Module III

Graphics

Graphics

- Terminal-based programs
- Simple Graphics using Turtle Operations
- 2D Shapes
- Colors and RGB Systems
- A case study.

Image Processing

- Basic image processing with inbuilt functions.

Graphical User Interfaces

- Event-driven programming
- Coding simple GUI-based programs
  - Windows, Labels, Displaying images, Input text entry,
    Popup dialog boxes, Command buttons
- A case study.

Graphics

The representation and display of geometric shapes in two- and three-dimensional space, as well as image processing.

Turtle graphics

A Turtle graphics toolkit provides a simple and enjoyable way to draw pictures in a window and gives you an opportunity to run several methods with an object.

Overview of Turtle Graphics

Turtle graphics were originally developed as part of the children's programming language Logo, created by Seymour Papert and his colleagues at MIT in the late 1960s.

The name is intended to suggest a way to think about the drawing process.

Imagine a turtle crawling on a piece of paper with a pen tied to its tail.

Commands direct the turtle as it moves across the paper and tell it to lift or lower its tail, turn some number of degrees left or right, and move a specified distance.

Whenever the tail is down, the pen drags along the paper, leaving a trail.

In this manner, it is possible to program the turtle to draw pictures ranging from the simple to the complex.

In the context of a computer, of course, the sheet of paper is a window on a display screen, and the turtle is an icon, such as an arrowhead.

At any given moment in time, the turtle is located at a specific position in the window This position is specified with (*x, y)* coordinates.

The coordinate system for Turtle graphics is the standard Cartesian system, with the origin (0, 0) at the center of a window.

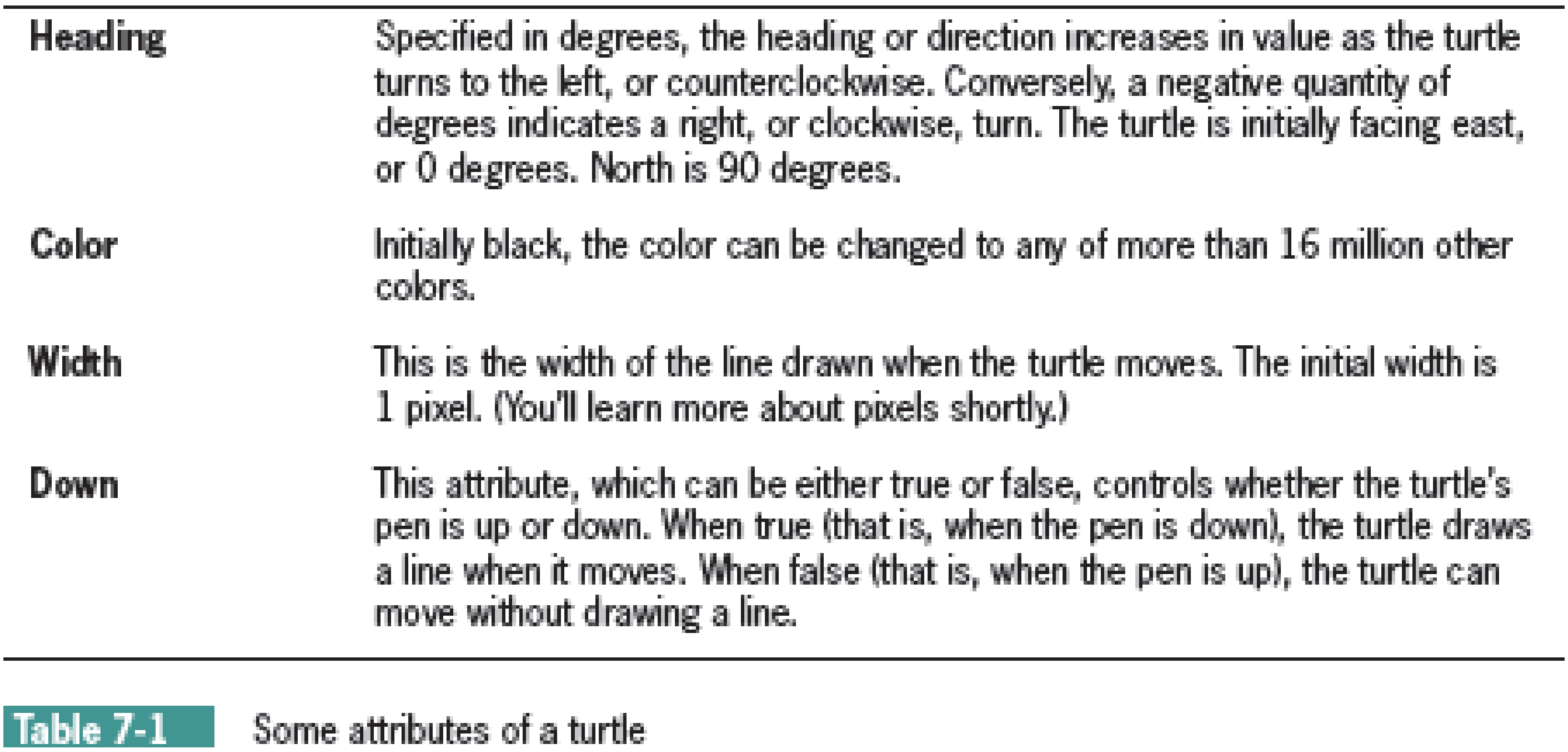
The turtle's initial position is the origin, which is also called the home.

An equally important attribute of a turtle is its heading, or the direction in which it currently faces.

The turtle's initial heading is 0 degrees, or due east on its map.

The degrees of the heading increase as it turns to the left, so 90 degrees is due north.

Together, these attributes make up a turtle's **state.**



Heading	Specified in degrees, the heading or direction increases in value as the turtle turns to the left, or counterclockwise. Conversely, a negative quantity of degrees indicates a right, or clockwise, turn. The turtle is initially facing east, or 0 degrees. North is 90 degrees.
Color	Initially black, the color can be changed to any of more than 16 million other colors.
Width	This is the width of the line drawn when the turtle moves. The initial width is 1 pixel. (You'll learn more about pixels shortly.)
Down	This attribute, which can be either true or false, controls whether the turtle's pen is up or down. When true (that is, when the pen is down), the turtle draws a line when it moves. When false (that is, when the pen is up), the turtle can move without drawing a line.

Table 7-1 Some attributes of a turtle

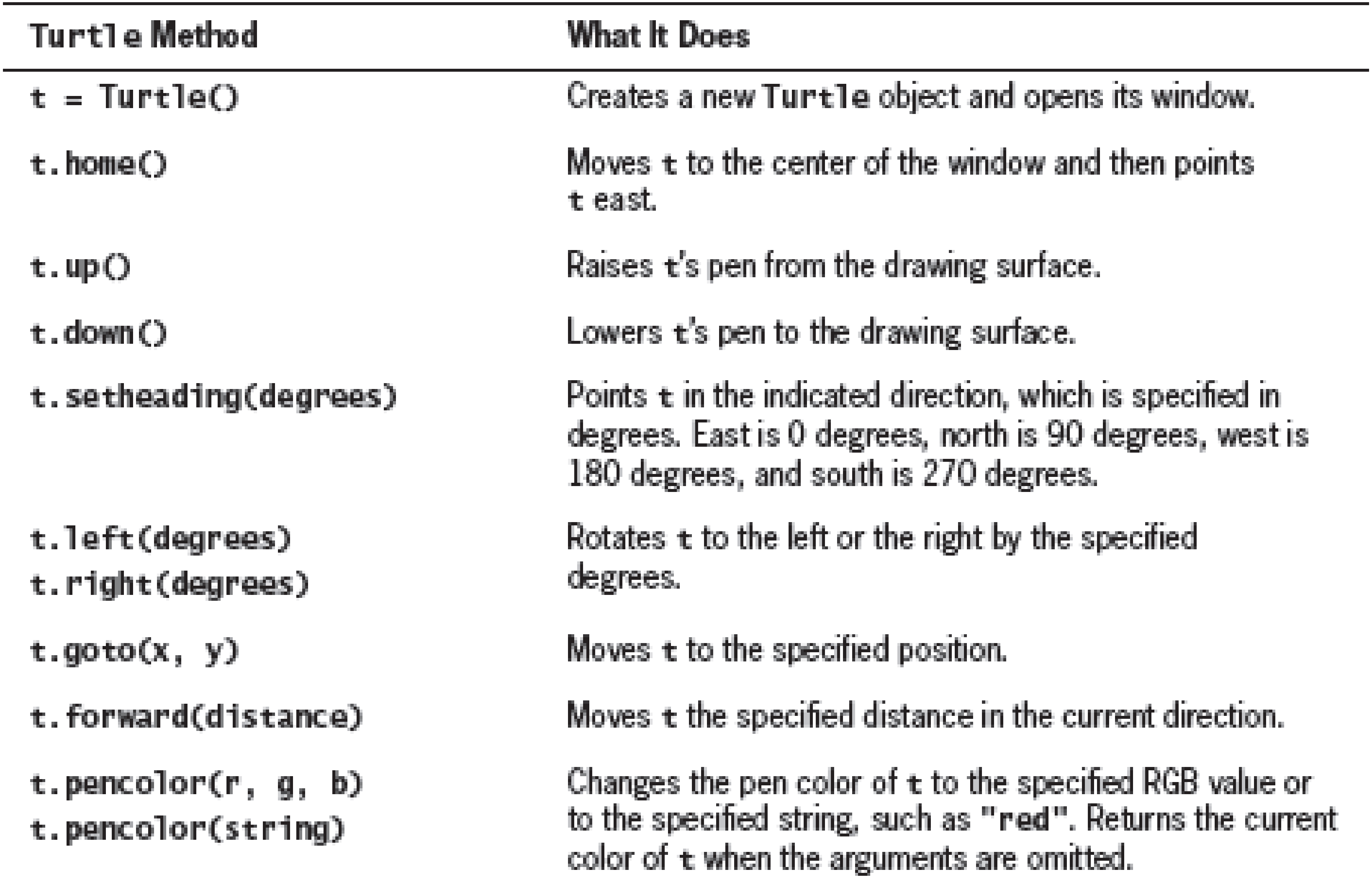
Turtle Operations

Every data value in Python is an **object.**

The types of objects are called **classes.**

Included in a class are the methods (or operations) that apply to objects of that class.

Turtle is an object, its operations are also defined as methods.



Turtle Method	What It Does
`t = Turtle()`	Creates a new `Turtle` object and opens its window.
`t.home()`	Moves `t` to the center of the window and then points `t` east.
`t.up()`	Raises `t`'s pen from the drawing surface.
`t.down()`	Lowers `t`'s pen to the drawing surface.
`t.setheading(degrees)`	Points `t` in the indicated direction, which is specified in degrees. East is 0 degrees, north is 90 degrees, west is 180 degrees, and south is 270 degrees.

`t.left(degrees)`
`t.right(degrees)`
	Rotates `t` to the left or the right by the specified degrees.

`t.goto(x, y)`	Moves `t` to the specified position.
`t.forward(distance)`	Moves `t` the specified distance in the current direction.

`t.pencolor(r, g, b)`
`t.pencolor(string)`
	Changes the pen color of `t` to the specified RGB value or to the specified string, such as `"red"`. Returns the current color of `t` when the arguments are omitted.

`t.fillcolor(r, g, b)`
`t.fillcolor(string)`
	Changes the fill color of `t` to the specified RGB value or to the specified string, such as `"red"`. Returns the current fill color of `t` when the arguments are omitted.

