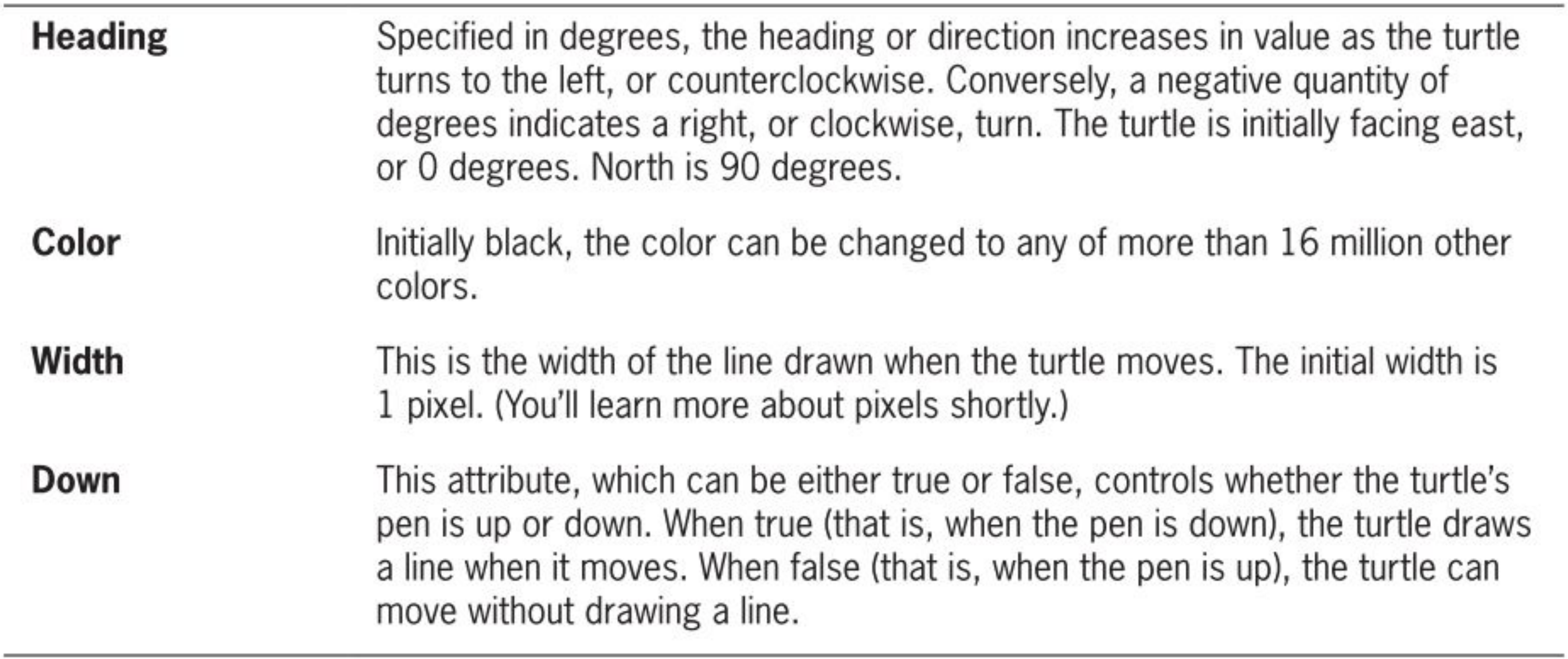
S6 CS Programme Elective
CST362 Programming in Python

