Name:

Readings

*How To Think Like A Computer Scientist*, chapters 12, 13 and 14.

Exercise 7.1 – Short Answers

1. What is the difference between a local variable and an object’s attribute?

2. What method is called when the object is created?

3. If you have an object instance, `obj`, and you want to call its `do_something()` method (assuming it has one), how would you do this? (write the line of code you would use)
7.2 – Understanding Objects

This is a pencil-and-paper exercise.

1. Write a class called Address that has two attributes: number and street_name. Make sure you have an __init__ method that initializes the object appropriately.

2. Consider the following code:

   ```python
   class Clock(object):
       def __init__(self, time):
           self.time = time

       def print_time(self):
           time = '6:30'
           print self.time

   clock = Clock('5:30')
   clock.print_time()
   ```

   (a) What does the code print out? If you aren’t sure, create a Python file and run it.

   (b) Is that what you expected? Why?
3. Consider the following code:

```python
class Clock(object):
    def __init__(self, time):
        self.time = time

    def print_time(self, time):
        print time

clock = Clock(‘5:30’)  
clock.print_time(‘10:30’)  

(a) What does the code print out? If you aren’t sure, create a Python file and run it.

(b) What does this tell you about giving parameters the same name as object attributes?
```

4. Consider the following code:

```python
class Clock(object):
    def __init__(self, time):
        self.time = time

    def print_time(self):
        print self.time

boston_clock = Clock(‘5:30’)  
paris_clock = boston_clock  
paris_clock.time = ‘10:30’  
boston_clock.print_time()

(a) What does the code print out? If you aren’t sure, create a Python file and run it.

(b) Why does it print what it does? (Are boston_clock and paris_clock different objects? Why or why not?)
7.3 – Inventing The Wheel

1. Graphics setup

Download the graphics.py file from the Materials section of the course webpage. Make sure to save it in the directory you’ll be doing your work in.
Run the module as if it were a Python program (open it in IDLE and run it). If everything was done correctly you will get a demo window with a triangle and some text.
Documentation on the graphics module is on the handout given in class, and also in the assignments section of the course.

2. Basic Graphics Application

Here is a skeleton program of any new graphics program for this class.

from graphics import *

#add any functions or classes you might define here

# create a window with width = 700 and height = 500
new_win = GraphWin('Program Name', 700, 500)

# add your code below this point

new_win.mainloop()

3. Animating the wheel

Download the file wheel.py from the Materials section of the course webpage (it is exactly the same as what you saw in class today). Run it and make sure that a wheel appears in a pop-up window.
Now let’s try to add an animate method that would move the wheel across the screen. We will make use of the move method in the wheel class that moves the object dx units in the x direction and dy units in the y direction. Here is what the animate method will look like.

from graphics import *

class Wheel(object):
...
    def animate(self, win, dx, dy, n):
        if n > 0:
            self.move(dx, dy)
            win.after(100, self.animate, win, dx, dy, n-1)

The animate method has 4 parameters - a GraphWin object win, the units by which to move the object in the x and y directions dx and dy, and the number of times to call the animate method, n. The animation will stop when n = 0. The interesting part here is the after method on the GraphWin object. The first parameter is the time in milliseconds after which the GraphWin object will call the animate method again. The second parameter is the function/method object the GraphWin object needs to call, in our case it is the animate method on the Wheel object. As we mentioned in class, in Python everything is an object even functions/methods and they can be passed as parameters to other functions/methods.
The rest of the parameters are the new parameters to the animate method. Note that we decrement n by 1 every time we setup a new call to animate.

Now write a program that will use the update Wheel class and create a Wheel object (you can pick the colors\(^1\) of the tire and wheel to be anything you want) and make it move the wheel across the screen by 1 unit in the x direction 100 times. Remember you first need to draw the wheel before you can move it.

### 7.4 – Drawing A Car

#### 1. Drawing Rectangles

To display a rectangle, you need to specify two points: the upper left corner and the bottom right corner. Remember our y-axis is flipped.

Make a file **car.py** and try the code below:

```python
from graphics import *

new_win = GraphWin("A Car", 300, 300)

rect = Rectangle(Point(10,10), Point(200,100))
rect.setFill("blue")
rect.draw(new_win)

new_win.mainloop()
```

Run your program and make sure that the rectangle appears on the screen.

Try changing the color and width of the outline of the rectangle. Look at the setOutline and setWidth methods.

#### 2. Drawing the car

In this exercise, we will create a class for a car that will use the Wheel class from exercise 3. The car will contain 3 attributes: two wheel objects and one rectangle object (the body of the car) that is horizontal and whose bottom corners correspond to the centers of the wheels. Below is an example on how to use the Car object. Try to figure out what the class Car should be based on the way it is used.

```python
new_win = GraphWin('A Car', 700, 300)

car1 = Car(Point(50, 50), 15, Point(100,50), 15, 40)
car1.draw( new_win )

car1.set_color('black', 'grey', 'pink' )

```

\(^1\)There's a list of available colors in the file rgb.txt on the course webpage.
# make the car move on the screen
car1.animate(new_win, 1, 0, 400)

new_win.mainloop()

The size of the wheel is given only by the radius of the tire circle. You can compute the radius of the wheel circle as a percentage of the radius of the tire circle, e.g. 60%.

### 7.5 – A Digital Clock – Optional!

#### 1. Drawing Text

The code below is an example of how to draw text on the screen:

```python
from graphics import *

# create the graphics window
new_win = GraphWin("Digital Clock", 300, 300)

# create a text objects centered at (100, 100)
msg1 = Text( Point( 100, 100 ), "Hello, world!" )
msg1.draw( new_win )

# process events
new_win.mainloop()
```

Run your program and make sure the string prints on the screen.

Try changing the font size and style and the color of the text. Look at the `setSize`, `setStyle`, and `setTextColor` methods in the documentation. All the set methods that change the attributes of the graphics object, automatically update its appearance on the screen. You can use the list of colors here.

#### 2. Drawing a Digital Clock

Create a file `digitalclock.py`. Create a class called `DigitalClock` that has attributes hour, minute, second and pos, and a draw method. The attributes store the time in military time, i.e. 3:30pm will be hour = 15, minute = 30, second = 23 and the position - the upper left corner of the rectangle face. Here is the code on how to use it:

```python
from graphics import *

# DigitalClock class definition goes here

new_win = GraphWin("Digital Clock", 300, 300)
clock = DigitalClock(15, 30, 23)
clock.draw(new_win)

new_win.mainloop()
```
And an example output:

![Image of 3:30:23 PM on a clock]

Feel free to choose the appearance of your clock.

**Hint:** You should add extra methods to help you draw the clock, e.g. a method for drawing the face, a method for drawing the text, a method returning the time as string. Choose appropriate names for your methods.

3. **Updating the clock (Super optional)**

Now you probably created a text object to display the time. Make it an attribute of the clock. Then add an update method that will update the time - both the object attributes and the display on the screen. Think about how you would increment the time. You may want to add other methods to help you. Take a look at the `setText` function on the Text class.

**Note:** The `setText` method will automatically redraw the text for you. You do not (and should not) call the draw method on the Text object again. You can only draw an object to the screen once.

You can create a tick method that would call update every second similar to the animate method in the previous exercise.

**Hint:** One thing you will have to worry about is handling scenarios, e.g. 05:35:59. The next time the clock updates it should show 05:36:00, not 05:35:60. Similarly for the minutes and hours. The modulus operator is your friend here.

**Another hint:** Here is an easy way to avoid trying to handle a lot of different cases. When you update, you first convert the time into seconds from the beginning of the day, then do the update, and then convert back to hour, minute, second. For example,

\[
\text{Current time: } 01:01:01 \implies 1*3600 + 1*60 + 1 = 3661 \\
\text{Update time: } 3661 + 1 = 3662 \\
\text{New time: } 3662 \implies 01:01:02
\]

Now you will only need to worry about how to handle updating 23:59:59 to 00:00:00. You may want to add extra methods to help you with this functionality - e.g. a method for converting the time to seconds, a method for splitting it back into hours, minutes, seconds, etc.