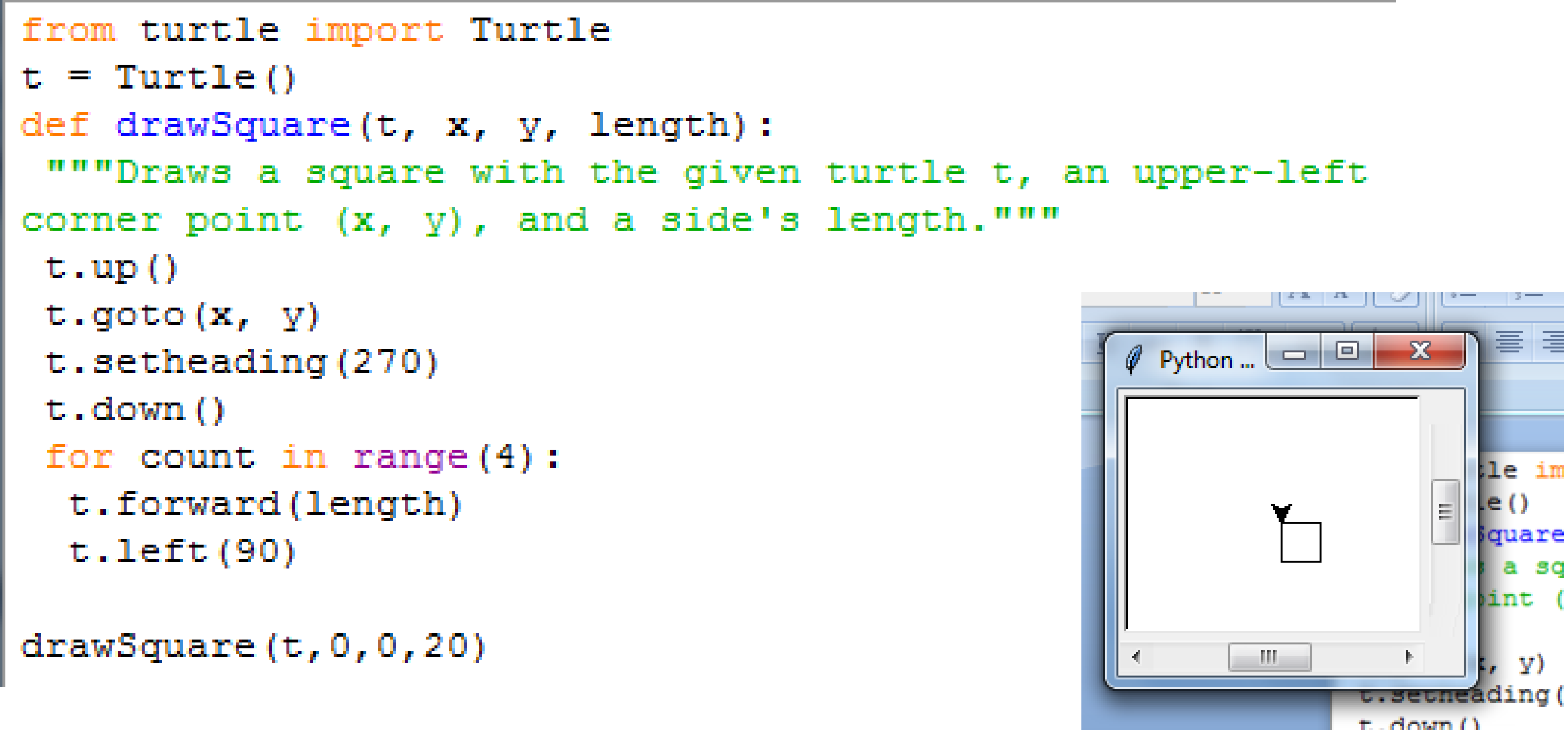
`t.begin_fill()`
`t.end_fill()`
	Enclose a set of turtle commands that will draw a filled shape using the current fill color.

`t.clear()`	Erases all of the turtle's drawings, without changing the turtle's state.
`t.width(pixels)`	Changes the width of `t` to the specified number of pixels. Returns `t`'s current width when the argument is omitted.

`t.hideturtle()`
`t.showturtle()`
	Makes the turtle invisible or visible.

`t.position()`	Returns the current position `(x, y)` of `t`.
`t.heading()`	Returns the current direction of `t`.
`t.isdown()`	Returns `True` if `t`'s pen is down or `False` otherwise.

The **Turtle** methods



```
from turtle import Turtle
t = Turtle()
def drawSquare(t, x, y, length):
 """Draws a square with the given turtle t, an upper-left
corner point (x, y), and a side's length."""
 t.up()
 t.goto(x, y)
 t.setheading(270)
 t.down()
 for count in range(4):
  t.forward(length)
  t.left(90)

drawSquare(t,0,0,20)
```

This function expects a Turtle object, a pair of integers that indicate the coordinates of the square’s upper-left corner, and an integer that designates the length of a side.

The function begins by lifting the turtle up and moving it to the square’s corner point.

It then points the turtle due south—270 degrees—and places the turtle’s pen down on the drawing surface.

Finally, it moves the turtle the given length and turns it left by 90 degrees, four times.

Two other important classes used in Python’s Turtle graphics system are Screen, which represents a turtle’s associated window.

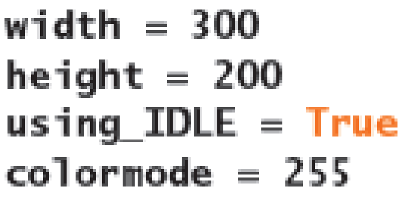
Canvas, which represents the area in which a turtle can move and draw lines.

A canvas can be larger than its window, which displays just the area of the canvas visible to the human user.

Setting Up a turtle.cfg File and Running IDLE

A Turtle graphics configuration file, which has the filename **turtle.cfg, is a text file that contains the initial settings of several attributes of Turtle,** Screen, and Canvas objects.

Python creates default settings for these attributes.



```
width = 300
height = 200
using_IDLE = True
colormode = 255
```

Object Instantiation and the turtle Module

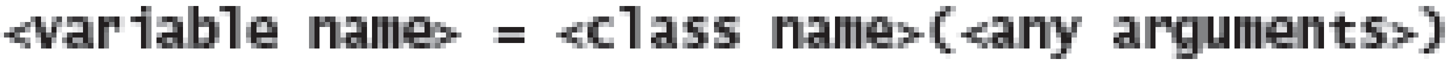
Before using objects, create them.

Create an **instance of the object's class.**

The process of creating an object is called instantiation.

Python automatically creates objects such as numbers, strings, and lists when it encountered them as literals.

The programmer must explicitly instantiate other classes of objects, including those that have no literals.



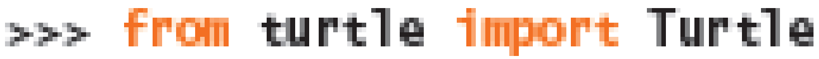
```
<variable name> = <class name>(<any arguments>)
```

The expression on the right side of the assignment, also called a **constructor, resembles** a function call.

The constructor can receive as arguments any initial values for the new object's attributes, or other information needed to create the object.

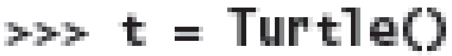
The Turtle class is defined in the turtle module.

The following code imports the Turtle class for use in a session:



```
>>> from turtle import Turtle
```

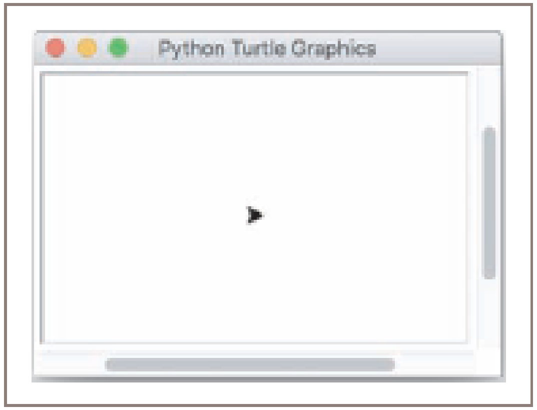
The next code segment creates and returns a Turtle object and opens a drawing window.



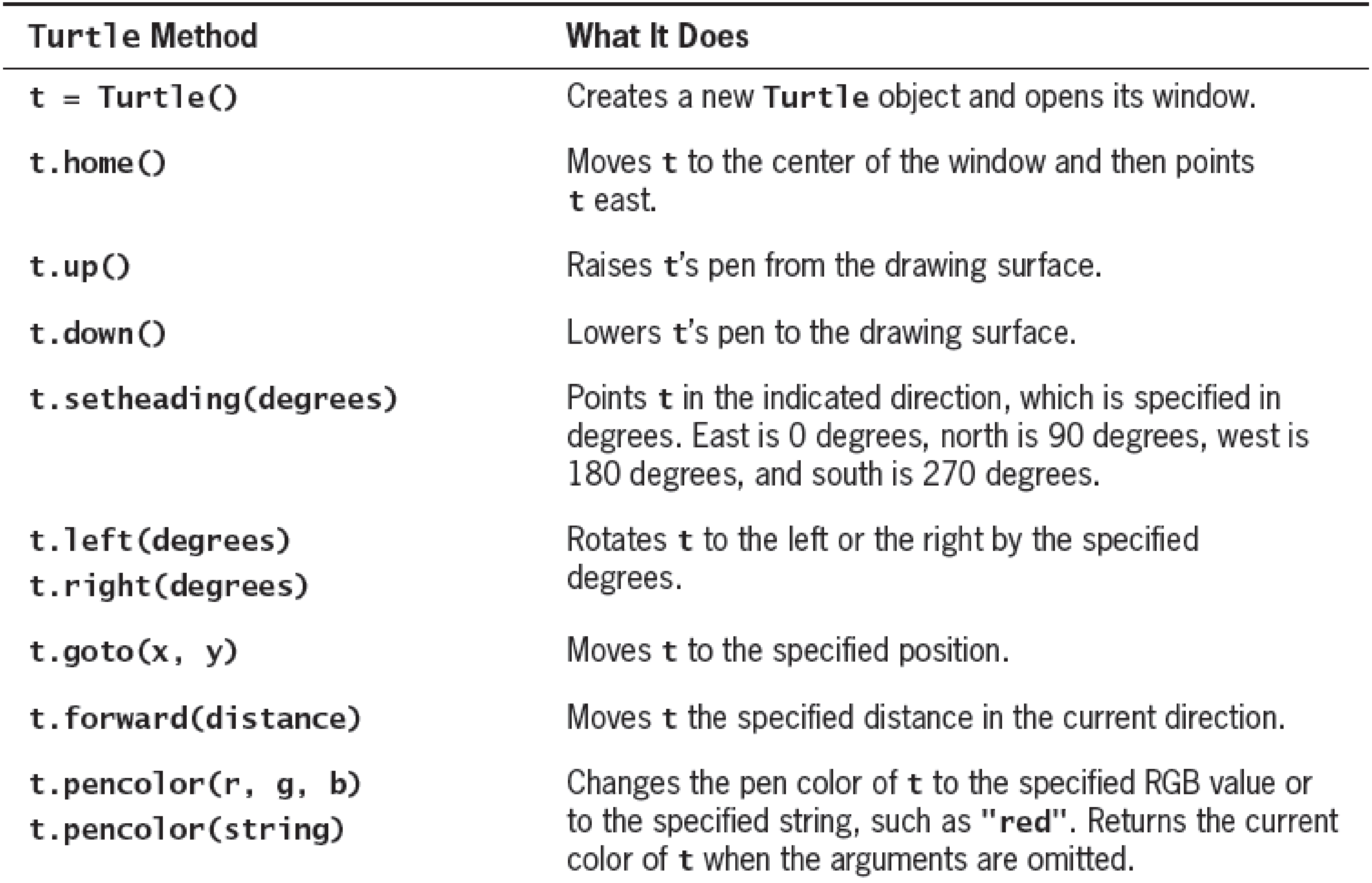
```
>>> t = Turtle()
```

The turtle’s icon is located at the home position (0, 0) in the center of the window, facing east and ready to draw.

The user can resize the window in the usual manner.



```
>>> t.width(2)                # For bolder lines
>>> t.left(90)                # Turn to face north
>>> t.forward(30)             # Draw a vertical line in black
>>> t.left(90)                # Turn to face west
>>> t.up()                    # Prepare to move without drawing
>>> t.forward(10)             # Move to beginning of horizontal line
>>> t.setheading(0)           # Turn to face east
>>> t.pencolor("red")
>>> t.down()                  # Prepare to draw
>>> t.forward(20)             # Draw a horizontal line in red
>>> t.hideturtle()            # Make the turtle invisible
```



Turtle Method	What It Does
`t = Turtle()`	Creates a new **`Turtle`** object and opens its window.
`t.home()`	Moves **`t`** to the center of the window and then points **`t`** east.
`t.up()`	Raises **`t`**'s pen from the drawing surface.
`t.down()`	Lowers **`t`**'s pen to the drawing surface.
`t.setheading(degrees)`	Points **`t`** in the indicated direction, which is specified in degrees. East is 0 degrees, north is 90 degrees, west is 180 degrees, and south is 270 degrees.

`t.left(degrees)`
`t.right(degrees)`
	Rotates **`t`** to the left or the right by the specified degrees.

`t.goto(x, y)`	Moves **`t`** to the specified position.
`t.forward(distance)`	Moves **`t`** the specified distance in the current direction.

`t.pencolor(r, g, b)`
`t.pencolor(string)`
	Changes the pen color of **`t`** to the specified RGB value or to the specified string, such as **`"red"`**. Returns the current color of **`t`** when the arguments are omitted.

`t.fillcolor(r, g, b)`
`t.fillcolor(string)`
	Changes the fill color of `t` to the specified RGB value or to the specified string, such as `"red"`. Returns the current fill color of `t` when the arguments are omitted.

`t.begin_fill()`
`t.end_fill()`
	Enclose a set of turtle commands that will draw a filled shape using the current fill color.

