Chapter 10

INHERITANCE

Chapter Goals

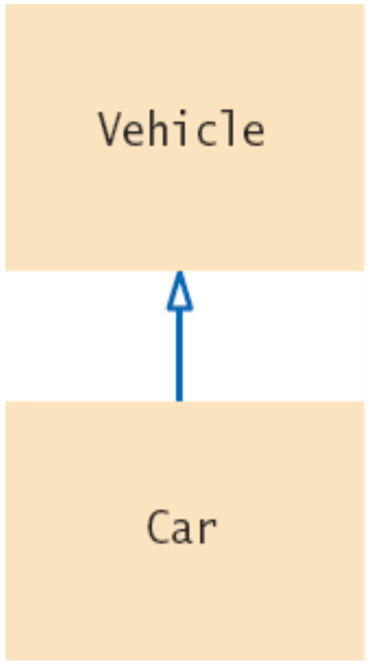
- To learn about inheritance
- To implement subclasses that inherit and override superclass methods
- To understand the concept of polymorphism

In this chapter, you will learn how the notion of inheritance expresses the relationship between specialized and general classes.

Contents

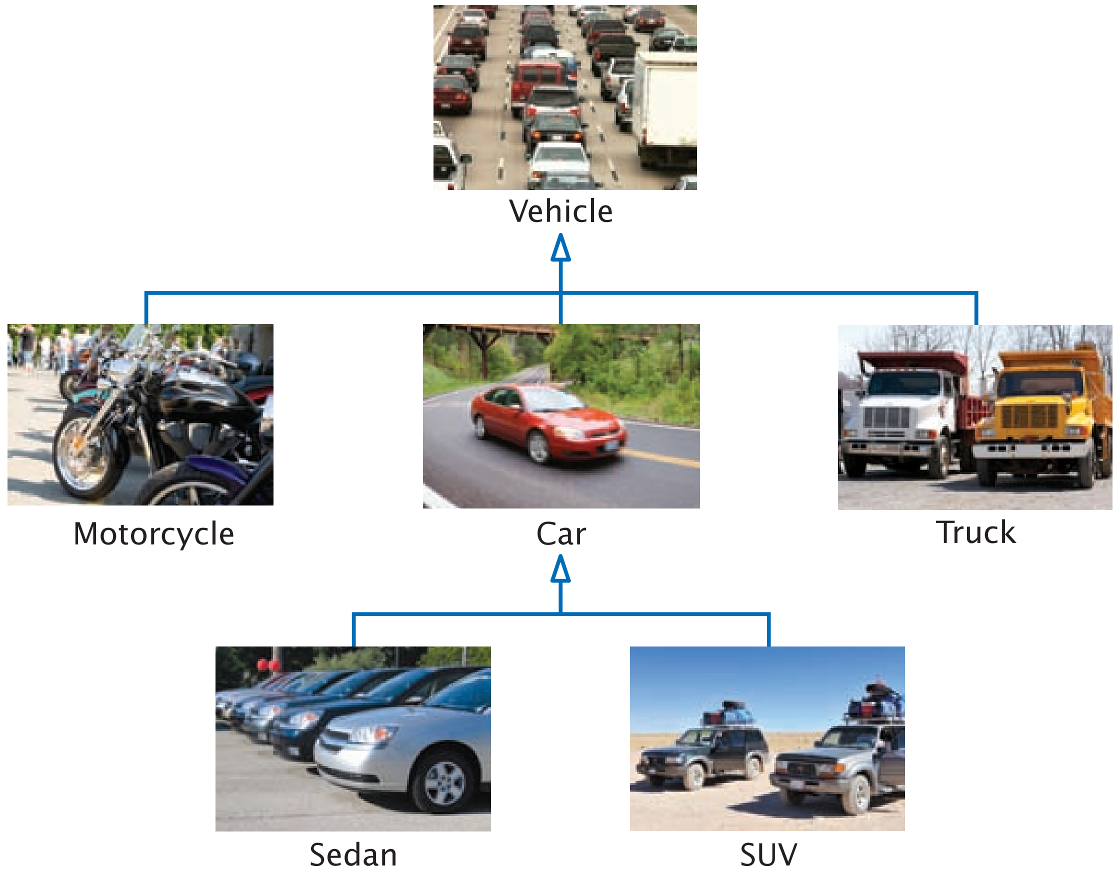
Inheritance Hierarchies

- In object-oriented programming, inheritance is a relationship between:
  - A *superclass*: a more generalized class
  - A *subclass*: a more specialized class
- The subclass ‘inherits’ data (variables) and behavior (methods) from the superclass



