Syntax: Instance Methods

- Use instance variables inside methods of the class
  - Similar to the constructor, all other instance methods must include the self parameter as the first parameter
  - You must specify the `self` implicit parameter when using instance variables inside the class

Syntax

```
class ClassName :
   . . .
   def methodName(self, parameterName1, parameterName2, . . .) :
      method body
   . . .
```

```
class CashRegister :
   . . .
   def addItem(self, price) :
      self._itemCount = self._itemCount + 1
      self._totalPrice = self._totalPrice + price
   . . .
```

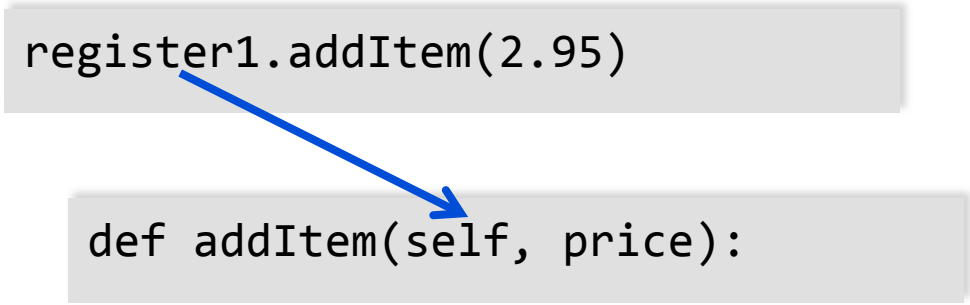
Every method must include the special `self` parameter variable. It is automatically assigned a value when the method is called.

Instance variables are referenced using the `self` parameter.

Local variable

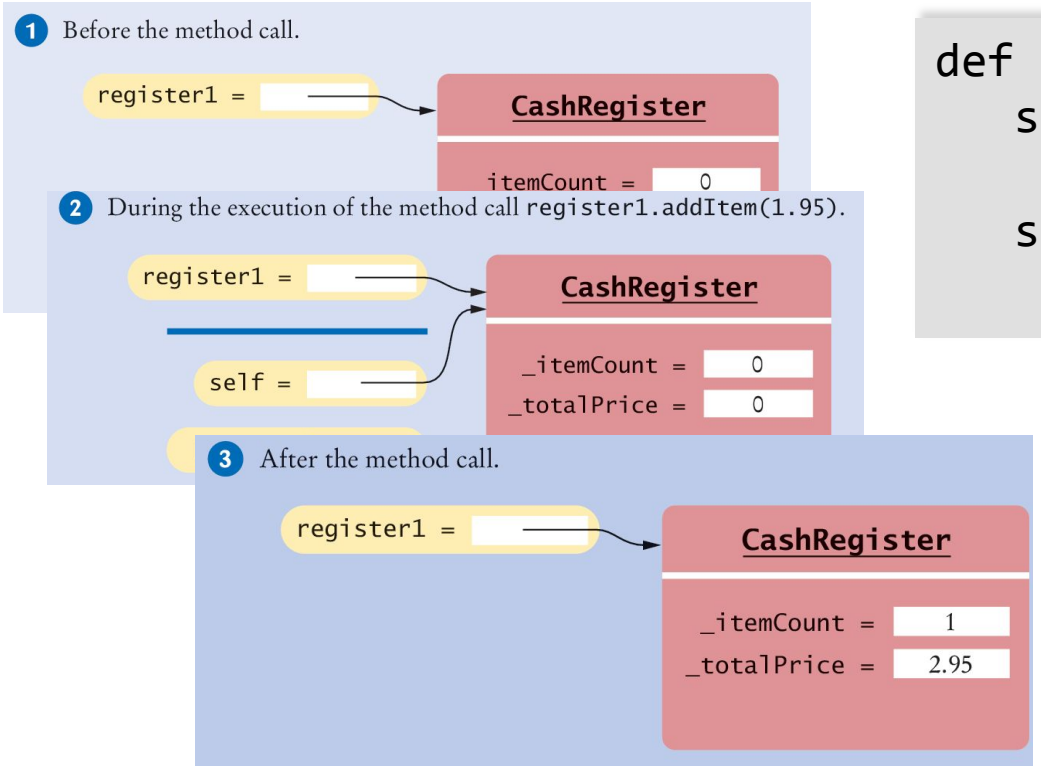
Invoking Instance Methods

- As with the constructor, every method must include the special `self` parameter variable, and it must be listed first.
- When a method is called, a reference to the object on which the method was invoked (`register1`) is automatically passed to the `self` parameter variable:



Tracing The Method Call

```
register1 = CashRegister()    #1 New object
register1.addItem(2.95)       #2  Calling method
…                             #3 After method
```




```
def addItem(self, price):
    self._itemCount =
        self._itemCount + 1
    self._totalPrice =
        self._totalPrice + price
```