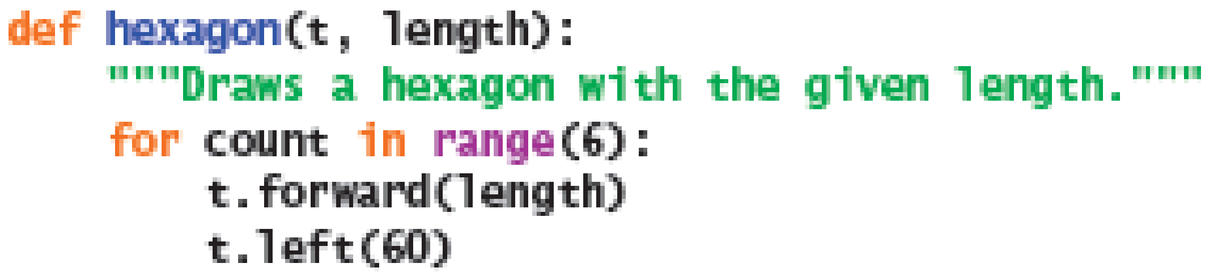
`t.clear()`	Erases all of the turtle's drawings, without changing the turtle's state.
`t.width(pixels)`	Changes the width of `t` to the specified number of pixels. Returns `t`'s current width when the argument is omitted.

`t.hideturtle()`
`t.showturtle()`
	Makes the turtle invisible or visible.

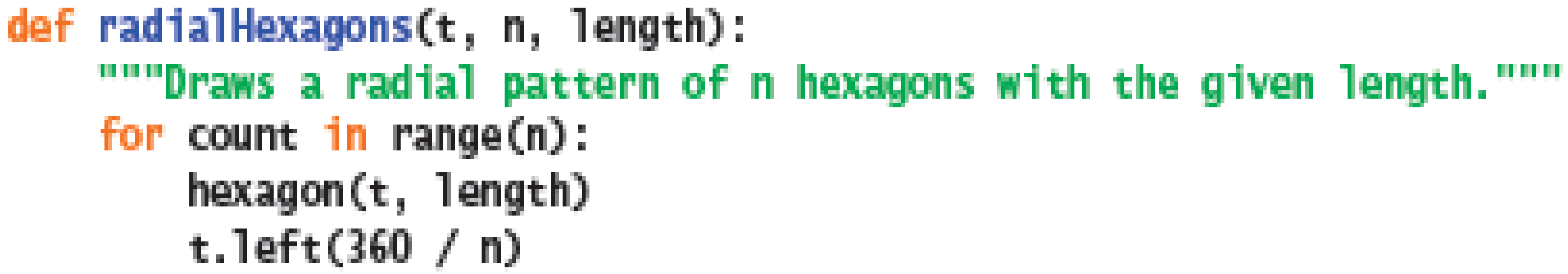
`t.position()`	Returns the current position `(x, y)` of `t`.
`t.heading()`	Returns the current direction of `t`.
`t.isdown()`	Returns `True` if `t`'s pen is down or `False` otherwise.

Drawing Two-Dimensional Shapes

Many graphics applications use vector graphics, which includes the drawing of simple two-dimensional shapes, such as rectangles, triangles, pentagons, and circles.



```
def hexagon(t, length):
    """Draws a hexagon with the given length."""
    for count in range(6):
        t.forward(length)
        t.left(60)
```



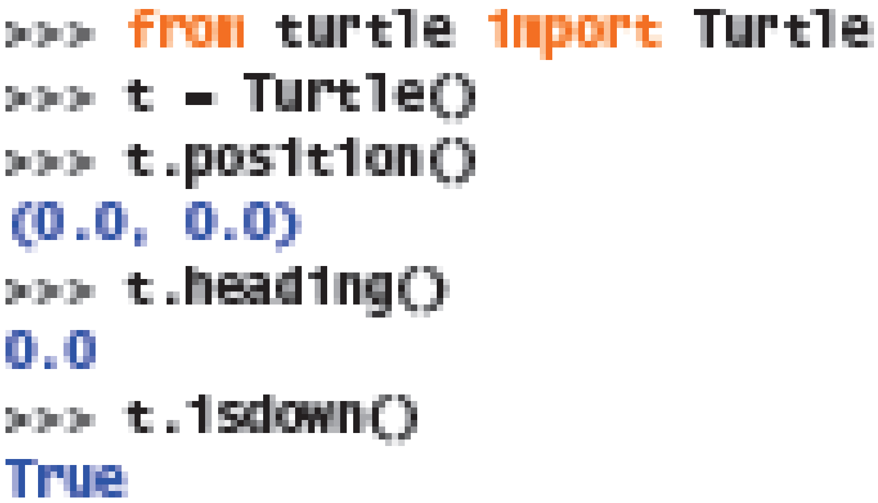
```
def radialHexagons(t, n, length):
    """Draws a radial pattern of n hexagons with the given length."""
    for count in range(n):
        hexagon(t, length)
        t.left(360 / n)
```

Examining an Object's Attributes

The Turtle methods modify a Turtle object's attributes, such as its position, heading, and color.

Mutator methods - Change the internal state of a Turtle object.

Accessor methods - Return the values of a Turtle object's attributes without altering its state.



```
>>> from turtle import Turtle
>>> t = Turtle()
>>> t.position()
(0.0, 0.0)
>>> t.heading()
0.0
>>> t.isdown()
True
```

Manipulating a Turtle's Screen

Turtle object of type (Class)

Screen - turtle's window

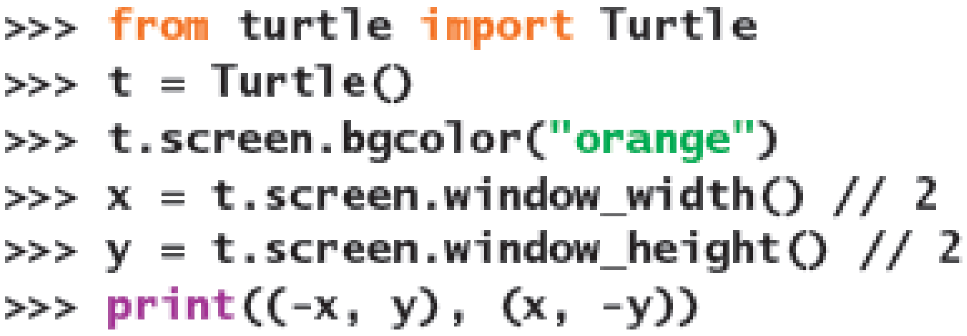
Canvas - drawing area in window

Screen object's attributes – width , height in pixels

background color

Access Screen object - t.screen.

window_width() and window_height() to locate the boundaries of a turtle's window.



```
>>> from turtle import Turtle
>>> t = Turtle()
>>> t.screen.bgcolor("orange")
>>> x = t.screen.window_width() // 2
>>> y = t.screen.window_height() // 2
>>> print((-x, y), (x, -y))
```

Colors and the RGB System

The rectangular display area on a computer screen is made up of colored dots called picture elements or pixels.

The smaller the pixel, the smoother the lines drawn with them will be.

Setting the resolution to smaller values increases the size of the pixels, making the lines on the screen appear more ragged.

Each pixel represents a color.

Turtle’s default color is black

pencolor method to change the colour .

RGB system common colour representation system

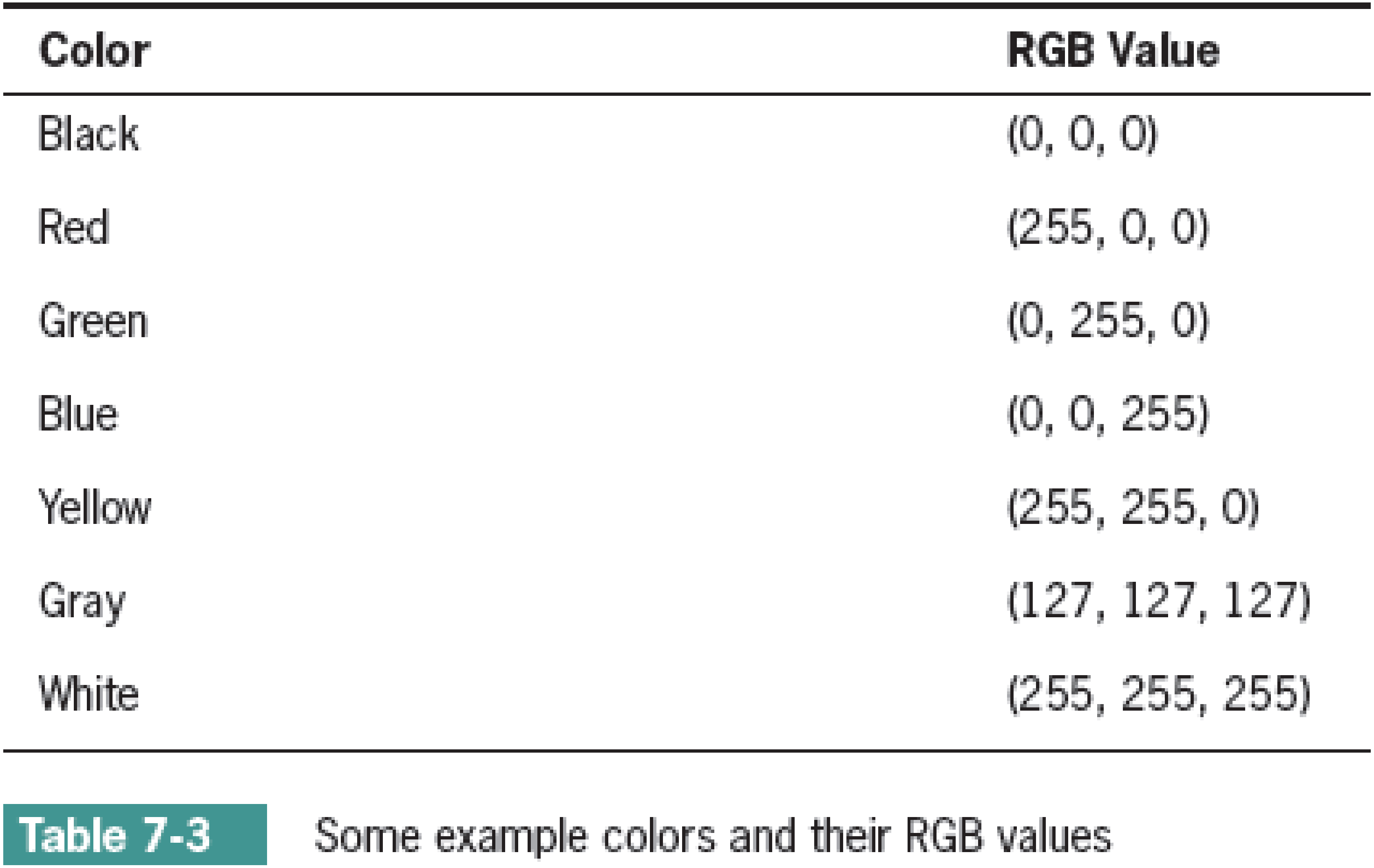
RGB components are mixed together to form a unique color value.

Computer represents these values as integers, and the display hardware translates this information to the colors .

Each color component can range from 0 through 255.

The value 255 represents the maximum saturation of a given color component.

Value 0 represents the total absence of that component.



Color	RGB Value
Black	(0, 0, 0)
Red	(255, 0, 0)
Green	(0, 255, 0)
Blue	(0, 0, 255)
Yellow	(255, 255, 0)
Gray	(127, 127, 127)
White	(255, 255, 255)

Table 7-3 Some example colors and their RGB values

How many total RGB color values are possible?

256 * 256 * 256, or 16,777,216 distinct color values

Human eye cannot discriminate between adjacent color values in this set, the RGB system is called a **true color system.**

Another way to consider color is from the perspective of the computer memory required to represent a pixel's color.

n bits of memory can represent 2^n distinct data values. Conversely, *n distinct data values require at least $\log_2 n$ bits of memory.*

when memory was expensive and displays came in black and white, only a single bit of memory was required to represent the two color.

When displays capable of showing 8 shades of gray came along, 3 bits of memory were required to represent each color value.

Early color monitors might have supported the display of 256 colors, so 8 bits were needed to represent each color value.