A Vehicle Class Hierarchy

- General
- Specialized
- More Specific

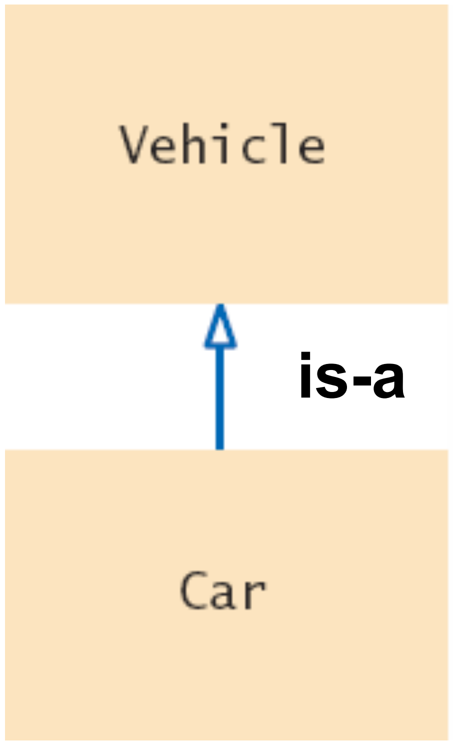


The Substitution Principle

- Since the subclass Car "**is-a**" Vehicle
  - Car shares common traits with Vehicle
  - You can substitute a Car object in an algorithm that expects a Vehicle object

```
myCar = Car(. . .)
processVehicle(myCar)
```

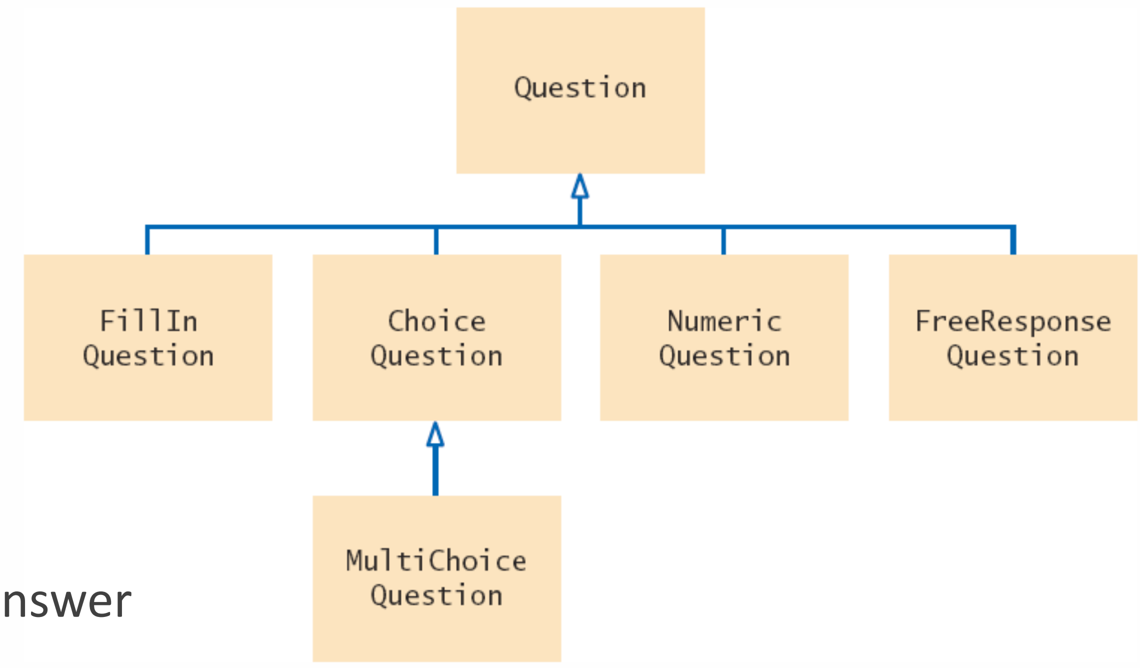
The 'is-a' relationship is represented by an arrow in a class diagram and means that the subclass can behave as an object of the superclass.



