(1000)
acct.withdraw(400)
print(acct.getBalance())
print("Expected: 600")
acct.withdraw(700)
print(acct.getBalance())
print("Expected: 600")
```

```
print("Expected:", acct.getBalance())
```

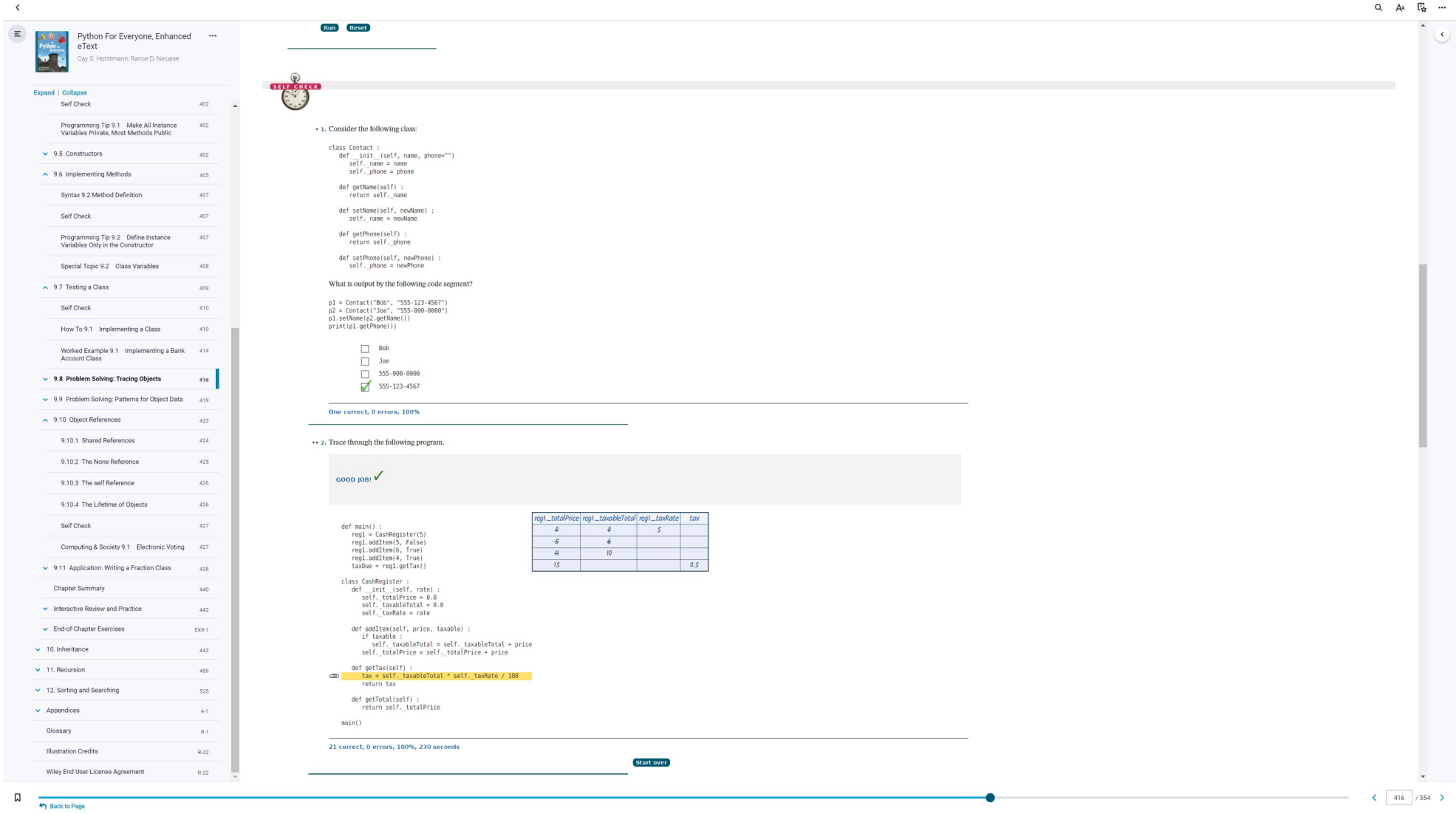
8 correct, 0 errors, 100%, 65 seconds

Start over

Back to Page

409 / 554

9.7



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

Expand | Collapse

Run Reset

SELF CHECK

• 1. Consider the following class:

```
class Contact :
   def __init__(self, name, phone="")
      self._name = name
      self._phone = phone

   def getName(self) :
      return self._name

   def setName(self, newName) :
      self._name = newName

   def getPhone(self) :
      return self._phone

   def setPhone(self, newPhone) :
      self._phone = newPhone
```

What is output by the following code segment?