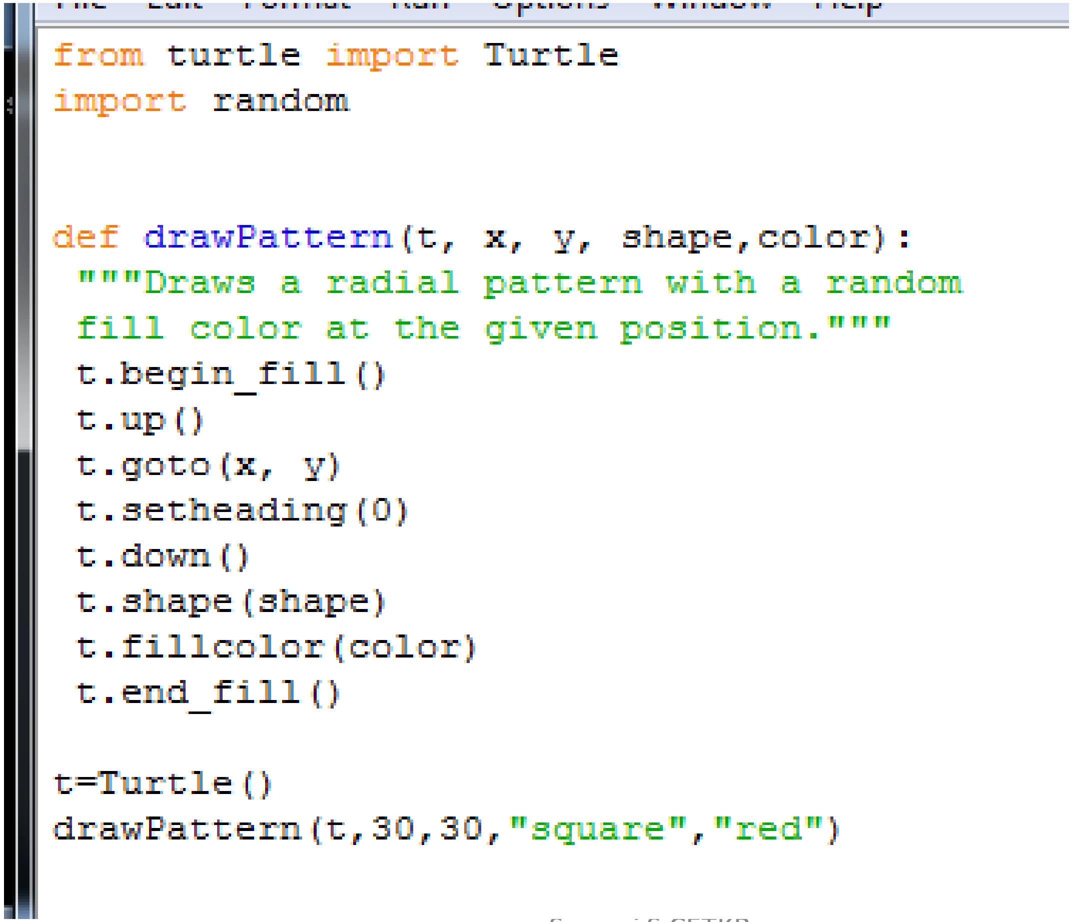
Each color component of an RGB color requires 8 bits, so the total number of bits needed to represent a distinct color value is 24.

The total number of RGB colors, 2^{24}, happens to be 16,777,216.

Filling Radial Patterns with Random Colors

The Turtle class includes the pencolor and fillcolor methods for changing the turtle's drawing and fill colors, respectively.

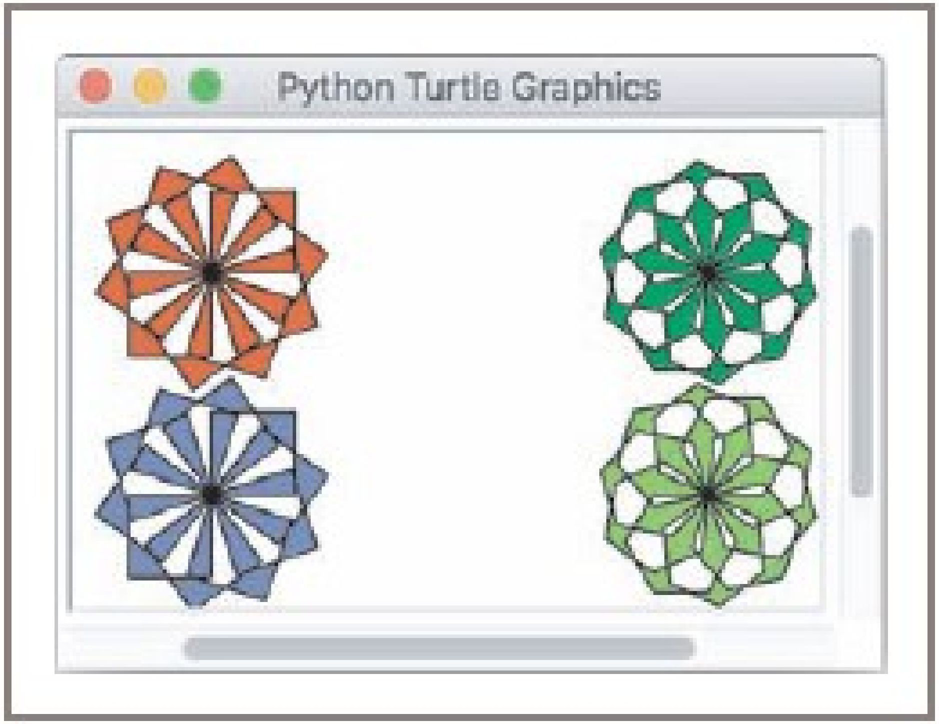
These methods can accept integers for the three RGB components as arguments.



```
from turtle import Turtle
import random


def drawPattern(t, x, y, shape,color):
 """Draws a radial pattern with a random
 fill color at the given position."""
 t.begin_fill()
 t.up()
 t.goto(x, y)
 t.setheading(0)
 t.down()
 t.shape(shape)
 t.fillcolor(color)
 t.end_fill()

t=Turtle()
drawPattern(t,30,30,"square","red")
```



Python Turtle Graphics

Image Processing

Discrete values

Analog values

Sampling and Digitizing Images

Sampling devices measure discrete color values at distinct points on a two-dimensional **grid.**

These values are pixels

More pixels that are sampled – More realistic image

Sampling of 10 pixels per millimeter (250 pixels per inch and 62,500 pixels per square inch)

3-inch by 5-inch image - 3 *5 *62,500 pixels/inch^2 = 937,500 pixels

approximately one megapixel

Image File Formats

- A raw image file saves all of the sampled information.
- Large in Size.
- Storage – Expensive.
- Scheme to Compress file size.
- Easy to send across networks.

Jpeg

lossless compression – Saves position rather than pixel value

lossy scheme – larger regions assigns an approximate pixel value

GIF

lossy compression scheme

The compression algorithm consists of two phases

1. Build a colour palette up to 256 of the most prevalent colours.

2. Visit each sample in the grid and replaces it with the *key of the closest color in the color palette*

Images with broad, flat areas of the same color,
such as cartoons, backgrounds, and banners.

Image-Manipulation Operations

Transform the information in the pixels

Alter the arrangement of the pixels in the image.

- Rotate an image
- Convert an image from colour to greyscale
- Apply colour filtering to an image
- Highlight a particular area in an image
- Blur all or part of an image
- Sharpen all or part of an image
- Control the brightness of an image
- Perform edge detection on an image
- Enlarge or reduce an image’s size
- Apply colour inversion to an image
- Morph an image into another image

The Properties of Images

The software maps the bits from the image file into a rectangular area of coloured pixels for display.

The coordinates of the pixels range from (0, 0) at the upper-left corner of an image to (*width – 1, height – 1) at the lower-right corner.*

Different from the standard Cartesian coordinate system used with Turtle graphics, where the origin (0, 0) is at the center of the rectangular grid.

An image consists of a width, a height, and a set of color values accessible by means of (*x, y) coordinates.*

A color value consists of the tuple (*r, g, b*)*, where the variables refer to the integer values of its red, green,* and *blue* components, respectively.

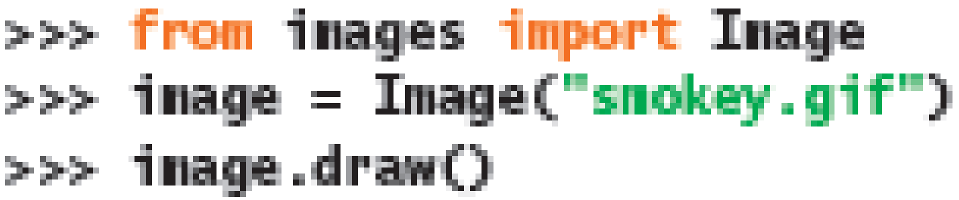
The images Module

The images module is a non-standard, open-source Python tool.

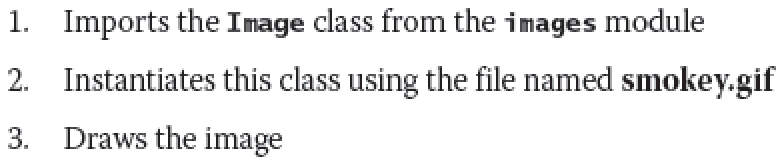
The images module includes a class named Image.

The Image class represents an image as a two-dimensional grid of RGB values.

Python raises an exception if it cannot locate the file in the current directory, or if the file is not a GIF file.

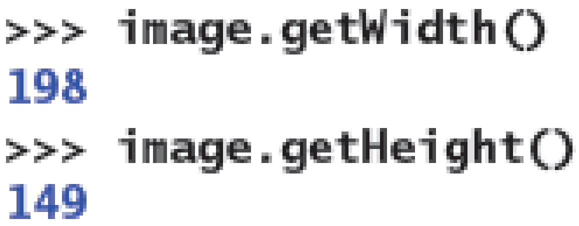


```
>>> from images import Image
>>> image = Image("smokey.gif")
>>> image.draw()
```

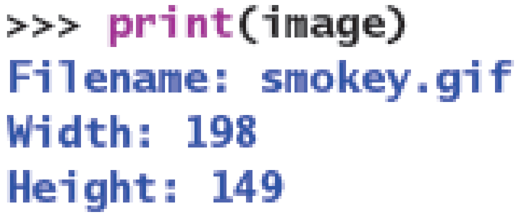


1. Imports the `Image` class from the `images` module
2. Instantiates this class using the file named **smokey.gif**
3. Draws the image

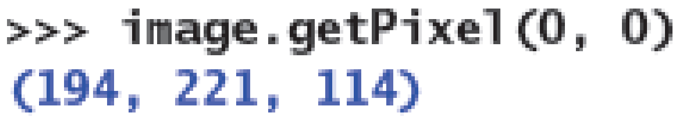
Image Method	What It Does
`i = Image(filename)`	Loads and returns an image from a file with the given filename. Raises an error if the filename is not found or the file is not a GIF file.
`i = Image(width, height)`	Creates and returns a blank image with the given dimensions. The color of each pixel is transparent, and the filename is the empty string.
`i.getWidth()`	Returns the width of `i` in pixels.
`i.getHeight()`	Returns the height of `i` in pixels.
`i.getPixel(x, y)`	Returns a tuple of integers representing the RGB values of the pixel at position (x, y).
`i.setPixel(x, y, (r, g, b))`	Replaces the RGB value at the position (x, y) with the RGB value given by the tuple `(r, g, b)`.
`i.draw()`	Displays `i` in a window. The user must close the window to return control to the method's caller.
`i.clone()`	Returns a copy of `i`.
`i.save()`	Saves `i` under its current filename. If `i` does not yet have a filename, `save` does nothing.
`i.save(filename)`	Saves `i` under `filename`. Automatically adds a **.gif** extension if `filename` does not contain it.



```
>>> image.getWidth()
198
>>> image.getHeight()
149
```



```
>>> print(image)
Filename: smokey.gif
Width: 198
Height: 149
```

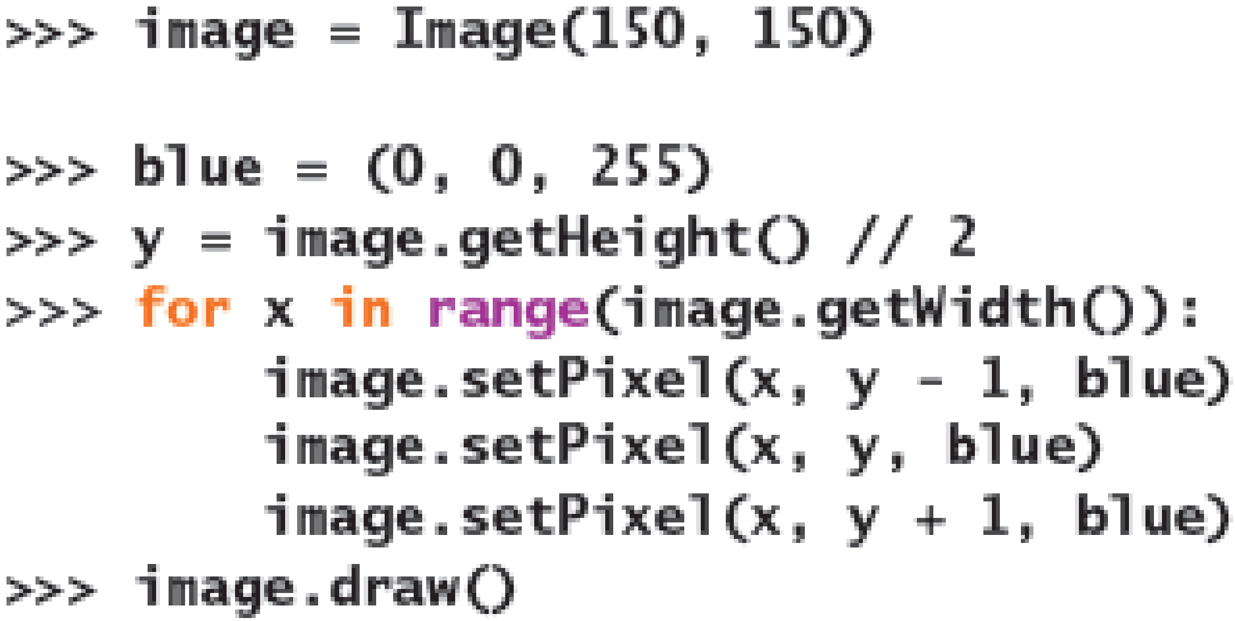


```
>>> image.getPixel(0, 0)
(194, 221, 114)
```

The method getPixel returns a tuple of the RGB values at the given coordinates.