Quiz Question Hierarchy

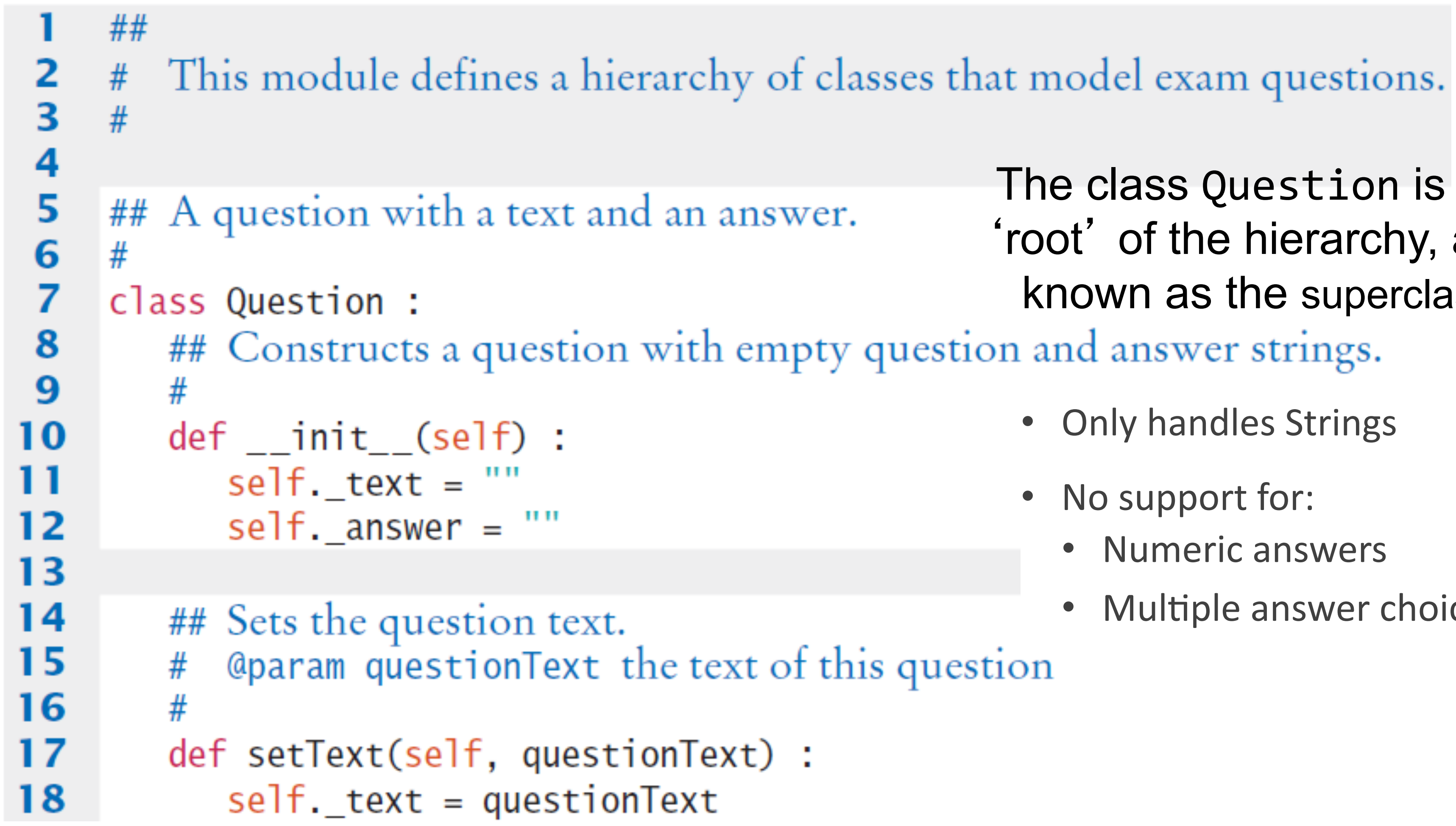
- There are different types of quiz questions:
  1) Fill-in-the-blank
  2) Single answer choice
  3) Multiple answer choice
  4) Numeric answer
  5) Free Response

The ‘root’ of the hierarchy is shown at the top.