Module 3

Overview of Turtle Graphics

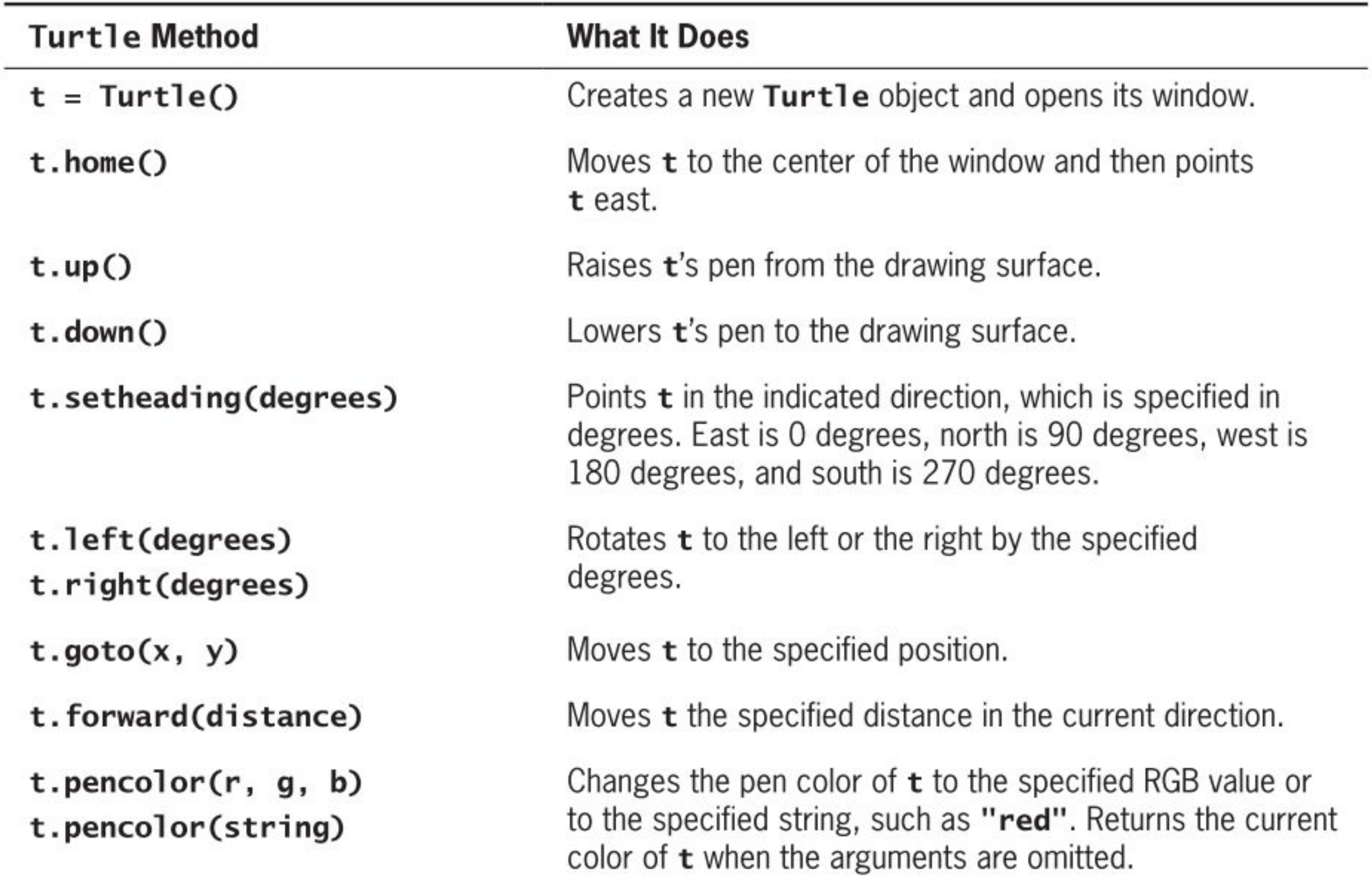
- Turtle is an icon located at a specific position in the window specified with (x,y)
- Initial position is the origin or home
- an important attribute is heading or the direction in which it currently faces
- Initial heading is 0 degrees due east on its map
- Attributes make up a turtle’s state which determines how the turtle will behave when any operations are applied

Some attributes of a Turtle



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.