See the information for the pixel at position (0, 0), which is at the image’s upper-left corner

The programmer can use the method setPixel to replace an RGB value at a given position in an image.



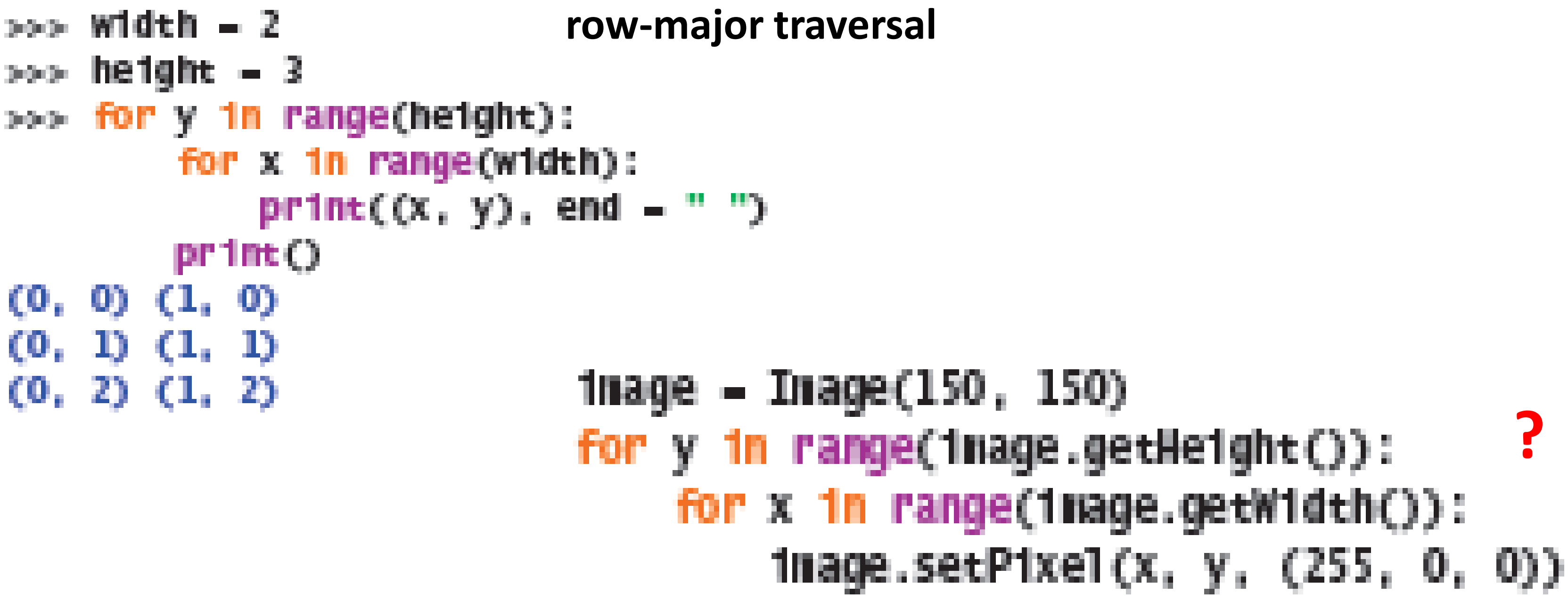
```
>>> image = Image(150, 150)

>>> blue = (0, 0, 255)
>>> y = image.getHeight() // 2
>>> for x in range(image.getWidth()):
        image.setPixel(x, y - 1, blue)
        image.setPixel(x, y, blue)
        image.setPixel(x, y + 1, blue)
>>> image.draw()
```

A Loop Pattern for Traversing a Grid

Image-processing algorithms use a nested loop structure to traverse a two-dimensional grid of pixels

The outer loop iterates over one coordinate, while the inner loop iterates over the other coordinate



```
>>> width = 2
>>> height = 3
>>> for y in range(height):
        for x in range(width):
            print((x, y), end = " ")
        print()
(0, 0) (1, 0)
(0, 1) (1, 1)
(0, 2) (1, 2)
```

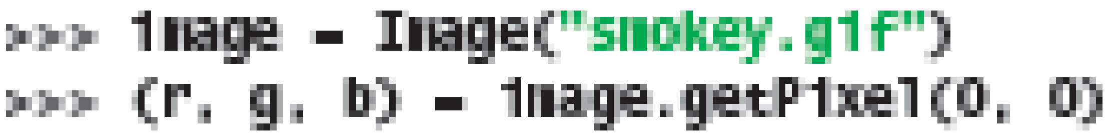
row-major traversal

```
image = Image(150, 150)
for y in range(image.getHeight()):
    for x in range(image.getWidth()):
        image.setPixel(x, y, (255, 0, 0))
```

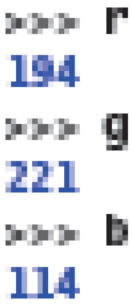
?

A pixel’s RGB values are stored in a tuple, manipulating them is quite easy.

getPixel() to retrieve a tuple

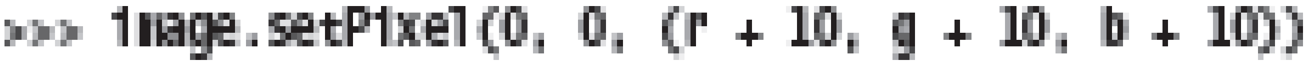


```
>>> image = Image("smokey.gif")
>>> (r, g, b) = image.getPixel(0, 0)
```

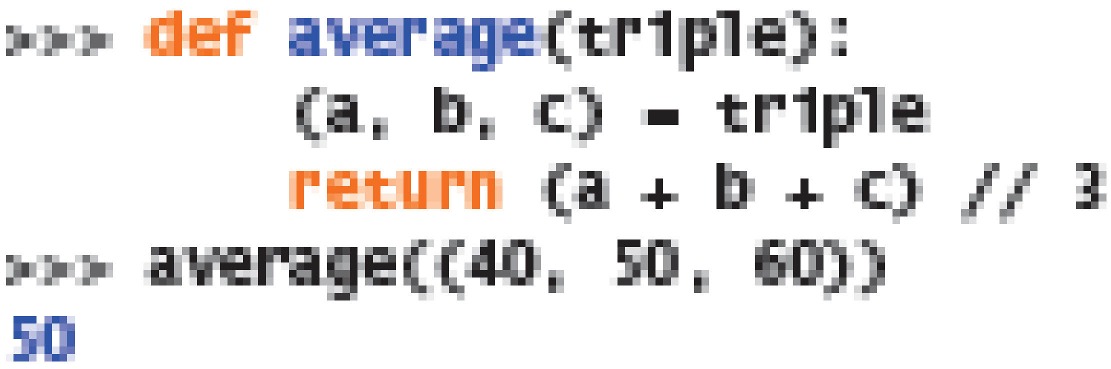


```
>>> r
194
>>> g
221
>>> b
114
```

New tuple with the results of the computations and resetting the pixel to that tuple:



```
>>> image.setPixel(0, 0, (r + 10, g + 10, b + 10))
```



```
>>> def average(triple):
        (a, b, c) = triple
        return (a + b + c) // 3
>>> average((40, 50, 60))
50
```

Converting an Image to Black and White

For each pixel, the algorithm computes the average of the red, green, and blue values.

The algorithm then resets the pixel's color values to 0 (black) if the average is closer to 0.

To 255 (white) if the average is closer to 255.

{code}

Converting an Image to Grayscale

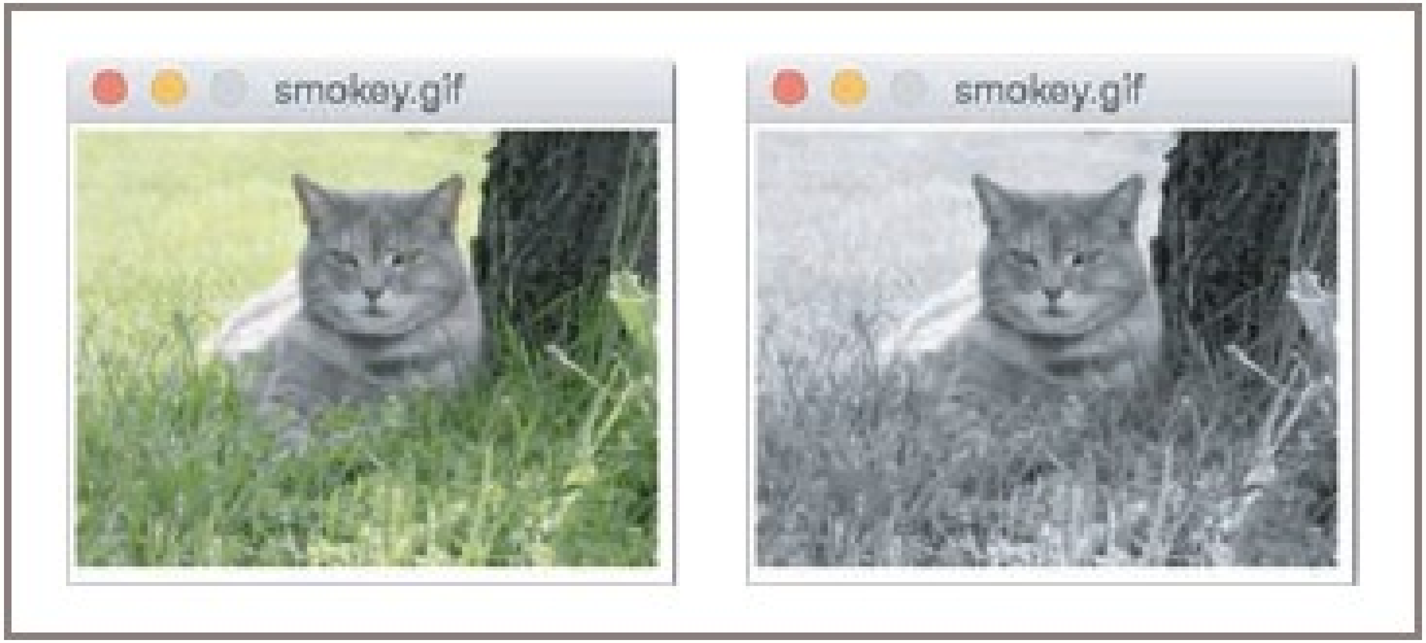
Black-and-white photographs are not really just black and white; they also contain various shades of gray known as grayscale.

Color values in Grayscale might be 8, 16, or 256 shades of gray (including black and white at the extremes).

To obtain the new RGB values, instead of adding up the color values and dividing by 3, multiply each one by a weight factor and add the results.

Relative luminance proportions of green, red, and blue are .587, .299, and .114, respectively. These values add up to 1.

{Code}


smokey.gif
smokey.gif

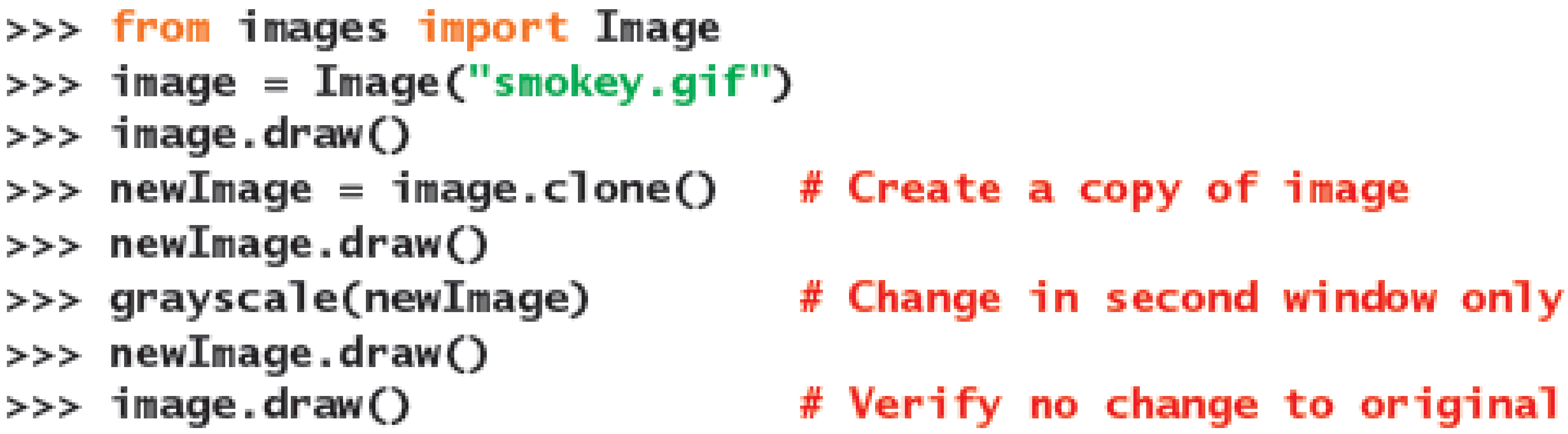
Copying an Image

One could create a new, blank image of the same height and width as the original.