- A question can:
  - Display its text
  - Check for correct answer

Questions.py

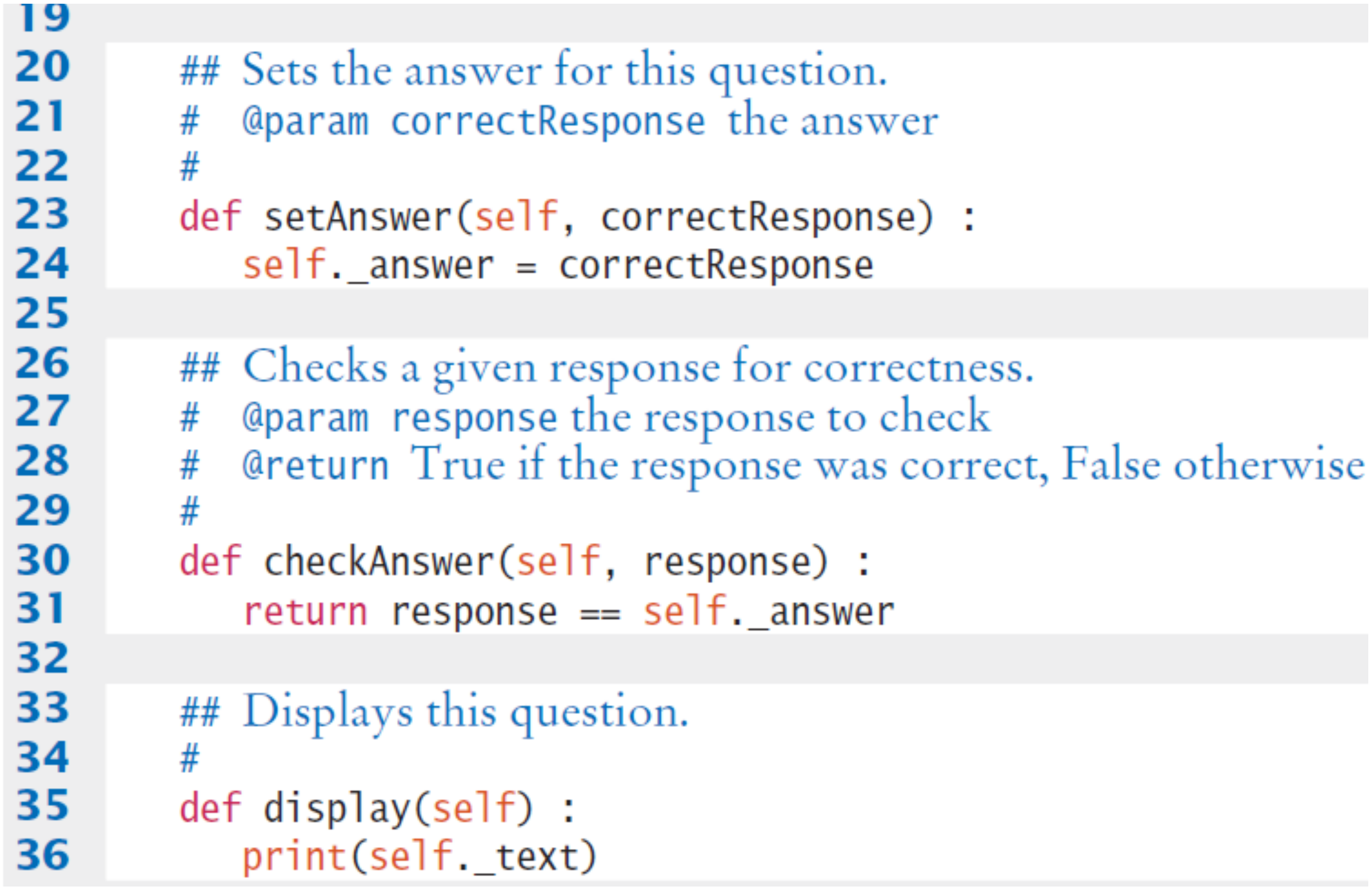


```
##
#  This module defines a hierarchy of classes that model exam questions.
#

## A question with a text and an answer.
#
class Question :
    ## Constructs a question with empty question and answer strings.
    #
    def __init__(self) :
        self._text = ""
        self._answer = ""

    ## Sets the question text.
    #  @param questionText the text of this question
    #
    def setText(self, questionText) :
        self._text = questionText
```

The class `Question` is the 'root' of the hierarchy, also known as the superclass

