Inheriting Class

Self-Review Questions

Self-review 7.1 What do the terms ‘subclass’ and ‘superclass’ mean in object-oriented programming?

Self-review 7.2 What does the following declaration mean?

```python
class Foo ( Bar ) :
```

Self-review 7.3 What is the built-in name object in Python?

Self-review 7.4 What is the difference between single and multiple inheritance?

Self-review 7.5 What is the Diamond Problem in multiple inheritance?

Self-review 7.6 Define the terms:

2. Aggregation.

Self-review 7.7 In graphical user interfaces, what are:

1. Widgets.
2. Callbacks.
3. Events.
4. Event loops.
5. Event handler.

Self-review 7.8 What does the pack method do in Tkinter widgets?

Self-review 7.9 What is a modal dialog box?

Self-review 7.10 What is polymorphism? How is it related to object-oriented programming? Give a short example of a piece of code that exploits polymorphism.
Self-review 7.11 Using the expression evaluator from Section ?? (page ??), draw the object diagram that represents the following Python expression:

\[
\text{Mul} (3, \text{Div} (\text{Add} (1, 2), \text{Sub} (5, 8)))
\]

What does that expression evaluate to?

Self-review 7.12 What is duck typing?

Self-review 7.13 What does the name `self` mean? Is `self` a Python keyword?

Self-review 7.14 Explain what is happening in the following code:

```python
class Foo ( Bar ) :
    def __init__ ( self ) :
        Bar.__init__ ( self )
        return
```

Programming Exercises

Exercise 7.1 Extend the traffic light program to include a short red+amber ‘prepare to go’ state as is used for traffic lights in the UK, and other places.

Exercise 7.2 Create a subclasses of your Account class (from ??) called CreditAccount in which the user is charged a set amount for every withdrawal that is made. If the user is overdrawn, the withdrawal charge is doubled.

Exercise 7.3 Create a subclasses of your Account class (from ??) called StudentAccount in which new accounts start off with a balance of £500 and an overdraft of up to £3000 is allowed, with no charges for withdrawal.

Exercise 7.4 Create versions of:

1. `DrawableRectangle`
2. `DrawableSquare`
3. `DrawableCircle`

that work using the Tkinter package.

Exercise 7.5 Create a class called `Number` and two subclasses `Binary` and `Roman` for dealing with different sorts of number representation. The constructors to `Binary` and `Roman` should take a string argument so that you can use them like this:

```python
b = Binary ( '11010101' )
r = Roman ( 'MCMLXXVIII' )
```

In your `Number` class you need to have methods that are common to all types of number representation:
1. You’ll want a private method called __to_int that returns an ordinary integer representation of the calling object.

2. You’ll need to override the built-in method __str__ that returns a string representation of the number which the calling object represents.

3. You’ll need to override the __int__ method that is used by calls to the int function which converts its argument to an integer.

In all three classes you’ll need an __int__ method that calls the __to_int method in the calling object – even though Binary and Roman are subclasses of Number, they still need an __int__ method of their own. Why is this? If you’re not sure you have understood why this works, when you’ve finished this exercise, comment out the definition of __int__ in Binary and Roman and see what happens.

You’ll need to override __to_int in the Binary and Roman classes. For the method in Roman, you may want to use the algorithm in Section ?? to parse a Roman numeral into an integer. Make the LUT (lookup table) a class variable (rather than an instance variable).

For the Binary class, you’ll need to convert between binary and decimal. As a reminder of the algorithm, here’s an example:

\[ 1010101 \rightarrow 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]

When you have written all your classes you’ll need to test them. Below is the beginnings of a test program you can use to help you:

```python
def test () :
    data = [
        (Binary ('0'), 0), (Binary ('1'), 1),
        (Binary ('10'), 2), (Binary ('11'), 3),
        (Binary ('100'), 4), (Binary ('101'), 5),
        (Binary ('10101010101'), 1365),
        (Roman ('I'), 1), (Roman ('II'), 2),
        (Roman ('IV'), 4), (Roman ('VI'), 6),
        (Roman ('IX'), 9), (Roman ('X'), 10),
        (Roman ('XI'), 11), (Roman ('MM'), 2000),
        (Roman ('MCMLXXVIII'), 1978)
    ]

    for entry in data :
        assert int ( entry[0] ) == entry[1]

    if __name__ == '__main__':
        test ()
```

When your code works you should get no assertion failures when you execute the test program.

**Exercise 7.6** On page ?? we introduced the Diamond Problem of object-oriented programming languages and explained how Python deals with it. In this exercise you will explore how Python handles multiple inheritance.
Below are class definitions that exhibit the Diamond Problem. Start off by drawing a diagram to show the inheritance relationships between the five classes.