The Image class includes a clone method for this purpose.

The method clone builds and returns a new image with the same attributes as the original one, but with an empty string as the filename.

The two images are thus structurally equivalent but not identical.



```
>>> from images import Image
>>> image = Image("smokey.gif")
>>> image.draw()
>>> newImage = image.clone()   # Create a copy of image
>>> newImage.draw()
>>> grayscale(newImage)        # Change in second window only
>>> newImage.draw()
>>> image.draw()               # Verify no change to original
```

Blurring an Image

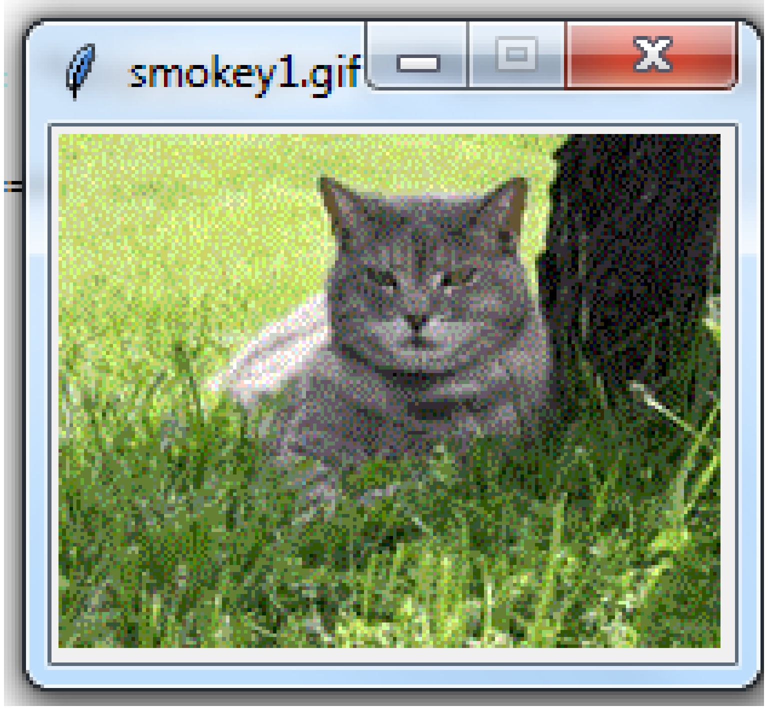
An image appears to contain rough, jagged edges, known as pixilation.

This can be mitigated by blurring the image’s problem areas.

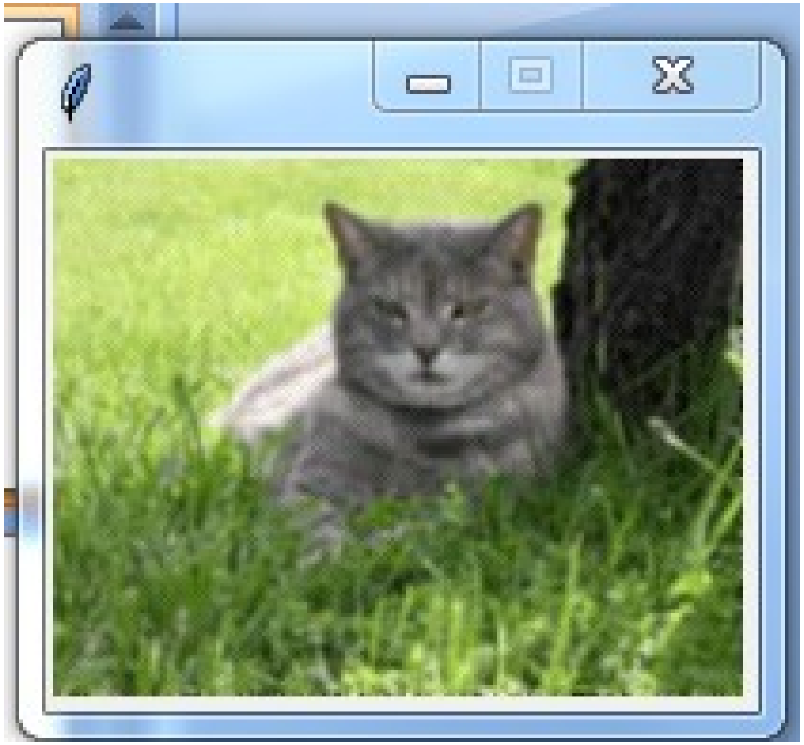
Blurring makes these areas appear softer, but losing some definition.

This algorithm resets each pixel’s color to the average of the colors of the four pixels that surround it.

{Code}



Before blur



After blur

Edge Detection

Inverse function on a color image.

Examines the below, left of each pixel in an image

Detects an edge

- if luminance differs by significant amount.
- set color to black, else to white.

{code}

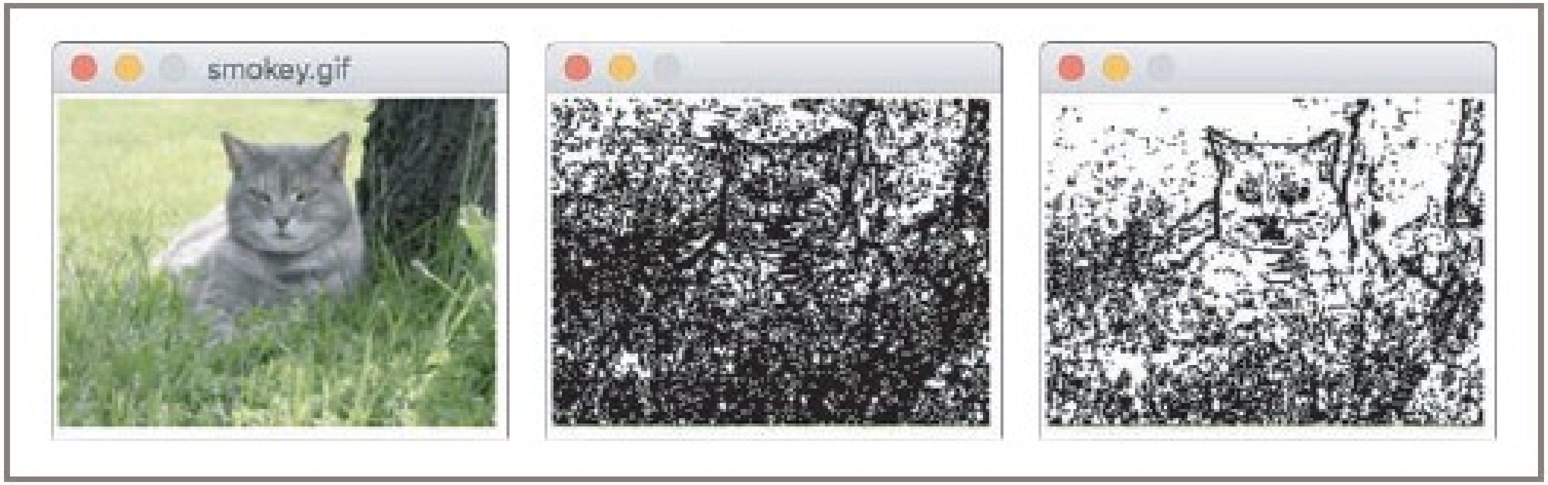


Figure 7-14 Edge detection: the original image, a luminance threshold of 10, and a luminance threshold of 20

Reducing the Image Size

- The size and the quality of an image depends on
  - Image’s width and height in pixels
  - Display medium’s resolution.
- Resolution is measured in pixels, or dots per inch (DPI).
- Decide resolution of an image itself before the image is captured.
- Reducing an image’s size
  - Reduces load time in a Web page.
  - Reduces space occupied on a storage medium

A size reduction usually preserves an image’s **aspect ratio.**

Method

1. Create a new image whose width and height are a constant fraction of the original image’s width and height.

2. Copy color value to new image

To reduce the size of an image by a factor of 2, copy the color values from every other row and every other column of the original image to the new image.

Shrink function   {code}

Disadvantages

- **T**hrows away pixel information
- **G**reater the reduction, the greater the information loss.

Human eye does not normally notice the loss of visual information.

The results are different when an image is enlarged

Because it requires transform the existing pixels to blend in with the new ones that are added

Graphical User Interfaces

Graphical user interface (GUI)

Displays text/icons

Commands activated by pressing the enter key or control keys

1. A GUI program is event driven
2. Inactive until user interacts GUI components
3. Enter inputs in any order
4. Running different data sets does not require re-entering all of the data

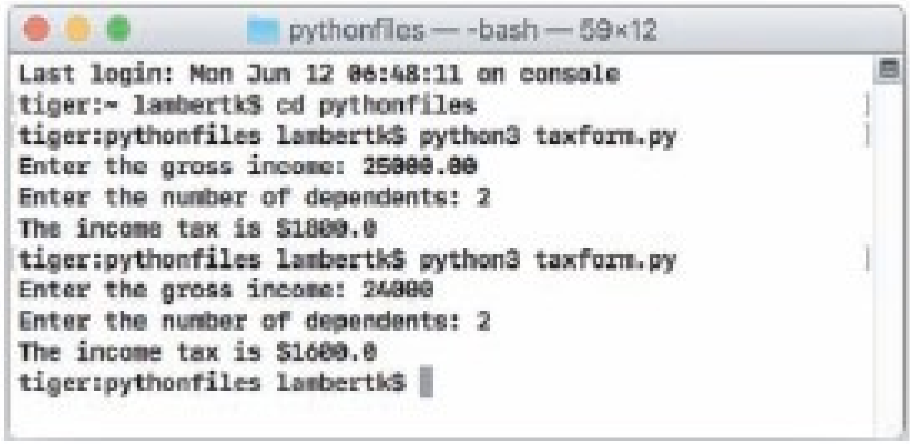
1. A CUI program maintains constant control over the interactions with the user.

2. Prompts users to enter successive inputs

3. The user is constrained to reply to a definite sequence of prompts for inputs

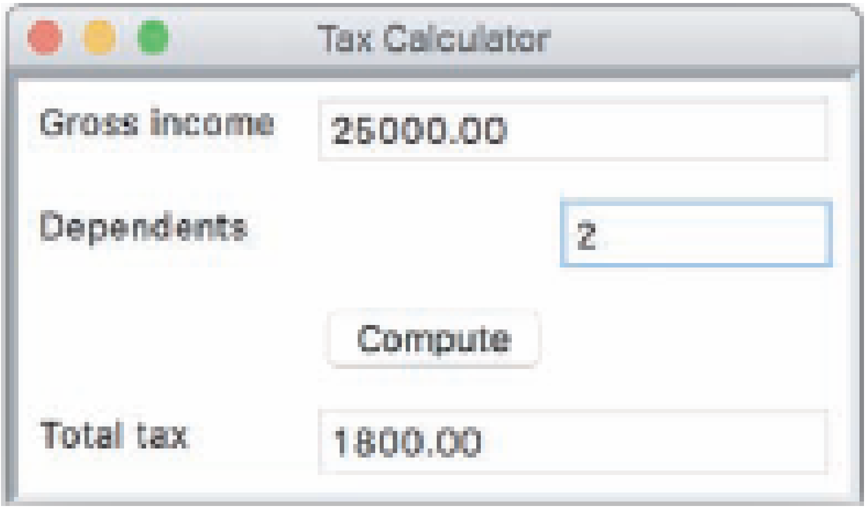
4. Once an input is entered, there is no way to back up and change it.
5. To obtain results for a different set of input data, the user must run the program again.

6. At that point, all of the inputs must be re-entered.



```
Last login: Mon Jun 12 06:48:11 on console
tiger:~ lambertk$ cd pythonfiles
tiger:pythonfiles lambertk$ python3 taxform.py
Enter the gross income: 25000.00
Enter the number of dependents: 2
The income tax is $1800.0
tiger:pythonfiles lambertk$ python3 taxform.py
Enter the gross income: 24000
Enter the number of dependents: 2
The income tax is $1600.0
tiger:pythonfiles lambertk$ 
```

A session with the terminal-based tax calculator program



Gross income 25000.00

Dependents 2

Total tax 1800.00

- **Window and widgets**
- **Title bar**
- **Labels**
- **entry fields**
- **command button**

A GUI-based tax calculator program

Event-driven programming

Type of programming used to create user-generated events (mouse clicks, pulling in inputs) processing them, displaying results is called event-driven programming.

GUI-based programs are almost always object based.