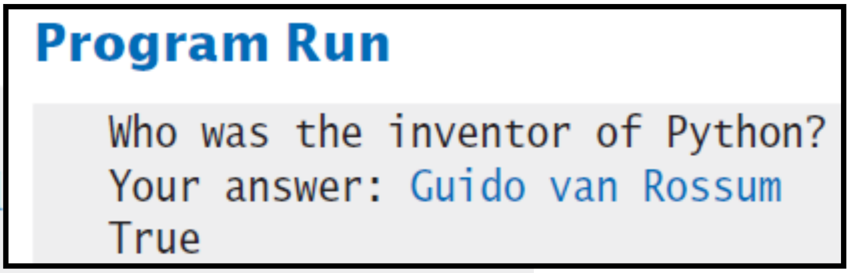
- Only handles Strings
- No support for:
  - Numeric answers
  - Multiple answer choice

Questions.py



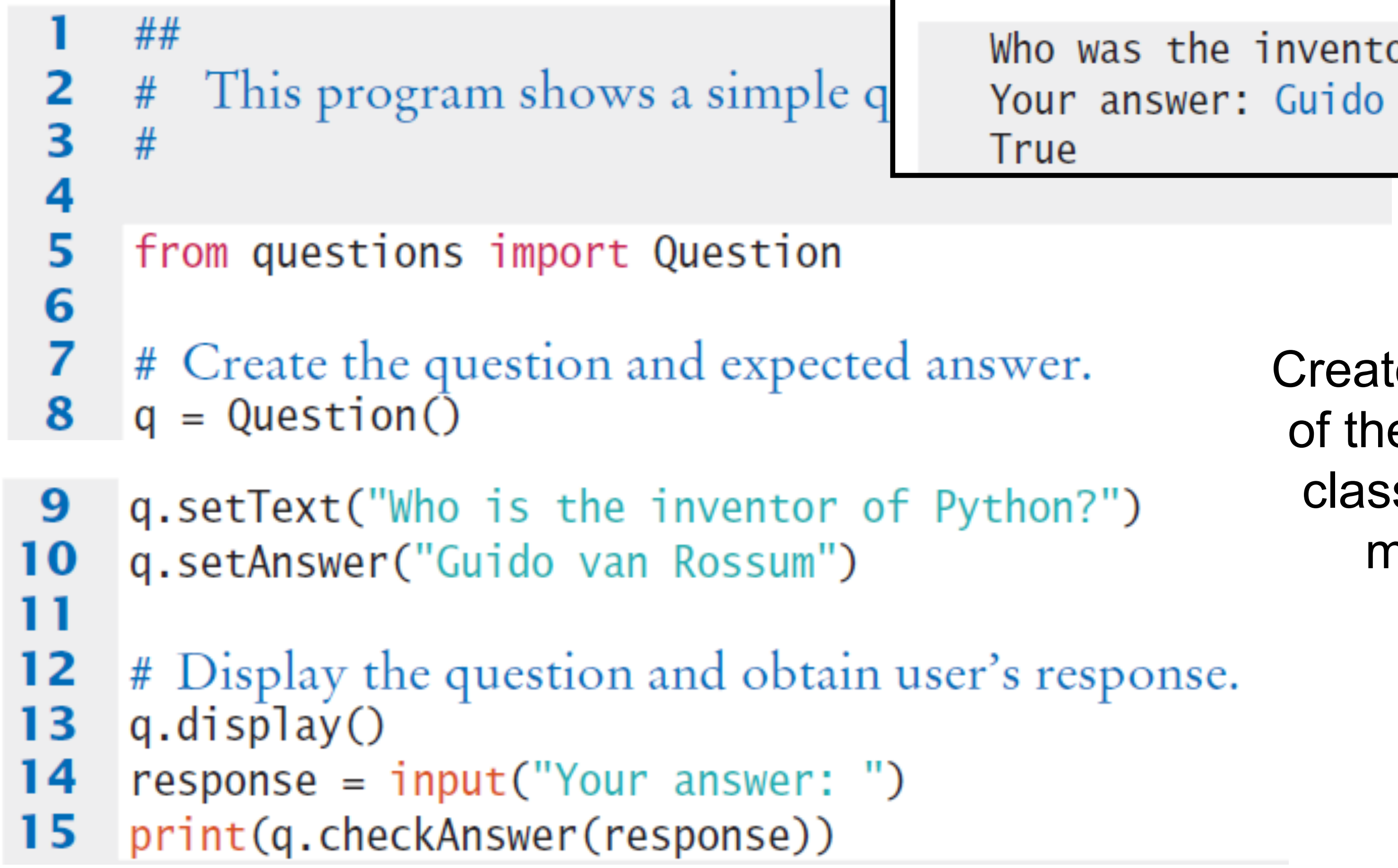
```

    ## Sets the answer for this question.
    #  @param correctResponse the answer
    #
    def setAnswer(self, correctResponse) :
        self._answer = correctResponse

    ## Checks a given response for correctness.
    #  @param response the response to check
    #  @return True if the response was correct, False otherwise
    #
    def checkAnswer(self, response) :
        return response == self._answer

    ## Displays this question.
    #
    def display(self) :
        print(self._text)
```

