2.6 Case Study: Turtle Graphics Objects

In our first case study, we will use a graphics tool to (visually) illustrate the concepts covered in this chapter: objects, classes and class methods, object-oriented programming, and modules. The tool, Turtle graphics, allows a user to draw lines and shapes in a way that is similar to using a pen on paper.

Turtle Graphics

Turtle graphics has a long history all the way back to the time when the field of computer science was developing. It was part of the Logo programming language developed by Daniel Bobrow, Wally Feurzig, and Seymour Papert in 1966. The Logo programming language and its most popular feature, turtle graphics, was developed for the purpose of teaching programming.

The turtle was originally a robot, that is, a mechanical device controlled by a computer operator. A pen was attached to the robot and it left a trace on the surface as the robot moved according to functions input by the operator.
Turtle graphics is available to Python developers through the turtle module. In the
module are defined 7 classes and more than 80 class methods and functions. We will not
exhaustively cover all the features of the module turtle. We only introduce a sufficient
number to allow us to do interesting graphics while cementing our understanding of objects,
classes, class methods, functions, and modules. Feel free to explore this fun tool on your
own.

We start by importing the turtle module and then instantiating a Screen object.

```python
>>> import turtle
>>> s = turtle.Screen()
```

You will note that a new window appears with a white background after executing the
second statement. The Screen object is the canvas on which we draw. The Screen class
is defined in the turtle module. Later we will introduce some Screen methods that change
the background color or close the window. Right now, we just want to start drawing.

To create our pen or, using the turtle graphics terminology, our turtle, we instantiate a
Turtle object we name t:

```python
>>> t = turtle.Turtle()
```

A Turtle object is essentially a pen that is initially located at the center of the screen,
at coordinates (0,0). The Turtle class, defined in the turtle module, provides many
methods for moving the turtle. As we move the turtle, it leaves a trace behind. To make our
first move, we will use the forward() method of the Turtle class. So, to move forward
100 pixels, the method forward() is invoked on Turtle object t with 100 (pixels) as the
distance:

```python
>>> t.forward(100)
```

The effect is shown in Figure 2.6.

![Figure 2.6 Turtle graphics.](image)
The black arrow tip represents the Turtle object. The line is the trace left by the turtle after moving forward 100 pixels.

Note that the move is to the right. When instantiated, the turtle faces right (i.e., to the east). To make the turtle face a new direction, you can rotate it counterclockwise or clockwise using the left() or right() methods, both Turtle class methods. To rotate 90 degrees counterclockwise, the method left() is invoked on Turtle object t with the argument 90:

```python
>>> t.left(90)
```

We can have several Turtle objects simultaneously on the screen. Next, we create a new
Turtle instance that we name u and make both turtles do some moves:

```python
>>> u = turtle.Turtle()
```

```python
>>> u.left(90)
```

```python
>>> u.forward(100)
```

```python
>>> t.forward(100)
```

```python
>>> u.right(45)
```

The current state of the two turtles and the trace they made is shown in Figure 2.7.
In the example we just completed, we used three methods of class Turtle: `forward()`, `left()`, and `right()`. In Table 2.8, we list those and some other methods (but by no means all). To illustrate some of the additional methods listed in the table, we go through the steps necessary to draw a smiley face emoticon shown in Figure 2.8.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t.forward(distance)</code></td>
<td>Move turtle in the direction the turtle is headed by distance pixels</td>
</tr>
<tr>
<td><code>t.left(angle)</code></td>
<td>Rotate turtle counterclockwise by angle degrees</td>
</tr>
<tr>
<td><code>t.right(angle)</code></td>
<td>Rotate turtle clockwise by angle degrees</td>
</tr>
<tr>
<td><code>t.undo()</code></td>
<td>Undo the previous move</td>
</tr>
<tr>
<td><code>t.goto(x, y)</code></td>
<td>Move turtle to coordinates defined by x and y; if pen is down, draw line</td>
</tr>
<tr>
<td><code>t.setx(x)</code></td>
<td>Set the turtle’s first coordinate to x</td>
</tr>
<tr>
<td><code>t.sety(y)</code></td>
<td>Set the turtle’s second coordinate to y</td>
</tr>
<tr>
<td><code>t.setheading(angle)</code></td>
<td>Set orientation of turtle to angle, given in degrees;</td>
</tr>
<tr>
<td></td>
<td>Angle 0 means east, 90 is north, and so on</td>
</tr>
<tr>
<td><code>t.circle(radius)</code></td>
<td>Draw a circle with given radius; the center of the circle is radius pixels to the left of the turtle</td>
</tr>
<tr>
<td><code>t.circle(radius, angle)</code></td>
<td>Draw only the part the circle (see above) corresponding to angle</td>
</tr>
<tr>
<td><code>t.dot(diameter, color)</code></td>
<td>Draw a dot with given diameter and color</td>
</tr>
<tr>
<td><code>t.penup()</code></td>
<td>Pull pen up; no drawing when moving</td>
</tr>
<tr>
<td><code>t.pendown()</code></td>
<td>Put pen down; drawing when moving</td>
</tr>
<tr>
<td><code>t.pensize(width)</code></td>
<td>Set the pen line thickness to width</td>
</tr>
<tr>
<td><code>t.pencolor(color)</code></td>
<td>Set the pen color to color described by string color</td>
</tr>
</tbody>
</table>