1. Define a new class to represent the main application window.

2. Instantiate the classes of window components such as labels, fields, and command buttons.

3. Position components in the window.

4. Register a method with each window component in which an event relevant to the application might occur.

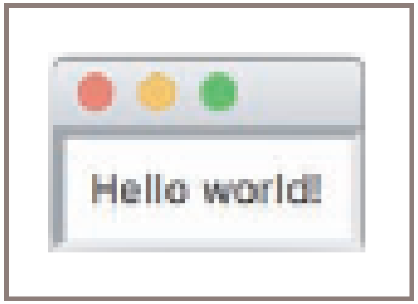
5. Define methods to handle the events.

6. Define a main function that instantiates the window class and runs the appropriate method to launch the GUI.

Coding simple GUI-based programs

Python’s standard tkinter module includes classes for windows and numerous types of window components,

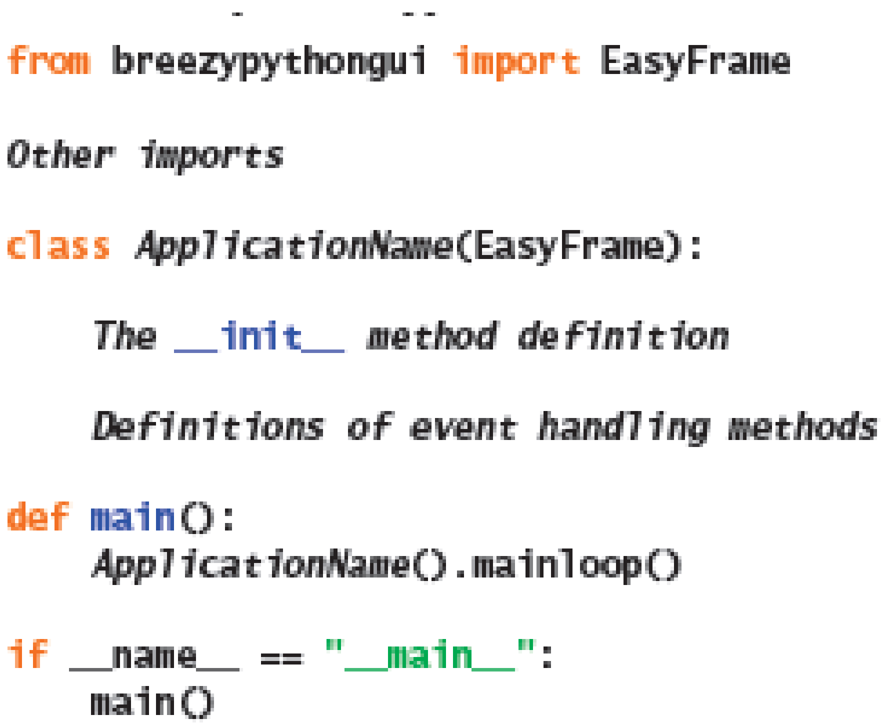
A Simple “Hello World” Program



Hello world!

[{code}]

A Template for All GUI Programs



```
from breezypythongui import EasyFrame

Other imports

class ApplicationName(EasyFrame):

    The __init__ method definition

    Definitions of event handling methods

def main():
    ApplicationName().mainloop()

if __name__ == "__main__":
    main()
```

__init__ method initializes the window by setting its attributes and populating it with the appropriate GUI components.

Python runs this method automatically when the constructor function is called.

The Syntax of Class and Method Definitions

Syntax of class and method definitions has a one-line header that begins with a keyword (class or def).

Followed by a body of code indented one level in the text.

A class header contains the name of the class.

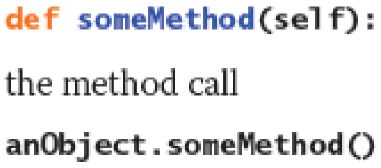
Followed by a parenthesized list of one or more parent classes.

The body of a class definition, nested one tab under the header, consists of one or more method definitions.

Method always has at least one parameter, in the first position, named self.

At call time, the PVM automatically assigns to this parameter a reference to the object on which the method is called.

No need to pass this object as an explicit argument at call time.



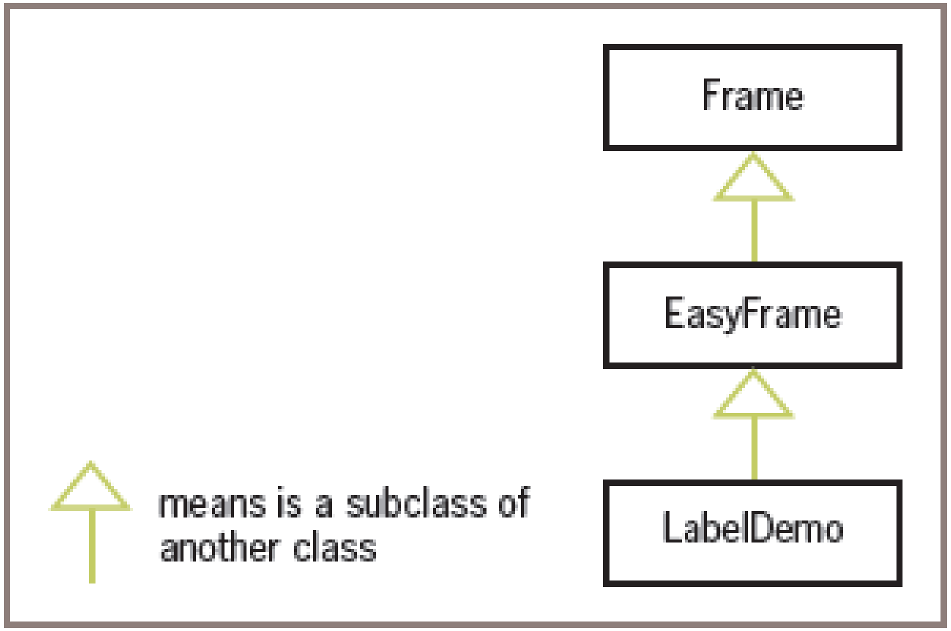
```
def someMethod(self):
```

the method call

```
anObject.someMethod()
```

The parameter self is used within class and method definitions to call other methods on the same object, or to access that object's instance variables or data

Sub classing and Inheritance as Abstraction Mechanisms



EasyFrame class is the parent of the LabelDemo class.

Frame class is the parent of the EasyFrame class.

Frame class is the ancestor of the LabelDemo class.

A new class a subclass of another class.

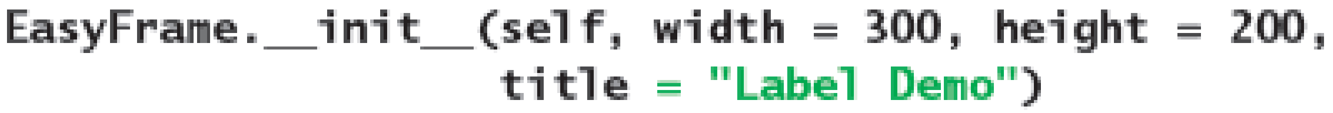
New class inherits and thereby acquires the attributes and behaviour defined by its parent class, and any of its ancestor classes.

Windows and Window Components

Windows and Their Attributes

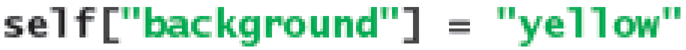
A window has several attributes.

1. title (an empty string by default)
2. width and height in pixels
3. resizability (true by default)
4. background color (white by default)



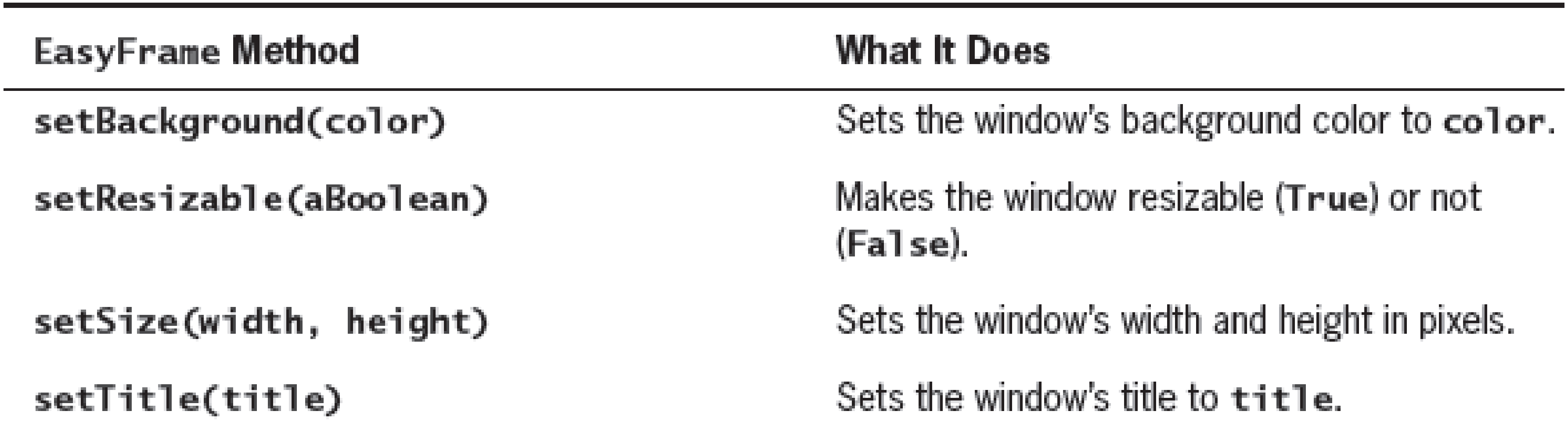
```
EasyFrame.__init__(self, width = 300, height = 200,
                   title = "Label Demo")
```

Override window’s default title, an empty string, by supplying another string as an optional title argument to the EasyFrame method __init__.



```
self["background"] = "yellow"
```

The window’s background color can be set to yellow



EasyFrame Method	What It Does
setBackground(color)	Sets the window's background color to color.
setResizable(aBoolean)	Makes the window resizable (True) or not (False).
setSize(width, height)	Sets the window's width and height in pixels.
setTitle(title)	Sets the window's title to title.

Window Layout

{Code}

Window components are laid out in the window's two-dimensional grid.

The grid's rows and columns are numbered from the position (0, 0) in the upper left corner of the window.

A window component's row and column position in the grid is specified when the component is added to the window.

Can override the default alignment by including the sticky attribute as a keyword argument when the label is added to the window.

The values of sticky are the strings “N,” “S,” “E,” and “W,” or any combination thereof.

{Code}

labels retain their alignments in the exact center of their grid positions.

window layout involves the spanning of a window component across several grid positions.

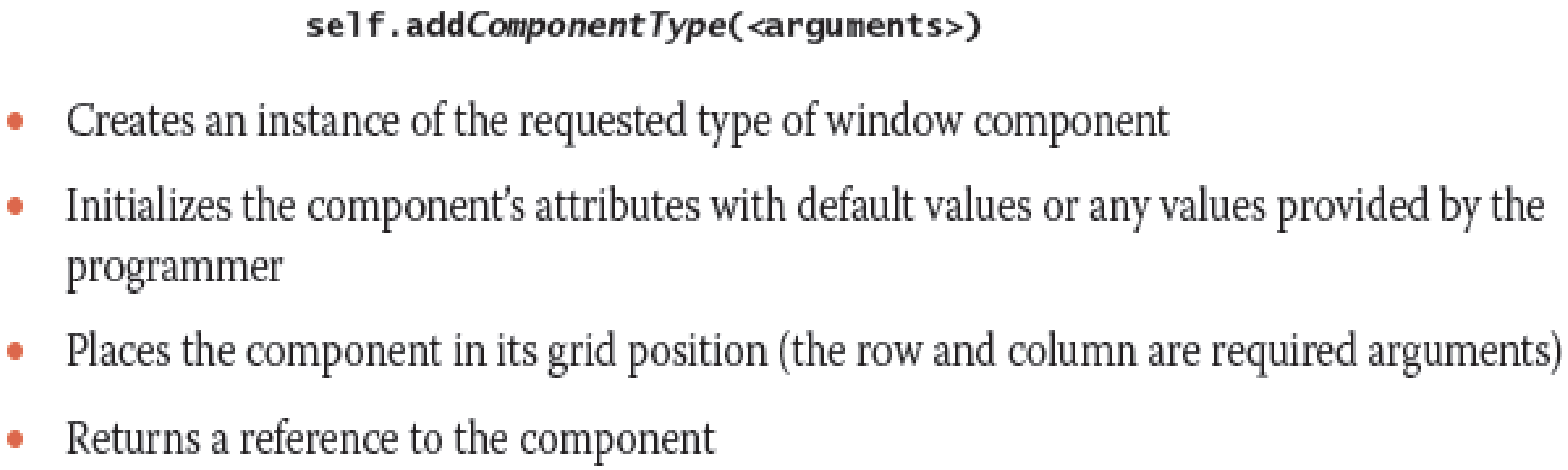
The programmer can force a horizontal and/ or vertical spanning of grid positions by supplying the rowspan and columnspan keyword arguments when adding a component