Questions.py



Program Run

```
Who was the inventor of Python?
Your answer: Guido van Rossum
True
```



```python
##
#  This program shows a simple q
#

from questions import Question

# Create the question and expected answer.
q = Question()
q.setText("Who is the inventor of Python?")
q.setAnswer("Guido van Rossum")

# Display the question and obtain user's response.
q.display()
response = input("Your answer: ")
print(q.checkAnswer(response))
```

Creates an object of the `Question` class and uses methods.

Programming Tip

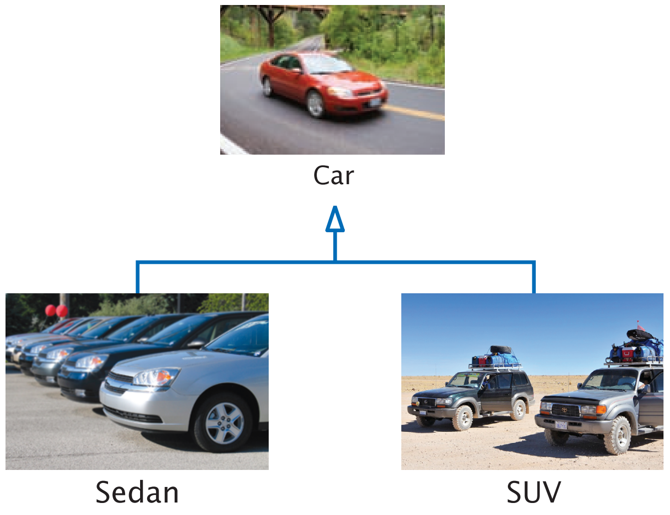
- Use a Single Class for Variation in *Values*, Inheritance for Variation in *Behavior*
  - If two vehicles only vary by fuel efficiency, use an instance variable for the variation, not inheritance

```
# Car instance variable
milesPerGallon
```