Figure 2.8 A Turtle smiley face drawing.
We start by instantiating a Screen and a Turtle object and setting the pen size.

```python
>>> import turtle
>>> s = turtle.Screen()
>>> t = turtle.Turtle()
>>> t.pensize(3)
```

We then define the coordinates where the chin of the smiley face will be located, and then move to that location.

```python
>>> x = -100
>>> y = 100
>>> t.goto(x, y)
```

Oooops! We drew a line from coordinate (0, 0) to coordinate (-100, 100); all we wanted was to move the pen, without leaving a trace. So we need to undo the last move, lift the pen, and then move it.

```python
>>> t.undo()
>>> t.penup()
>>> t.goto(x, y)
>>> t.pendown()
```

Now we want to draw the circle outlining the face of our smiley face. We call the method `circle()` of the class Turtle with one argument, the radius of the circle. The circle is drawn as follows: The current turtle position will be a point of the circle, and the center of the circle is defined to be to the turtle’s left, with respect to the current turtle heading.

```python
>>> t.circle(100)
```

Now we want to draw the left eye. We choose the left eye coordinates relative to \((x, y)\) (i.e., the chin position) and “jump” to that location. We then use the dot function to draw a black dot of diameter 10.

```python
>>> t.penup()
>>> t.goto(x - 35, y + 120)
>>> t.pendown()
>>> t.dot(25)
```

Next, we jump and draw the right eye.

```python
>>> t.penup()
>>> t.goto(x + 35, y + 120)
>>> t.pendown()
>>> t.dot(25)
```

Finally, we draw the smile. I chose the exact location of the left endpoint of the smile using trial and error. You could also use geometry and trigonometry to get it right if you prefer. We use here a variant of the method `circle()` that takes a second argument in addition to the radius: an angle. What is drawn is just a section of the circle, a section corresponding to the given angle. Note that we again have to jump first.

```python
>>> t.penup()
>>> t.goto(x - 60.62, y + 65)
>>> t.pendown()
>>> t.setheading(-60)
>>> t.circle(70, 120)
```
We're done! As we end this case study, you may wonder how to close cleanly your turtle graphics window. The Screen method `bye()` closes it:

```python
>>> s.bye()
```

This method and several other Screen methods are listed in Table 2.9.

---

### Table 2.9 Methods of the Screen Class

<table>
<thead>
<tr>
<th>Usage</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sbgcolor(color)</code></td>
<td>Changes the background color of screen s to color described by string color</td>
</tr>
<tr>
<td><code>s.clearscreen()</code></td>
<td>Clears screen s</td>
</tr>
<tr>
<td><code>s.turtles()</code></td>
<td>Returns the list of all turtles in the screen s</td>
</tr>
<tr>
<td><code>s.bye()</code></td>
<td>Closes the screen s window</td>
</tr>
</tbody>
</table>

---

Start by executing these statements:

```python
>>> s = turtle.Screen()
>>> t = turtle.Turtle(shape='turtle')
>>> t.penup()
>>> t.goto(-300, 0)
>>> t.pendown()
```

A turtle pen will appear on the left side of the screen. Then execute a sequence of Python turtle graphics statements that will produce this image:

```
```

---

A turtle pen will appear on the left side of the screen. Then execute a sequence of Python turtle graphics statements that will produce this image:

```
```