{Code}

Types of Window Components and Their Attributes

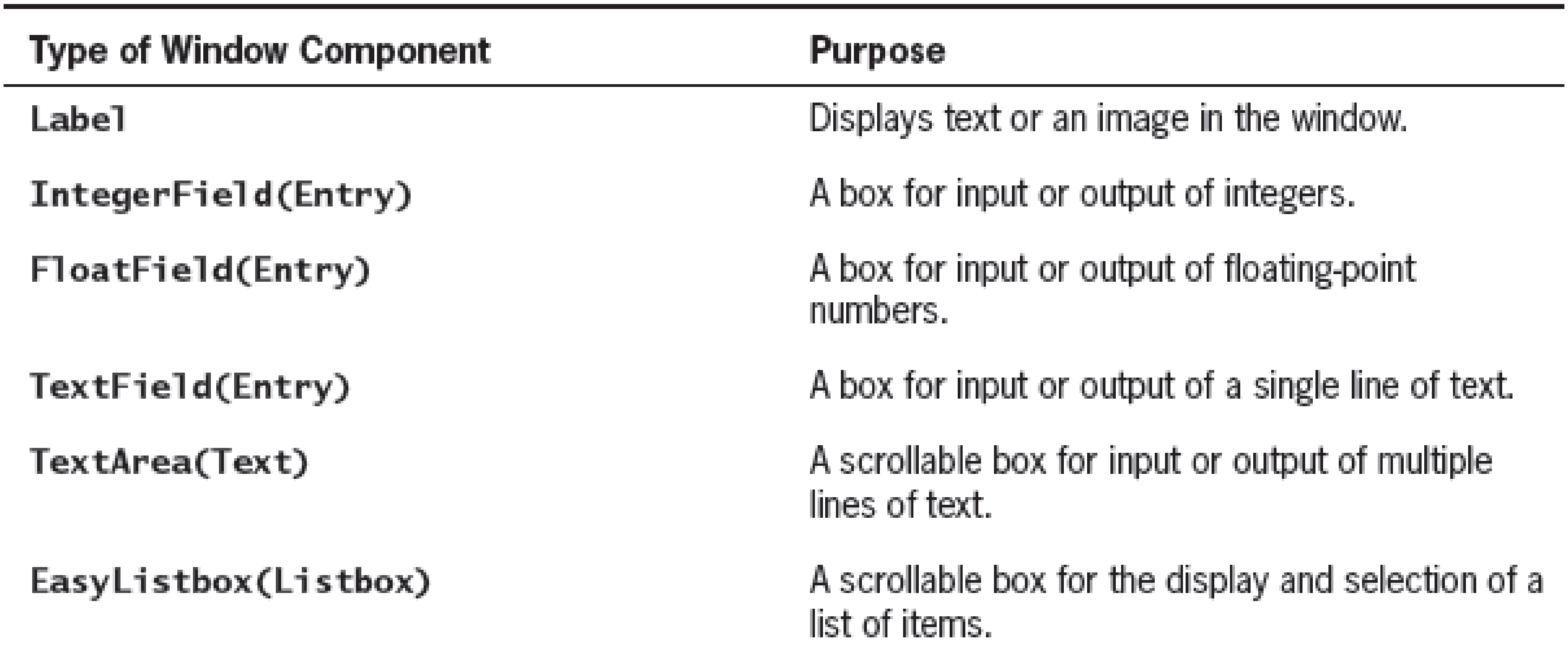
GUI programs use several types of window components, or widgets as they are commonly called.

These include labels, entry fields, text areas, command buttons, drop-down menus, sliding scales, scrolling list boxes, canvases, and many others.



```
self.addComponentType(<arguments>)
```

- Creates an instance of the requested type of window component
- Initializes the component's attributes with default values or any values provided by the programmer
- Places the component in its grid position (the row and column are required arguments)
- Returns a reference to the component

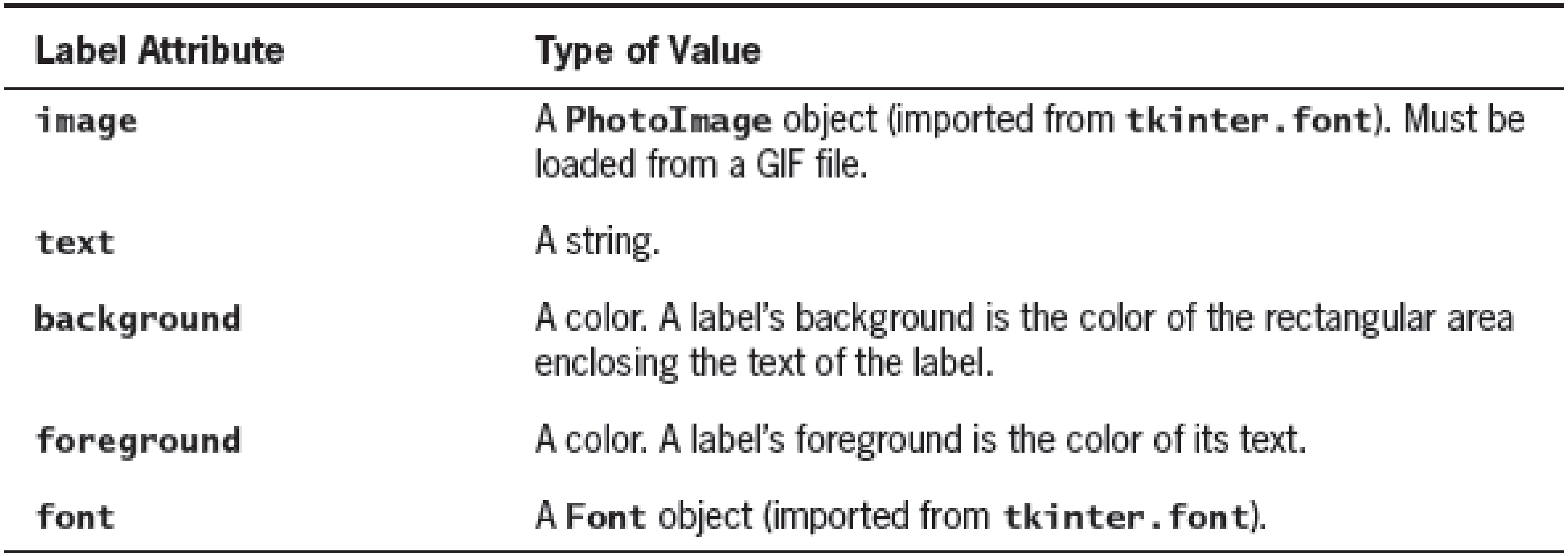


Type of Window Component	Purpose
`Label`	Displays text or an image in the window.
`IntegerField(Entry)`	A box for input or output of integers.
`FloatField(Entry)`	A box for input or output of floating-point numbers.
`TextField(Entry)`	A box for input or output of a single line of text.
`TextArea(Text)`	A scrollable box for input or output of multiple lines of text.
`EasyListbox(Listbox)`	A scrollable box for the display and selection of a list of items.

Type of Window Component	Purpose
Button	A clickable command area.
EasyCheckbutton(Checkbutton)	A labeled checkbox.
Radiobutton	A labeled disc that, when selected, deselects related radio buttons.
EasyRadiobuttonGroup(Frame)	Organizes a set of radio buttons, allowing only one at a time to be selected.
EasyMenuBar(Frame)	Organizes a set of menus.
EasyMenubutton(Menubutton)	A menu of drop-down command options.
EasyMenuItem	An option in a drop-down menu.
Scale	A labeled slider bar for selecting a value from a range of values.
EasyCanvas(Canvas)	A rectangular area for drawing shapes or images.
EasyPanel(Frame)	A rectangular area with its own grid for organizing window components.
EasyDialog(simpleDialog.Dialog)	A resource for defining special-purpose popup windows.

Displaying Images

One label displays the image and the other label displays the caption. {Code}



Label Attribute	Type of Value
`image`	A `PhotoImage` object (imported from `tkinter.font`). Must be loaded from a GIF file.
`text`	A string.
`background`	A color. A label's background is the color of the rectangular area enclosing the text of the label.
`foreground`	A color. A label's foreground is the color of its text.
`font`	A `Font` object (imported from `tkinter.font`).

The `tkinter.Label` attributes

https://docs.python.org/3/library/tkinter.html#moduletkinter

Command Buttons and Responding to Events

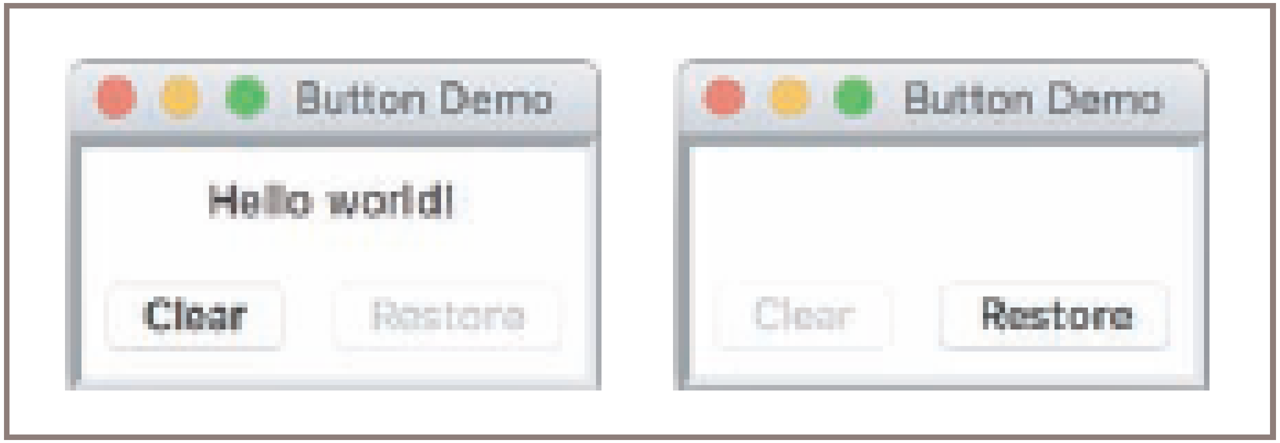
A command button can be added to a window by specifying its text and position in the grid.

A button is centered in its grid position by default.

The method **addButton** accomplishes all this and returns an object of type tkinter.Button.

Button can display an image instead of a string.

A button also has a state attribute, which can be set to “normal” to enable the button (its default state) or “disabled” to disable it.


Button Demo
Hello world!
Clear
Restore
Button Demo
Clear
Restore

code

Input and Output with Entry Fields

An entry field is a box in which the user can position the mouse cursor and enter a number or a single line of text.

Text Fields

A text field is appropriate for entering or displaying a single-line string of characters.

The method addTextField is used to add a text field to a window.

The method returns an object of type TextField, which is a subclass of tkinter.Entry.

Required arguments to addTextField are text (the string to be initially displayed), row, and column.

Optional arguments are rowspan, columnspan, sticky, width, and state.

A text field is aligned by default to the northeast of its grid cell.

A text field has a default width of 20 characters.

This represents the maximum number of characters viewable in the box, but the user can continue typing or viewing them by moving the cursor key to the right.

The programmer can set a text field's state attribute to "readonly" to prevent the user from editing an output field.

The TextField method getText returns the string currently contained in a text field.

The method setText outputs its string argument to a text field.

{code}

Using Pop-Up Message Boxes

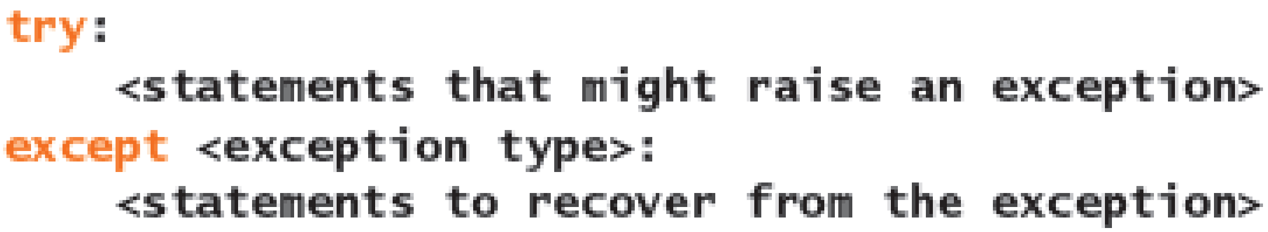
When errors arise in a GUI-based program, the program often responds by popping up a dialog window with an error message.

The program detects the error, pops up the dialog to inform the user, and, when the user closes the dialog, continues to accept and check input data.

In a terminal-based program, this process usually requires an explicit loop structure.

In a GUI-based program, Python’s implicit event-driven loop continues the process automatically

Python’s try-except statement.



```
try:
    <statements that might raise an exception>
except <exception type>:
    <statements to recover from the exception>
```

If an exception is raised anywhere in this process, control shifts immediately to the except clause.

{code}

End

Assignment 2

Write a program to enlarge an image.

Last date : 24/06/22

Mode: Online and Offline