```
p1 = Contact("Bob", "555-123-4567")
p2 = Contact("Joe", "555-000-0000")
p1.setName(p2.getName())
print(p1.getPhone())
```

- ☐ Bob
- ☐ Joe
- ☐ 555-000-0000
- ☑ 555-123-4567

One correct, 0 errors, 100%

•• 2. Trace through the following program.

GOOD JOB! ✓

```
def main() :
   reg1 = CashRegister(5)
   reg1.addItem(5, False)
   reg1.addItem(6, True)
   reg1.addItem(4, True)
   taxDue = reg1.getTax()

class CashRegister :
   def __init__(self, rate) :
      self._totalPrice = 0.0
      self._taxableTotal = 0.0
      self._taxRate = rate

   def addItem(self, price, taxable) :
      if taxable :
         self._taxableTotal = self._taxableTotal + price
      self._totalPrice = self._totalPrice + price

   def getTax(self) :
      tax = self._taxableTotal * self._taxRate / 100
      return tax

   def getTotal(self) :
      return self._totalPrice

main()
```

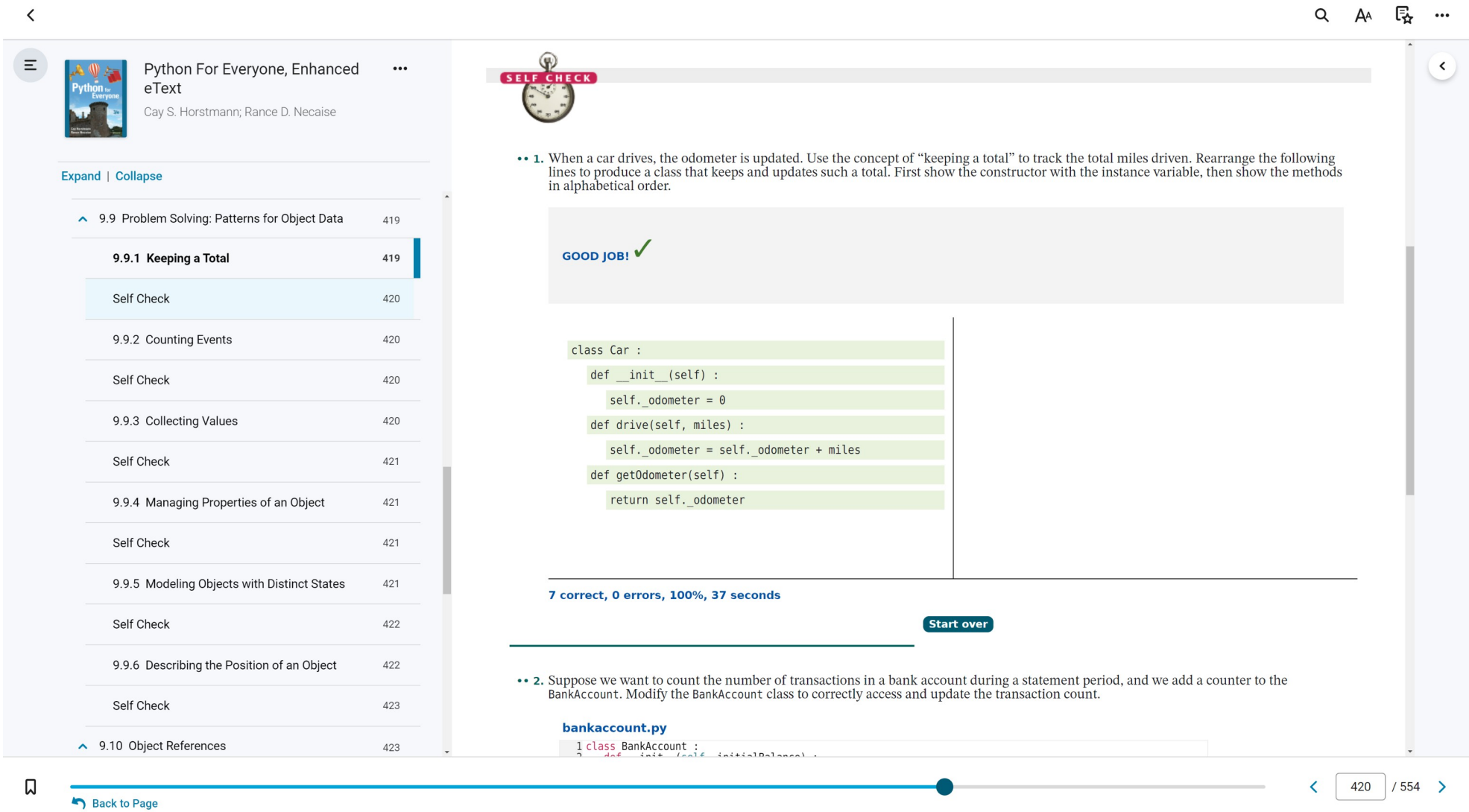
reg1._totalPrice	reg1._taxableTotal	reg1._taxRate	tax
0	0	5	
~~5~~	~~6~~		
11	10		
15			0.5

21 correct, 0 errors, 100%, 230 seconds

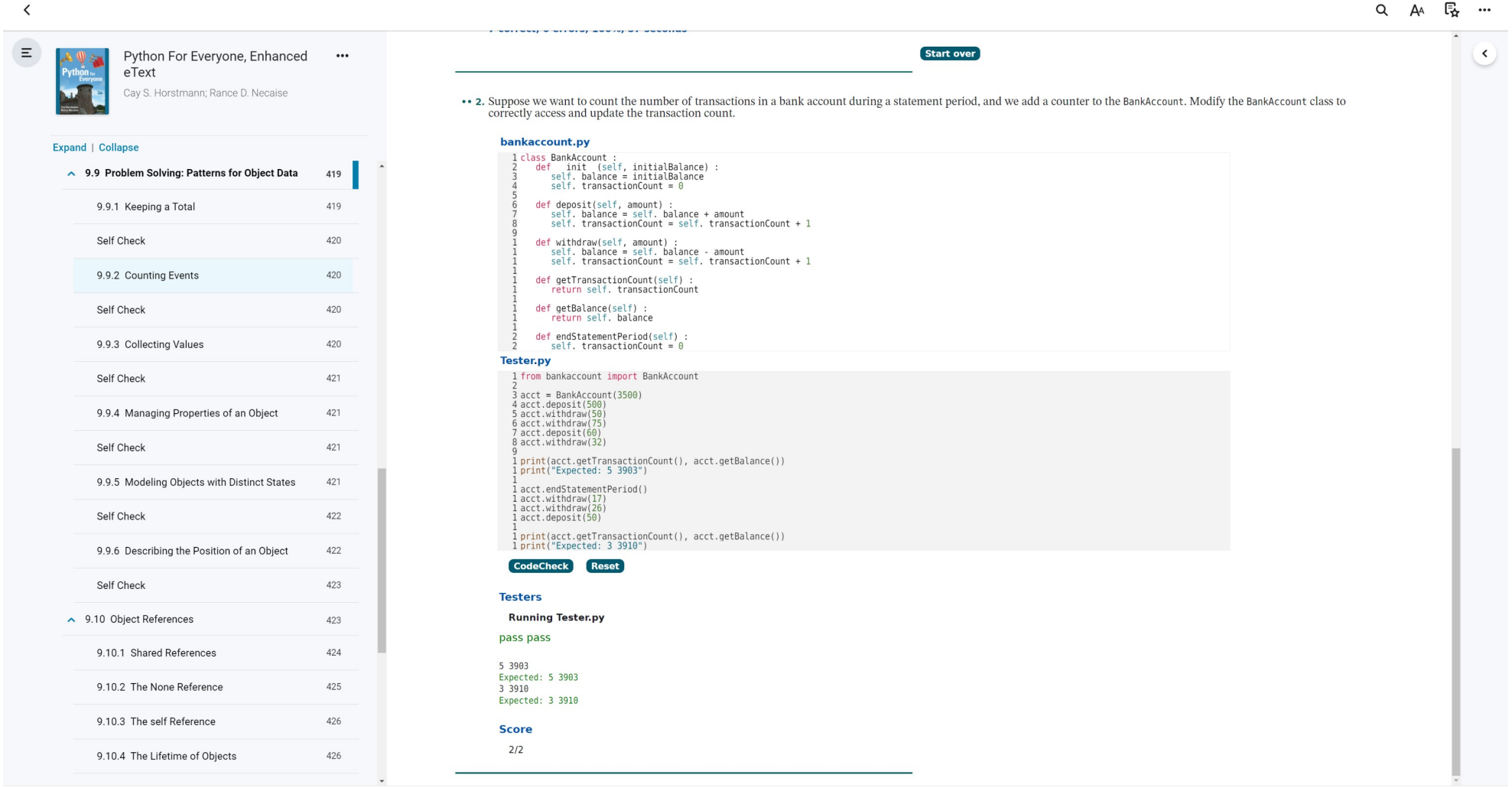
Start over

Back to Page 416 / 554

9.8


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
9.9 Problem Solving: Patterns for Object Data 419
9.9.1 Keeping a Total 419
Self Check 420
9.9.2 Counting Events 420
Self Check 420
9.9.3 Collecting Values 420
Self Check 421
9.9.4 Managing Properties of an Object 421
Self Check 421
9.9.5 Modeling Objects with Distinct States 421
Self Check 422
9.9.6 Describing the Position of an Object 422
Self Check 423
9.10 Object References 423
SELF CHECK
•• 1. When a car drives, the odometer is updated. Use the concept of "keeping a total" to track the total miles driven. Rearrange the following lines to produce a class that keeps and updates such a total. First show the constructor with the instance variable, then show the methods in alphabetical order.
GOOD JOB!
class Car :
def __init__(self) :
self._odometer = 0
def drive(self, miles) :
self._odometer = self._odometer + miles
def getOdometer(self) :
return self._odometer
7 correct, 0 errors, 100%, 37 seconds
Start over
•• 2. Suppose we want to count the number of transactions in a bank account during a statement period, and we add a counter to the BankAccount. Modify the BankAccount class to correctly access and update the transaction count.
bankaccount.py
1 class BankAccount :
Back to Page
420 / 554

9.9



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

Expand | Collapse

Start over

•• 2. Suppose we want to count the number of transactions in a bank account during a statement period, and we add a counter to the `BankAccount`. Modify the `BankAccount` class to correctly access and update the transaction count.

bankaccount.py