  - If two vehicles behave differently,

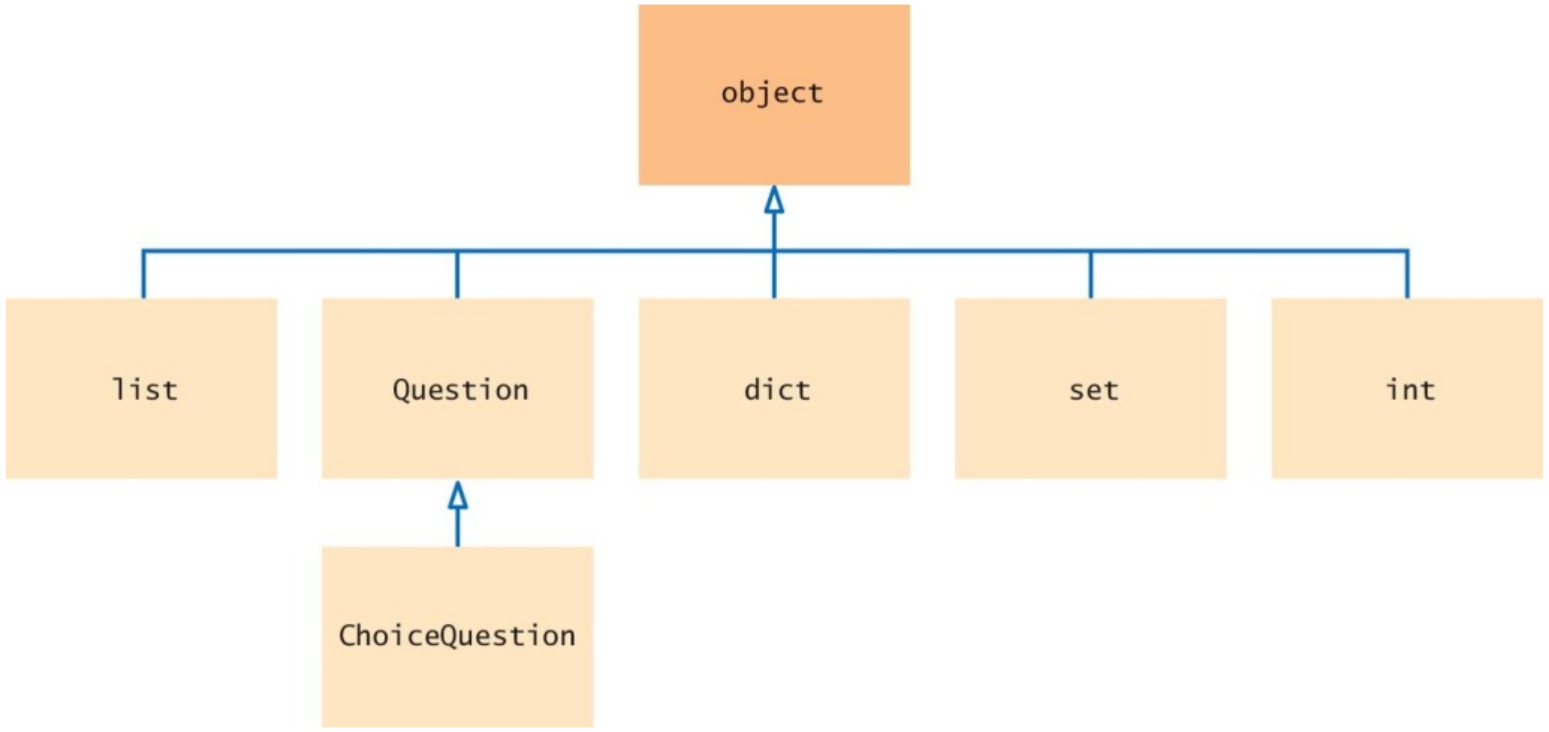
  use inheritance

Be careful not to over-use inheritance



The Cosmic Superclass: `object`

- In Python, every class that is declared without an explicit superclass automatically extends the class `object`



Implementing Subclasses

- Consider implementing `ChoiceQuestion` to handle:

```
In which country was the inventor of Python born?
1. Australia
2. Canada
3. Netherlands
4. United States
```

- How does `ChoiceQuestion` differ from `Question`?
  - It stores choices (1,2,3 and 4) in addition to the question
  - There must be a method for adding multiple choices
    - The `display()` method will show these choices below the question, numbered appropriately

In this section you will see how to form a subclass and how a subclass automatically inherits from its superclass

Inheriting from the Superclass

- Subclasses inherit from the superclass:
  - All methods that it does not override
  - All instance variables
- The Subclass can
  - Add new instance variables
  - Add new methods
  - Change the implementation of inherited methods

Form a subclass by specifying what is different from the superclass.

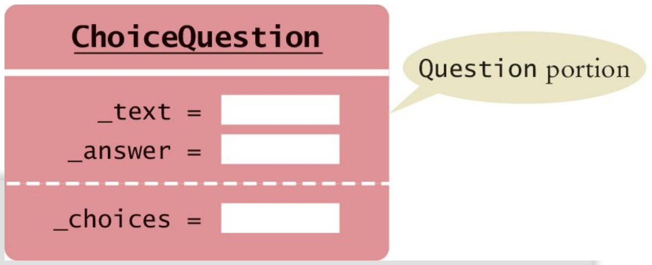
Overriding Superclass Methods

- Can you re-use any methods of the `Question` class?
  - Inherited methods perform exactly the same
  - If you need to change how a method works:
    - Write a new more specialized method in the subclass
    - Use the same method name as the superclass method you want to replace
    - It must take all of the same parameters
  - This will ***override*** the superclass method
- The new method will be invoked with the same method name when it is called on a subclass object

A subclass can override a method of the superclass by providing a new implementation.