Turtle Methods



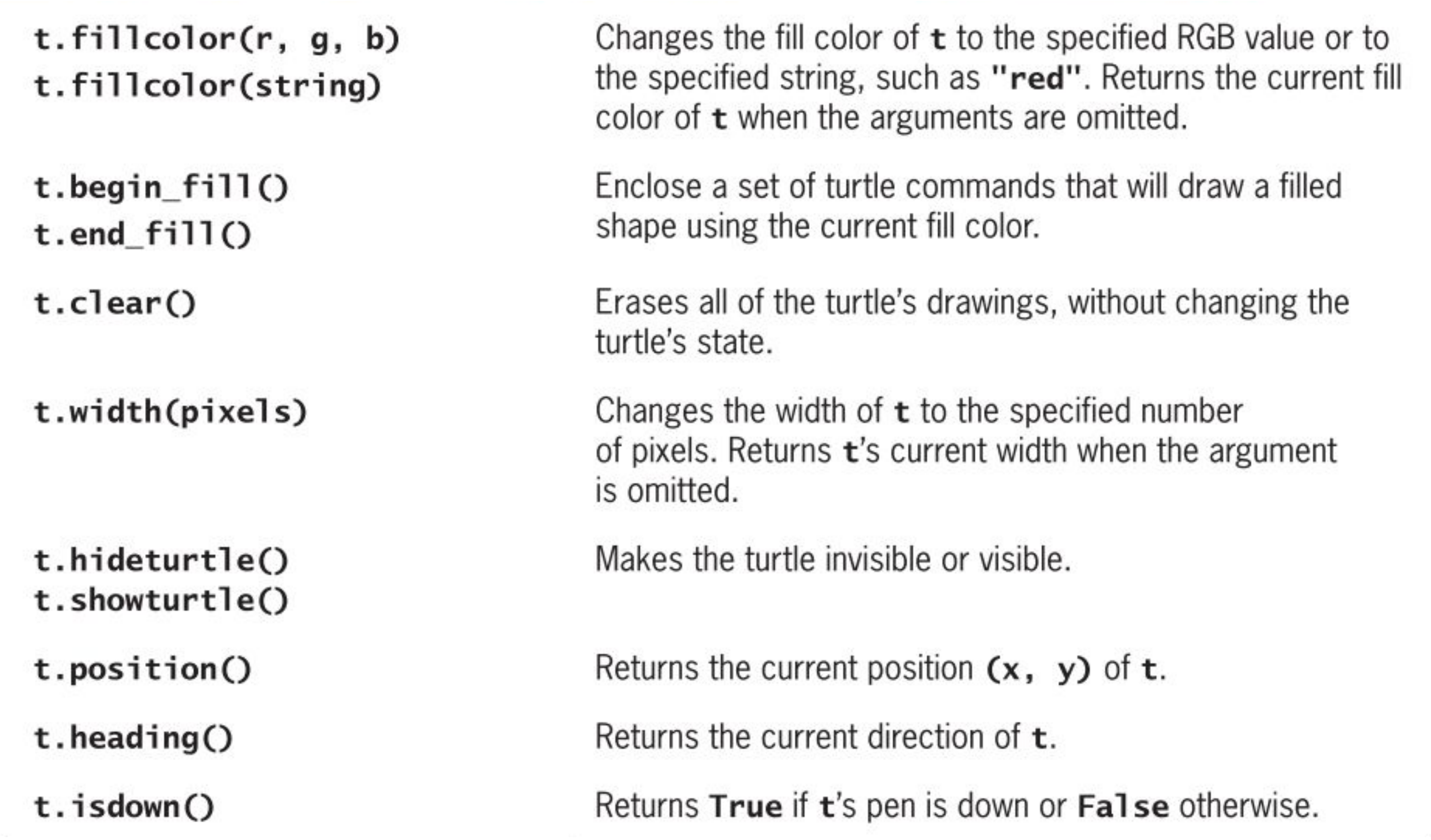
Turtle Method	What It Does
`t = Turtle()`	Creates a new **Turtle** object and opens its window.
`t.home()`	Moves **t** to the center of the window and then points **t** east.
`t.up()`	Raises **t**'s pen from the drawing surface.
`t.down()`	Lowers **t**'s pen to the drawing surface.
`t.setheading(degrees)`	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.