```
class BankAccount :
   def __init__(self, initialBalance) :
      self.balance = initialBalance
      self.transactionCount = 0

   def deposit(self, amount) :
      self.balance = self.balance + amount
      self.transactionCount = self.transactionCount + 1

   def withdraw(self, amount) :
      self.balance = self.balance - amount
      self.transactionCount = self.transactionCount + 1

   def getTransactionCount(self) :
      return self.transactionCount

   def getBalance(self) :
      return self.balance

   def endStatementPeriod(self) :
      self.transactionCount = 0
```

Tester.py

```
from bankaccount import BankAccount

acct = BankAccount(3500)
acct.deposit(500)
acct.withdraw(50)
acct.withdraw(75)
acct.deposit(60)
acct.withdraw(32)

print(acct.getTransactionCount(), acct.getBalance())
print("Expected: 5 3903")

acct.endStatementPeriod()
acct.withdraw(17)
acct.withdraw(26)
acct.deposit(50)

print(acct.getTransactionCount(), acct.getBalance())
print("Expected: 3 3910")
```

CodeCheck Reset

Testers

Running Tester.py

pass pass

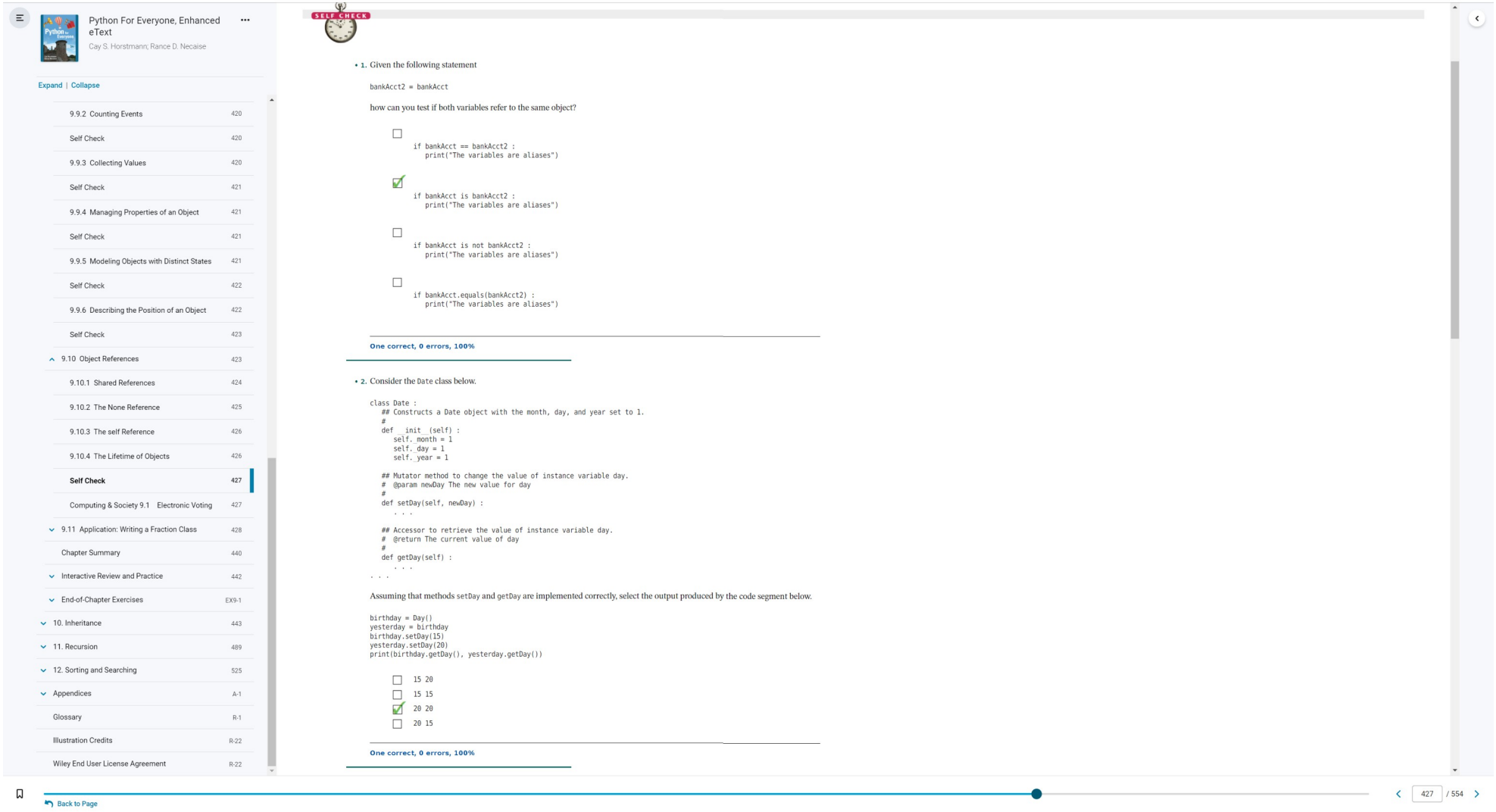
```
5 3903
Expected: 5 3903
3 3910
Expected: 3 3910
```

Score

2/2

Back to Page

419 / 554


Python For Everyone, Enhanced eText
Cay S. Horstmann, Rance D. Necaise
Expand | Collapse
9.9.2 Counting Events 420
Self Check 420
9.9.3 Collecting Values 420
Self Check 421
9.9.4 Managing Properties of an Object 421
Self Check 421
9.9.5 Modeling Objects with Distinct States 421
Self Check 422
9.9.6 Describing the Position of an Object 422
Self Check 423
9.10 Object References 423
9.10.1 Shared References 424
9.10.2 The None Reference 425
9.10.3 The self Reference 426
9.10.4 The Lifetime of Objects 426
Self Check 427
Computing & Society 9.1 Electronic Voting 427
9.11 Application: Writing a Fraction Class 428
Chapter Summary 440
Interactive Review and Practice 442
End-of-Chapter Exercises EX9-1
10. Inheritance 443
11. Recursion 489
12. Sorting and Searching 525
Appendices A-1
Glossary R-1
Illustration Credits R-22
Wiley End User License Agreement R-22
SELF CHECK
• 1. Given the following statement
bankAcct2 = bankAcct
how can you test if both variables refer to the same object?
if bankAcct == bankAcct2 :
print("The variables are aliases")
if bankAcct is bankAcct2 :
print("The variables are aliases")
if bankAcct is not bankAcct2 :
print("The variables are aliases")
if bankAcct.equals(bankAcct2) :
print("The variables are aliases")
One correct, 0 errors, 100%
• 2. Consider the Date class below.
class Date :
Constructs a Date object with the month, day, and year set to 1.
#
def __init__(self) :
self. month = 1
self. day = 1
self. year = 1
Mutator method to change the value of instance variable day.
@param newDay The new value for day
#
def setDay(self, newDay) :
. . .
Accessor to retrieve the value of instance variable day.
@return The current value of day
#
def getDay(self) :
. . .
. . .
Assuming that methods setDay and getDay are implemented correctly, select the output produced by the code segment below.
birthday = Day()
yesterday = birthday
birthday.setDay(15)
yesterday.setDay(20)
print(birthday.getDay(), yesterday.getDay())
15 20
15 15
20 20
20 15
One correct, 0 errors, 100%
Back to Page
427 / 554

9.10

•• 3. Walk through the following code that demonstrates how copying numbers is different from copying object references.

GOOD JOB! ✓