Planning the Subclass

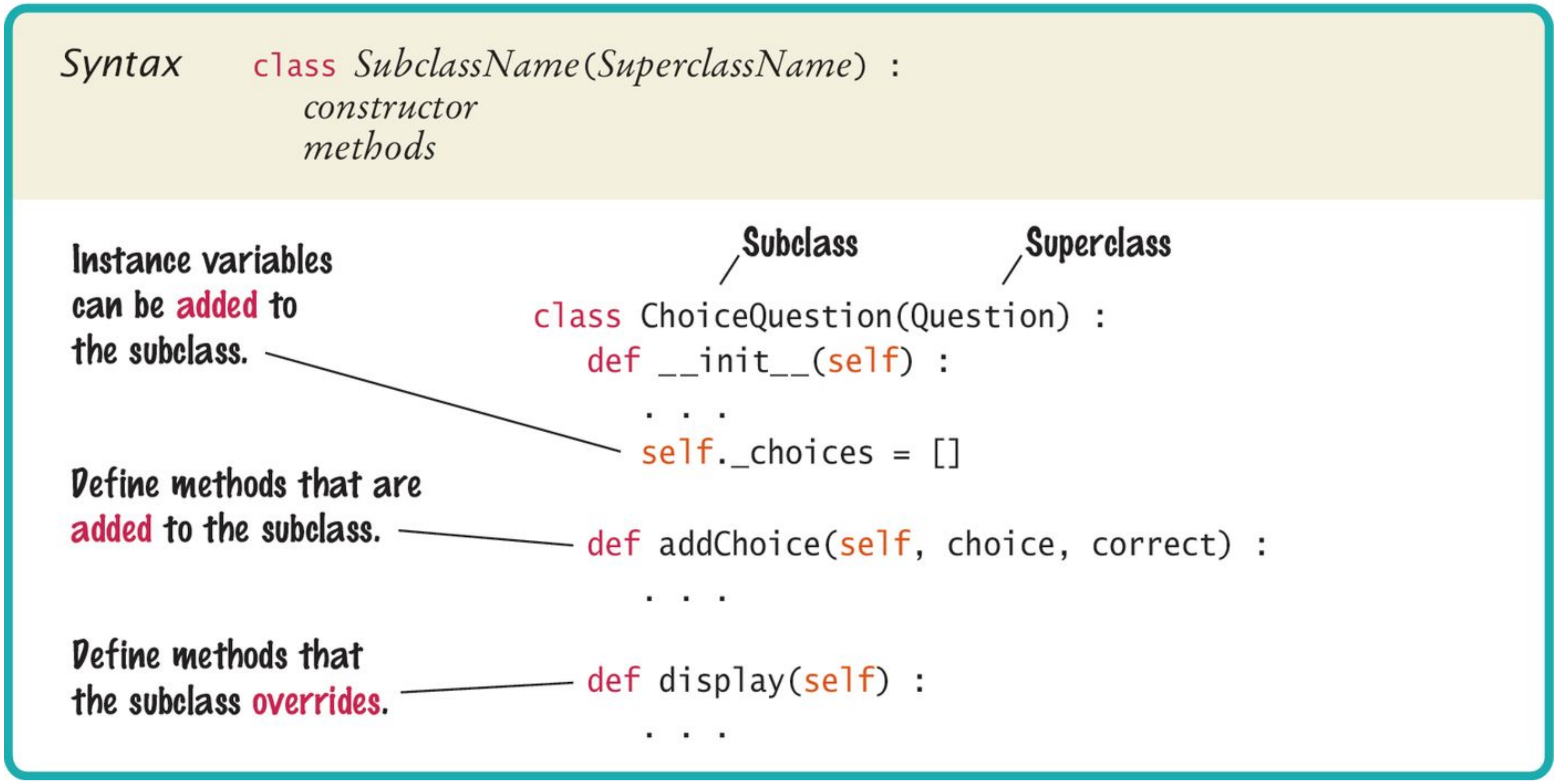
- Pass the name of the superclass Question as part of the definition of the subclass
  - Inherits text and answer variables
  - Add new instance variable choices



```
class ChoiceQuestion(Question):
    # The subclass has its own constructor.
    def _ _init_ _(self) :
        . . .
        # This instance variable is added to the subclass.
        self._choices = []

    # This method is added to the subclass
    def addChoice(self, choice, correct) :
        . . .
    # This method overrides a method from the superclass
    def void display(self) :
        . . .
```

Syntax 10.1: Subclass Definition

- The class name inside parentheses in the class header denotes inheritance.



Syntax

```
class SubclassName(SuperclassName) :
    constructor
    methods
```

```
class ChoiceQuestion(Question) :
   def __init__(self) :
      . . .
      self._choices = []

   def addChoice(self, choice, correct) :
      . . .

   def display(self) :
      . . .
```

Subclass
Superclass
Instance variables can be added to the subclass.
Define methods that are added to the subclass.
Define methods that the subclass overrides.