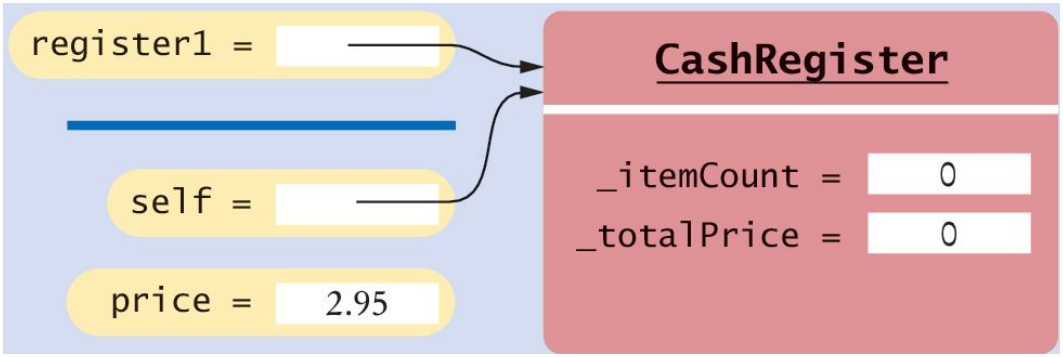
Accessing Instance Variables

- To access an instance variable, such as `_itemCount` or `_totalPrice`, in a method, you must access the variable name through the `self` reference
  - This indicates that you want to access the instance variables of the object on which the method is invoked, and not those of some other `CashRegister` object
- The first statement in the `addItem()` method is

```
self._itemCount = self._itemCount + 1
```

Accessing Instance Variables

- Which `_itemCount` is incremented?
  - In this call, it is the `_itemCount` of the `register1` object.



Calling One Method Within Another

- When one method needs to call another method on the same object, you invoke the method on the self parameter

```
def addItems(self, quantity, price) :
    for i in range(quantity) :
        self.addItem(price)
```

Example: CashRegister.py (1)

```
 7  class CashRegister :
 8     ## Constructs a cash register with cleared item count and total.
 9     #
10     def __init__(self) :
11        self._itemCount = 0
12        self._totalPrice = 0.0
13
14     ## Adds an item to this cash register.
15     #  @param price the price of this item
16     #
17     def addItem(self, price) :
18        self._itemCount = self._itemCount + 1
19        self._totalPrice = self._totalPrice + price
20
21     ## Gets the price of all items in the current sale.
22     #  @return the total price
23     #
24     def getTotal(self) :
25        return self._totalPrice
```

```
27     ## Gets the number of items in the current sale.
28     #  @return the item count
29     #
30     def getCount(self) :
31        return self._itemCount
32
33     ## Clears the item count and the total.
34     #
35     def clear(self) :
36        self._itemCount = 0
37        self._totalPrice = 0.0
```

Programming Tip 9.2

- Instance variables should only be defined in the constructor
- All variables, including instance variables, are created at run time
  - There is nothing to prevent you from creating instance variables in any method of a class
- The constructor is invoked before any method can be called, so any instance variables that were created in the constructor are sure to be available in all methods

Class Variables

- They are a value properly belongs to a class, not to any object of the class
- Class variables are often called “static variables”
- Class variables are declared at the same level as methods
  - In contrast, instance variables are created in the constructor

Class Variables: Example (1)

- We want to assign bank account numbers sequentially: the first account is assigned number 1001, the next with number 1002, and so on
- To solve this problem, we need to have a single value of `_lastAssignedNumber` that is a property of the *class*, not any object of the class

```
class BankAccount :
    _lastAssignedNumber = 1000 # A class variable
    def _ _init_ _(self) :
        self._balance = 0
        BankAccount._lastAssignedNumber =
           BankAccount._lastAssignedNumber + 1
        self._accountNumber =
           BankAccount._lastAssignedNumber
```

Class Variables: Example (2)

- Every `BankAccount` object has its own `_balance` and `_accountNumber` instance variables, but there is only a single copy of the `_lastAssignedNumber` variable
- That variable is stored in a separate location, outside any `BankAccount` object
- Like instance variables, class variables should always be private to ensure that methods of other classes do not change their values. However, class *constants* can be public

Class Variables: Example (3)

- For example, the `BankAccount` class can define a public constant value, such as

```
class BankAccount :
    OVERDRAFT_FEE = 29.95
    . . .
```

- Methods from any class can refer to such a constant as `BankAccount.OVERDRAFT_FEE`

Testing a Class

- In the long run, your class may become a part of a larger program that interacts with users, stores data in files, and so on
- You should always test your class in isolation integrating a class into a program
- Testing in isolation, outside a complete program, is called **unit testing**

Choices for Testing: The Python shell

- Some interactive development environments provide access to the Python shell in which individual statements can be executed
- You can test a class simply by constructing an object, calling methods, and verifying that you get the expected return values

```
>>> from cashregister import CashRegister
>>> reg = CashRegister()
>>> reg.addItem(1.95)
>>> reg.addItem(0.95)
>>> reg.addItem(2.50)
>>> print(reg.getCount())
3
>>> print(reg.getTotal())
5.4
>>>
```

Choices for Testing: Test Drivers

- Interactive testing is quick and convenient but it has a drawback
  - When you find and fix a mistake, you need to type in the tests again
- As your classes get more complex, you should write tester programs
  - A tester program is a driver module that imports the class and contains statements to run methods of your class

Steps Performed by a Tester Program

1. Construct one or more objects of the class that is being tested
2. Invoke one or more methods
3. Print out one or more results
4. Print the expected results
5. Compare the computed results with the expected

Example Test Program

- It runs and tests the methods of the `CashRegister` class

```
 5  from cashregister import CashRegister
 6
 7  register1 = CashRegister()
 8  register1.addItem(1.95)
 9  register1.addItem(0.95)
10  register1.addItem(2.50)
11  print(register1.getCount())
12  print("Expected: 3")
13  print("%.2f" % register1.getTotal())
14  print("Expected: 5.40")
```

Program execution

```
3
Expected: 3
5.40
Expected: 5.40
```