```
def main() :
   balance1 = 1000
   balance2 = balance1
   balance2 = balance2 - 100
   print(balance1)
   acct1 = BankAccount(1000)
   acct2 = acct1
   acct2.withdraw(100)
   print(acct1.getBalance())

class BankAccount :
   def __init__(self, initialBalance) :
      self._balance = initialBalance

   def deposit(self, amount) :
      self._balance = self._balance + amount

   def withdraw(self, amount) :
      self._balance = self._balance - amount

   def getBalance(self) :
      return self._balance

main()
```

balance1	balance2	acct1._balance	Output
1000	~~1000~~		
	900		~~1000~~
		~~1000~~	
		900	900

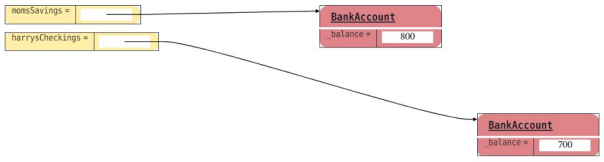
11 correct, 0 errors, 100%, 57 seconds

Start over

••• 4. What happens when the following code is executed? Pay particular attention to the `self` reference.

```
def main() :
   momsSavings = BankAccount(1000)
   harrysChecking = BankAccount(500)
   momsSavings.transfer(200, harrysChecking)
   . . .

class BankAccount :
   def __init__(self, initialBalance) :
      self._balance = initialBalance

   def deposit(self 13 14 , amount 15 16 ) :
      self._balance = self._balance + amount 17 18

   def withdraw(self 7 8 , amount 9 10 ) :
      self._balance = self._balance - amount 11 12

