1 Exercises

Exercise 1. Reimplement the Circle data type from Problem 2 of Section 3.1, but this time representing a circle internally using the coordinates \((x, y)\) of the lower left corner of the square that inscribes the circle and has side length \(s\).

Exercise 2. Implement, along with a suitable test client, a comparable data type called Color, that represents a color in terms of its red, green, and blue components, and supports the following API:

<table>
<thead>
<tr>
<th>Color Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color(r = 0, g = 0, b = 0)</td>
<td>construct a color object (c) given its red, green, and blue components as integers from the interval ([0, 255])</td>
</tr>
<tr>
<td>c.getRed()</td>
<td>the red component of (c)</td>
</tr>
<tr>
<td>c.getGreen()</td>
<td>the green component of (c)</td>
</tr>
<tr>
<td>c.getBlue()</td>
<td>the blue component of (c)</td>
</tr>
<tr>
<td>c.luminosity()</td>
<td>the luminosity of (c) calculated as (0.299r + 0.587g + 0.114b)</td>
</tr>
<tr>
<td>(c + d)</td>
<td>a new color whose red, green, and blue components are the average values of the corresponding components of (c) and (d)</td>
</tr>
<tr>
<td>(c == d)</td>
<td>do (c) and (d) represent the same color?</td>
</tr>
<tr>
<td>cmp(c, d)</td>
<td>-1, 0, or 1 depending on whether (c)’s luminosity is less than, equal to, or greater than (d)’s luminosity</td>
</tr>
<tr>
<td>str(c)</td>
<td>string representation of (c) in ((r, g, b)) format</td>
</tr>
</tbody>
</table>

Exercise 3. Implement, along with a suitable test client, an iterable data type called RandomColors, that can be used to build and iterate over a collection of random Color objects. The data type must support the following API:

<table>
<thead>
<tr>
<th>RandomColors Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomColors(n)</td>
<td>an iterable object (r) for iterating over (n) random Color objects</td>
</tr>
<tr>
<td>iter(r)</td>
<td>an iterable object (riter) on (r)</td>
</tr>
<tr>
<td>next(riter)</td>
<td>the next random Color object from (riter)</td>
</tr>
</tbody>
</table>

Exercise 4. In the test client (_main_) in randomcolors.py, we sorted the list \(colors\) containing \(n\) random Color objects in the order of their luminosities (see definition \____cmp___.(self, other)\). How would you rewrite the statement \(colors.sort()\) to

a. sort the list in the order of the blue components of the colors?

b. sort the list in the order of the distance of the colors from black, ie, \((0, 0, 0)\)? If we have a color \(c = (r, g, b)\), we define its distance from black as \(r + g + b\).

2 Solutions to Exercises

Solution 1.

```python
class Circle:
    import math
    def __init__(self, h = 0.0, k = 0.0, r = 1.0):
        self._x = h - r
        self._y = k - r
        self._s = 2 * r
    def area(self):
        r = self._s / 2
        return math.pi * r ** 2
    def contains(self, x, y):
        r = self._s / 2
        h = self._x + r
        k = self._y + r
        return (x - h) ** 2 + (y - k) ** 2 <= r ** 2
    def __lt__(self, other):
        return self.area() < other.area()
```
Designing Data Types

```python
def __eq__(self, other):
    return self._x == other._x and self._y == other._y and 
            self._s == other._s

def __str__(self):
    r = self._s / 2
    h = self._x + r
    k = self._y + r
    return '(' + str(h) + ', ' + str(k) + ', ' + str(r) + ')'

def _main():
    import stdio

    c1 = Circle(1.0, 1.0, 2.0)
    c2 = Circle()
    stdio.writeln(c1.area())
    stdio.writeln(c1.contains(1.2, 2.2))
    stdio.writeln(c1 < c2)
    stdio.writeln(c1 == Circle(r=2.0, h=1.0, k=1.0))
    stdio.writeln(c1)

    if __name__ == '__main__':
        _main()

Solution 2.

*/
Circle
class Color:
    def __init__(self, r=0, g=0, b=0):
        self._r = r
        self._g = g
        self._b = b
    def getRed(self):
        return self._r
    def getGreen(self):
        return self._g
    def getBlue(self):
        return self._b
    def luminosity(self):
        return (.299 * self._r) + (.587 * self._g) + (.114 * self._b)
    def __add__(self, other):
        r = (self._r + other._r) // 2
        g = (self._g + other._g) // 2
        b = (self._b + other._b) // 2
        return Color(r, g, b)
    def __eq__(self, other):
        return self._r == other._r and 
                        self._g == other._g and 
                        self._b == other._b
    def __cmp__(self, other):
        if self.luminosity() < other.luminosity():
            return -1
        elif self.luminosity() == other.luminosity():
            return 0
        else:
            return 1
    def __str__(self):
        return '(' + str(self._r) + ', ' + str(self._g) + ', ' + str(self._b) + ')

def _main():
    import stdio
    c1 = Color(23, 45, 156)
    c2 = Color(34, 101, 78)
    c3 = c1 + c2
    a = [c1, c2, c3]
    a.sort()
    for v in a:
        stdio.writeln(str(v) + ' ' + str(v.luminosity()))
    stdio.writeln(c1 == c2)
```
```python
stdio.writeln(c1 == Color(23, 45, 156))

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

**Solution 3.**

```python
circle

import stdrandom
from color import Color

class RandomColors:
    def __init__(self, n):
        self._n = n
        self._current = 0

    def __iter__(self):
        return self

    def next(self):
        if self._current >= self._n:
            raise StopIteration
        self._current += 1
        r = stdrandom.uniformInt(0, 256)
        g = stdrandom.uniformInt(0, 256)
        b = stdrandom.uniformInt(0, 256)
        return Color(r, g, b)

def _main():
    import stdio
    import sys
    n = int(sys.argv[1])
    colors = []
    for color in RandomColors(n):
        colors.append([color])
    colors.sort()
    for color in colors:
        stdio.writeln(color)

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

**Solution 4.**

a. `colors.sort(cmp=lambda x, y: cmp(x.getBlue(), y.getBlue()))`

b. `colors.sort(cmp=lambda x, y: cmp(x.getRed() + x.getGreen() + x.getBlue(), y.getRed() + y.getGreen() + y.getBlue()))`