Create objects that are instances of classes D and E. Call the foobar and barfoo methods in both objects. What is the difference between the output from the two objects? Why does this happen?

```python
class A:
    def foobar(self):
        print('Class A')
    def barfoo(self):
        print('barfoo from class A!')
class B(A):
    def foobar(self):
        print('Class B inherits from A')
    def barfoo(self):
        print('barfoo from class B!')
class C(A):
    def foobar(self):
        print('Class C inherits from A')
    def barfoo(self):
        print('barfoo from class C!')
class D(B,C):
    def __init__(self):
        return

class E(C,B):
    def __init__(self):
        return
```

**Exercise 7.7** Write a program that takes a list of Point objects (from Section ??, page ??) and draws them with the Turtle module. Your code should be polymorphic. A Point2D object represents a point on the screen, but if you need to move the turtle to a point represented by a Point3D object, just ignore the z component.

Here is some test data to get you started:

```python
square = [
    Point2D(0.0, 0.0),
    Point2D(100.0, 0.0),
    Point2D(100.0, 100.0),
    Point2D(0.0, 100.0),
]
rectangle = [
    Point2D(0.0, 0.0),
    Point2D(200.0, 0.0),
    Point2D(200.0, 100.0),
    Point2D(0.0, 100.0),
]
pentagon = [
    Point2D(0.0, 0.0),
    Point2D(100.0, 0.0),
    Point2D(131.0, 95.0),
    Point2D(50.0, 154.0),
    Point2D(-30.0, 95.0),
]
square3 = [
    Point3D(0.0, 0.0, 1.0),
    Point3D(100.0, 0.0, 1.0),
    Point3D(100.0, 100.0, 1.0),
    Point3D(0.0, 100.0, 1.0),
]
square23 = [
    Point3D(0.0, 0.0, 1.0),
    Point2D(100.0, 0.0),
    Point2D(100.0, 100.0),
]
Point3D (0.0, 100.0, 1.0), Point2D (0.0, 0.0)

Hint: To draw these shapes elegantly, it would be sensible to lift the 'pen' of the turtle off the screen before moving from the origin to the first point.

Exercise 7.8 Write a version of the Caesar cipher from Section ?? (page ??) with a simple GUI. Allow users to enter text in a text box, then generate an enciphered version of the text when they press a button. Let users change the key that is used by the cipher either by providing a text box to enter the key, or a drop-down list of available keys.

You should research the HCI literature to see which of these two options the HCI community believe is a better choice of data entry in this context.

Exercise 7.9 Extend the Caesar cipher GUI by adding a button that causes deciphering (rather than enciphering) of the text in the text box.

Exercise 7.10 Write a simple color picker using Tkinter. You should provide the user with three sliders (using the Tkinter.Scale class) which they can use to set red, green and blue values. Whenever a slider is moved, a callback should be issued that changes the background color in a Canvas widget.

Exercise 7.11 Extend your answer to the previous exercise to allow users to also select specific colors from a drop-down list. Here is a random list of colors that you may want to include:

<table>
<thead>
<tr>
<th>RGB values</th>
<th>Color Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>255 255 255</td>
<td>white</td>
</tr>
<tr>
<td>0 0 0</td>
<td>black</td>
</tr>
<tr>
<td>47 79 79</td>
<td>dark slate gray</td>
</tr>
<tr>
<td>0 0 128</td>
<td>NavyBlue</td>
</tr>
<tr>
<td>30 144 255</td>
<td>dodger blue</td>
</tr>
<tr>
<td>70 130 180</td>
<td>SteelBlue</td>
</tr>
<tr>
<td>175 238 238</td>
<td>PaleTurquoise</td>
</tr>
<tr>
<td>85 107 47</td>
<td>DarkOliveGreen</td>
</tr>
<tr>
<td>124 252 0</td>
<td>LawnGreen</td>
</tr>
<tr>
<td>0 255 0</td>
<td>green</td>
</tr>
<tr>
<td>154 205 50</td>
<td>YellowGreen</td>
</tr>
<tr>
<td>240 230 140</td>
<td>khaki</td>
</tr>
<tr>
<td>255 255 224</td>
<td>LightYellow</td>
</tr>
<tr>
<td>255 255 0</td>
<td>yellow</td>
</tr>
</tbody>
</table>

The three numbers are the red, green, and blue (RGB) values used by a screen to generate the color of the associated name name.
Challenges

Challenge 7.1 Write a GUI for the expression evaluator in Section ???. Let the user enter an expression and press a button to have that expression evaluated. To make this a bit simpler, you might want to start out with a GUI where the user enters a Python expression such as this:

\[
\text{Add}(1, 2)
\]

You’ll need to turn the string from the user into a real Python object. Fortunately, Python provides a function called `eval` that does that. Here’s an example of `eval` in action:

```python
>>> class Foo:
...     def __init__(self):
...         return
...     def foo(self):
...         print('foobar!')
...
>>> eval('Foo().foo()')
foobar!
```

Note that you can only use `eval` with expressions, not with statements!

Challenge 7.2 This challenge is to create a GUI that allows you to apply various image processing algorithms to images. To do this you will need an image processing library. Python doesn’t come with one as standard, but you can download the Python Imaging Library (PIL) from http://www.pythonware.com/ – it is free. You will also need the PIL handbook http://www.pythonware.com/library/pil/handbook/, which is the best documentation available for the library.

The GUI should allow you to select an image, maybe using a list-box or a modal dialog from the tkFileDialog module. You will also need some buttons that apply transformations to the current image and display the processed image. To get you started, we have provided some code below that uses Tkinter to display an image, and some examples of image processing using PIL.

The **negative** of an image has the amount of red, green and blue in each image inverted. The negative of a red pixel (255, 0, 0) is a cyan pixel (0, 255, 255). The negative of a black pixel (0, 0, 0) is a white pixel (255, 255, 255). So to create the negative of an image, we just have to subtract the current red, green and blue values of each pixel from 255. Here’s an example image and the code that implements the transformation:
"""Produces the negative of a positive color image."""

```python
__author__ = 'Sarah Mount'
__date__ = '2006-09'
__version__ = '1.0'
__copyright__ = 'Copyright (c) 2006 Sarah Mount'
__licence__ = 'GNU General Public Licence (GPL)'

import Image, sys, viewport

if __name__ == '__main__':
    title = 'Color->Negative Using Python Image Library'
    filename = sys.argv[1]
    image = Image.open(filename, 'r')
    image.load()
    negative = image.point(lambda pixel: 255 - pixel)
    viewport.display_image(negative, title)
```

Two things to note here. Firstly, the PIL library stores the red, green and blue pixel values of an image as separate list elements (rather than storing a list of three-tuples). Secondly, each PIL image has a method called `point` that allows us to apply a transformation on each pixel value in the image. `point` can be used in two ways: you can either pass it a function or `lambda` expression to apply to each pixel value, or you can pass it a lookup table. Above we have used a `lambda` expression.

The PIL library also defines a number of `filters` that can be used to perform transformations on images. These are located in the `ImageFilter` module. To use a filter, you create an instance of a filter class and pass it to the `filter` method of an image object.

This example creates an embossed version of the image in which the edges of the objects in the image appear raised above the surface of the image:
Inheriting Class

"Emboss an image."

```python
__author__ = 'Sarah Mount'
__date__ = '2006-09'
__version__ = '1.0'
__copyright__ = 'Copyright (c) 2006 Sarah Mount'
__licence__ = 'GNU General Public Licence (GPL)'

import Image, ImageFilter, sys, viewport

if __name__ == '__main__':
    filename = sys.argv[1]
    image = Image.open(filename, 'r')
    image = image.filter(ImageFilter.EMBOSS())
    viewport.display_image(image, 'Image Embossed Using Python Image Library')

Below is the code for the viewport module that we have used in the examples above:

"View a PIL image on a Tkinter canvas."

```python
__author__ = 'Sarah Mount'
__date__ = '2006-09'
__version__ = '1.0'
__copyright__ = 'Copyright (c) 2006 Sarah Mount'
__licence__ = 'GNU General Public Licence (GPL)'

import Tkinter, Image, ImageTk, sys

def display_image(pil_image, title):
    """Take a PIL image and display it in a GUI."
    root = Tkinter.Tk()
    root.title(title)
    im_width, im_height = pil_image.getbbox()[2:4]
    canvas = Tkinter.Canvas(root, width=im_width, height=im_height)
    canvas.pack(side=Tkinter.LEFT, fill=Tkinter.BOTH, expand=1)
    photo = ImageTk.PhotoImage(pil_image)
    item = canvas.create_image(0, 0, anchor=Tkinter.NW, image=photo)
    Tkinter.mainloop()

if __name__ == '__main__':
    filename = sys.argv[1]
    image = Image.open(filename, 'r')
    display_image(image, 'Tk Viewport')
```
<table>
<thead>
<tr>
<th>Color</th>
<th>59</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emboss</td>
<td>61</td>
</tr>
<tr>
<td>Filter</td>
<td>61</td>
</tr>
<tr>
<td>Negative</td>
<td>60</td>
</tr>
<tr>
<td>PIL</td>
<td>60</td>
</tr>
<tr>
<td>Python Image Library</td>
<td>60</td>
</tr>
<tr>
<td>RGB</td>
<td>59</td>
</tr>
</tbody>
</table>