   def transfer(self 1 2 , amount 3 4 , toAcct 5 6 ) :
      self.withdraw(amount)
      toAcct.deposit(amount) 19
```

GOOD JOB! ✓

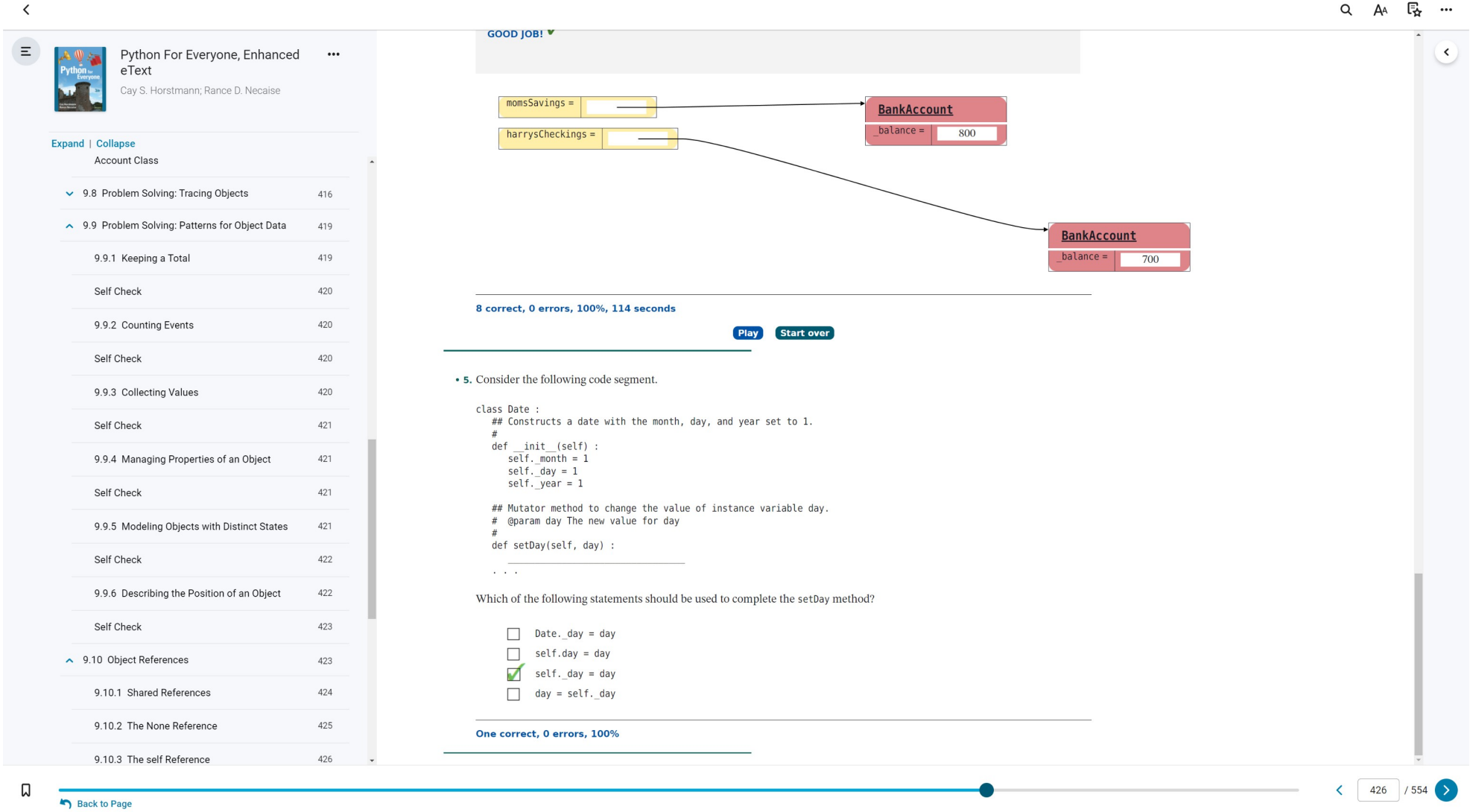


8 correct, 0 errors, 100%, 114 seconds

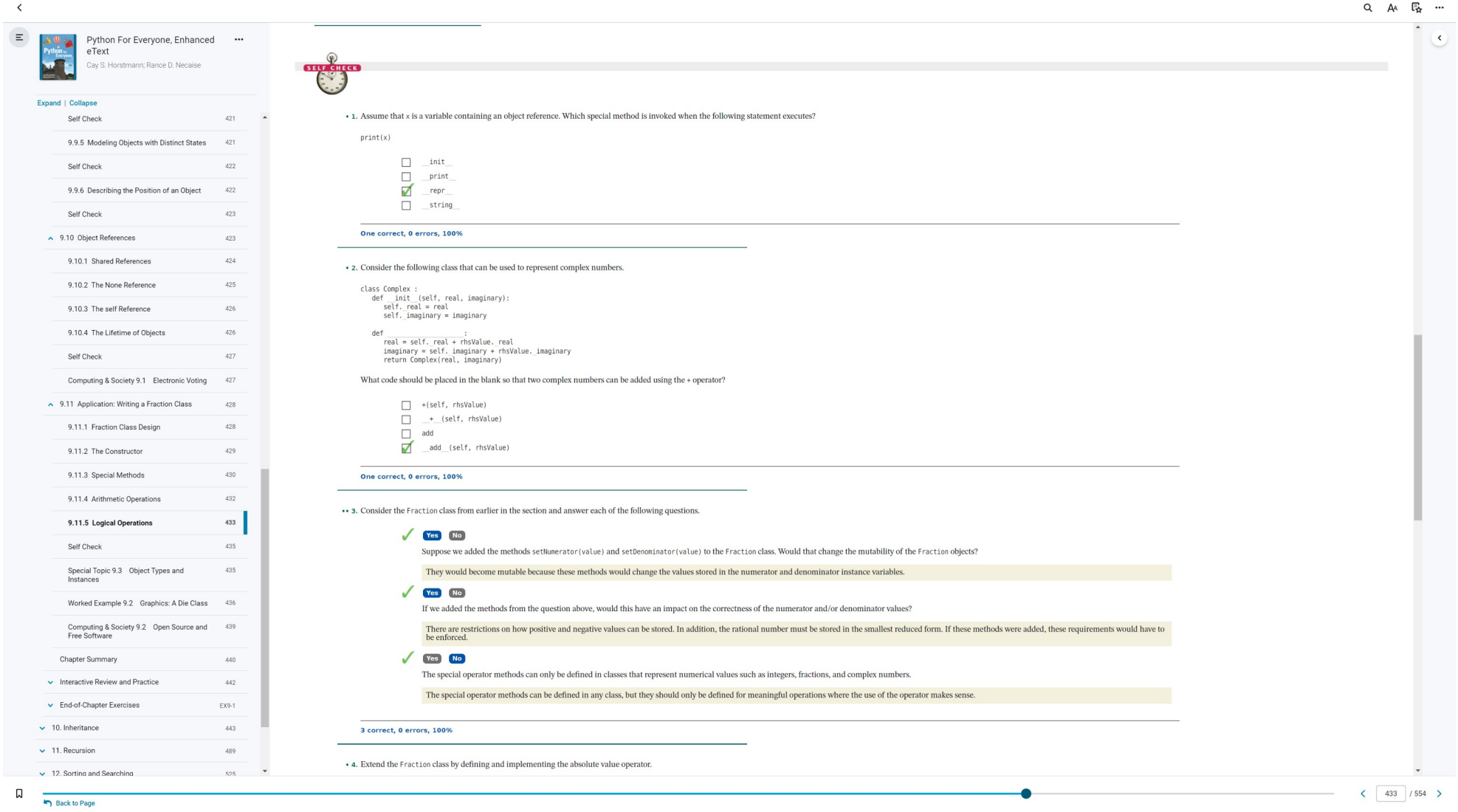
Back to Page

426 / 554

9.10


Python For Everyone, Enhanced eText
Cay S. Horstmann; Rance D. Necaise
Expand | Collapse
Account Class
9.8 Problem Solving: Tracing Objects 416
9.9 Problem Solving: Patterns for Object Data 419
9.9.1 Keeping a Total 419
Self Check 420
9.9.2 Counting Events 420
Self Check 420
9.9.3 Collecting Values 420
Self Check 421
9.9.4 Managing Properties of an Object 421
Self Check 421
9.9.5 Modeling Objects with Distinct States 421
Self Check 422
9.9.6 Describing the Position of an Object 422
Self Check 423
9.10 Object References 423
9.10.1 Shared References 424
9.10.2 The None Reference 425
9.10.3 The self Reference 426
GOOD JOB!
momsSavings =
harrysCheckings =
BankAccount
_balance = 800
BankAccount
_balance = 700
8 correct, 0 errors, 100%, 114 seconds
Play
Start over
• 5. Consider the following code segment.
class Date :
Constructs a date with the month, day, and year set to 1.
#
def __init__(self) :
self._month = 1
self._day = 1
self._year = 1
Mutator method to change the value of instance variable day.
@param day The new value for day
#
def setDay(self, day) :
. . .
Which of the following statements should be used to complete the setDay method?
Date._day = day
self.day = day
self._day = day
day = self._day
One correct, 0 errors, 100%
Back to Page
426 / 554

9.10



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

Expand | Collapse

SELF CHECK

• 1. Assume that x is a variable containing an object reference. Which special method is invoked when the following statement executes?