Turtle Methods



`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.

```
>>> from turtle import Turtle
>>> t = Turtle()
```

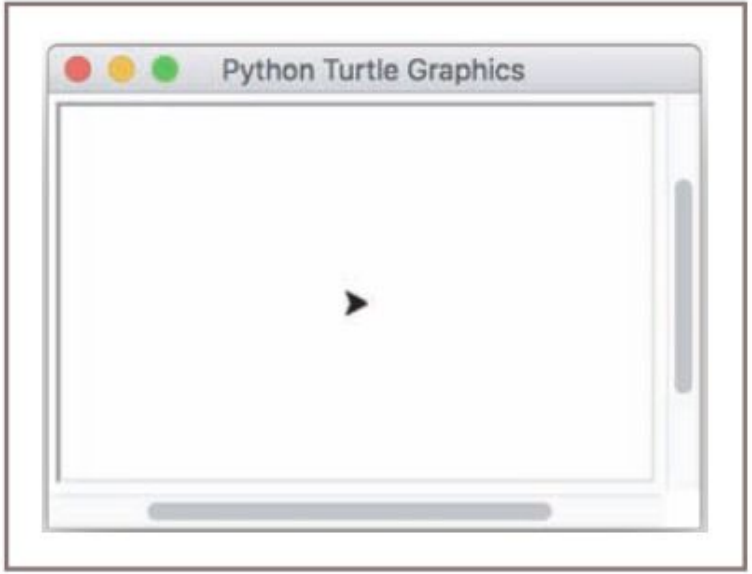
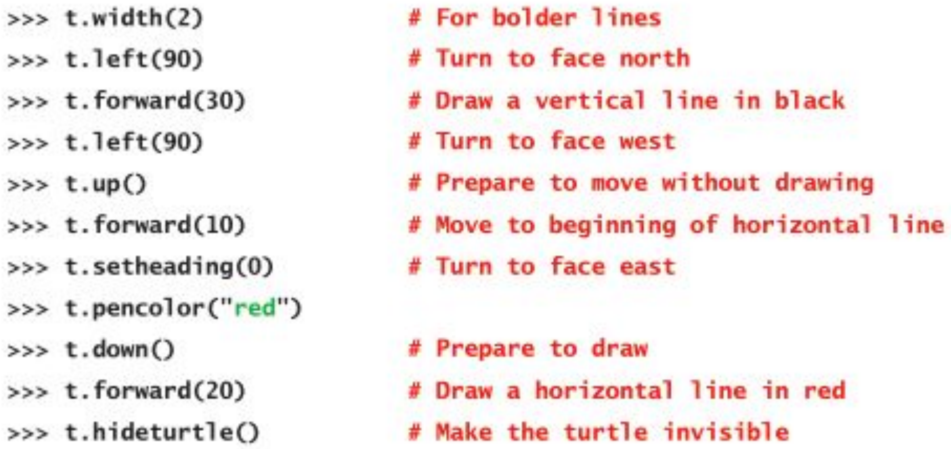


Figure 7-1 Drawing window for a turtle

Draw T shape

```
>>> from turtle import Turtle

>>> t = Turtle()
```



```
>>> 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
```