```
print(x)
```

- [ ] `__init__`
- [ ] `__print__`
- [x] `__repr__`
- [ ] `__string__`

One correct, 0 errors, 100%

• 2. Consider the following class that can be used to represent complex numbers.

```
class Complex :
   def __init__(self, real, imaginary):
      self.real = real
      self.imaginary = imaginary

   def ____________________:
      real = self.real + rhsValue.real
      imaginary = self.imaginary + rhsValue.imaginary
      return Complex(real, imaginary)
```

What code should be placed in the blank so that two complex numbers can be added using the + operator?

- [ ] `+(self, rhsValue)`
- [ ] `__+__(self, rhsValue)`
- [ ] `add`
- [x] `__add__(self, rhsValue)`

One correct, 0 errors, 100%

•• 3. Consider the `Fraction` class from earlier in the section and answer each of the following questions.

✓ Yes No

Suppose we added the methods `setNumerator(value)` and `setDenominator(value)` to the `Fraction` class. Would that change the mutability of the `Fraction` objects?

They would become mutable because these methods would change the values stored in the numerator and denominator instance variables.

✓ Yes No

If we added the methods from the question above, would this have an impact on the correctness of the numerator and/or denominator values?

There are restrictions on how positive and negative values can be stored. In addition, the rational number must be stored in the smallest reduced form. If these methods were added, these requirements would have to be enforced.

✓ Yes No

The special operator methods can only be defined in classes that represent numerical values such as integers, fractions, and complex numbers.

The special operator methods can be defined in any class, but they should only be defined for meaningful operations where the use of the operator makes sense.

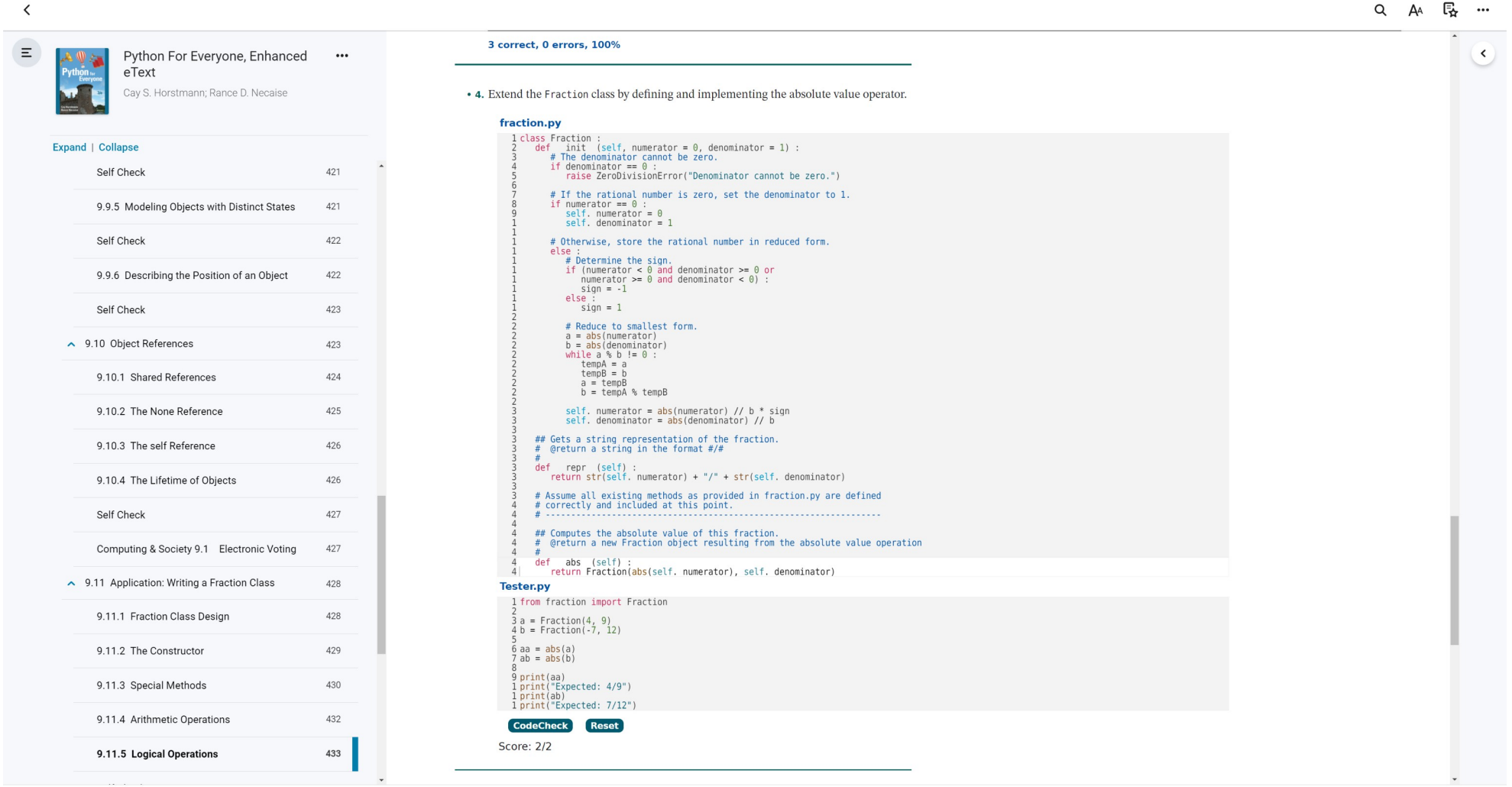
3 correct, 0 errors, 100%

• 4. Extend the `Fraction` class by defining and implementing the absolute value operator.

Back to Page

433 / 554

9.11



3 correct, 0 errors, 100%

• **4.** Extend the `Fraction` class by defining and implementing the absolute value operator.

fraction.py

```
class Fraction :
   def __init__(self, numerator = 0, denominator = 1) :
      # The denominator cannot be zero.
      if denominator == 0 :
         raise ZeroDivisionError("Denominator cannot be zero.")

      # If the rational number is zero, set the denominator to 1.
      if numerator == 0 :
         self._numerator = 0
         self._denominator = 1

      # Otherwise, store the rational number in reduced form.
      else :
         # Determine the sign.
         if (numerator < 0 and denominator >= 0 or
             numerator >= 0 and denominator < 0) :
            sign = -1
         else :
            sign = 1

         # Reduce to smallest form.
         a = abs(numerator)
         b = abs(denominator)
         while a % b != 0 :
            tempA = a
            tempB = b
            a = tempB
            b = tempA % tempB

         self._numerator = abs(numerator) // b * sign
         self._denominator = abs(denominator) // b

   ## Gets a string representation of the fraction.
   #  @return a string in the format #/#
   #
   def __repr__(self) :
      return str(self._numerator) + "/" + str(self._denominator)

   # Assume all existing methods as provided in fraction.py are defined
   # correctly and included at this point.
   # ---------------------------------------------------------------------

   ## Computes the absolute value of this fraction.
   #  @return a new Fraction object resulting from the absolute value operation
   #
   def __abs__(self) :
      return Fraction(abs(self._numerator), self._denominator)
```

Tester.py

```
from fraction import Fraction

a = Fraction(4, 9)
b = Fraction(-7, 12)

aa = abs(a)
ab = abs(b)

print(aa)
print("Expected: 4/9")
print(ab)
print("Expected: 7/12")
```

CodeCheck Reset

Score: 2/2

9.11

Python For Everyone, Enhanced eText

Cay S. Horstmann; Rance D. Necaise

• **5.** Extend the `Counter` class by implementing the the string conversion and equality operator special methods.

counter.py

```
##
#  Models a tally counter.
#
class Counter :
   ##
   #  Constructs a tally counter set to 0.
   #
   def __init__(self) :
      self.value = 0

   ##
   #  Gets the current value of this counter.
   #  @return the current value
   #
   def getValue(self) :
      return self.value

   ##
   #  Advances the value of this counter by 1.
   #
   def click(self) :
      self.value = self.value + 1

   ##
   #  Resets the value of this counter to 0.
   #
   def reset(self) :
      self.value = 0

   ##
   #  Determines if this tally counter has the same value as another counter.
   #  @param other the other tally counter being compared
   #  @return True if the two tally counters contain the same value
   #
   def __eq__(self, other) :
      if type(self) == type(other) :
         return self.value == other.value
      return False

   ##
   #  Gets a string representation of the tally counter.
   #  @return a string containing the numerical value of the counter
   #
   def __repr__(self) :
      return str(self.value)
```

Tester.py

```
from counter import Counter

tally = Counter()
tally.click()
tally.click()
tally.click()

tally2 = Counter()
tally2.click()
tally2.click()

print(tally == tally2)
print("Expected: False")

tally2.click()
print(tally == tally2)
print("Expected: True")

print(tally)
print("Expected: 3")
```

CodeCheck Reset

Score: 3/3