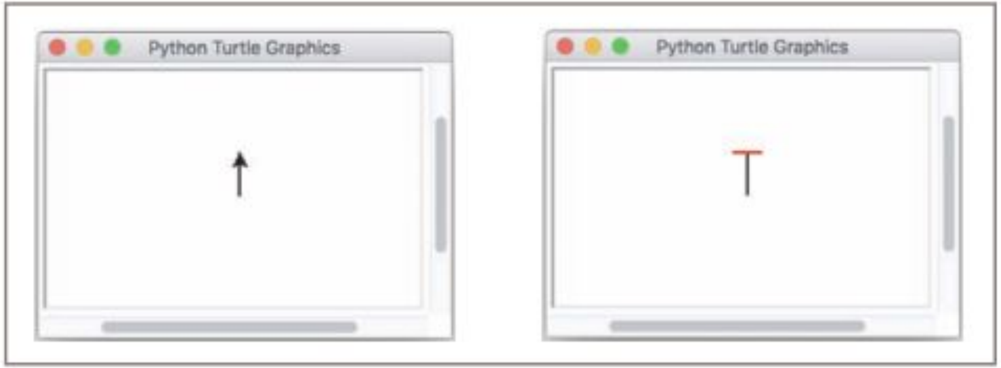
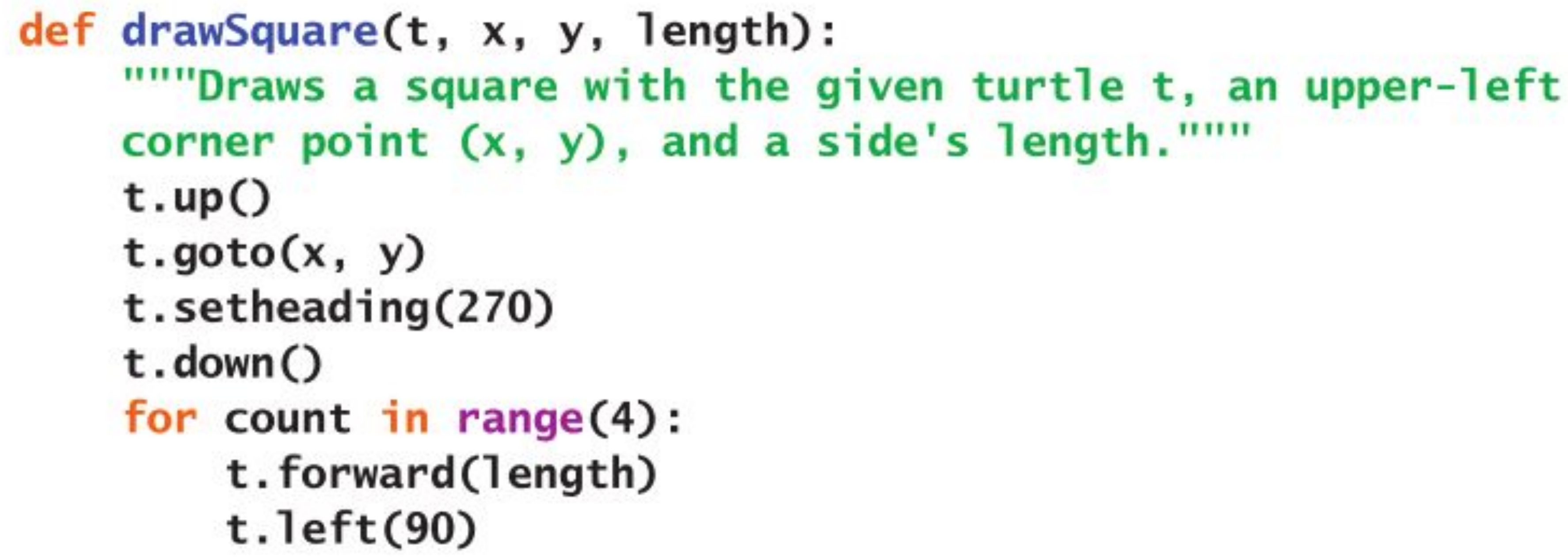


Figure 7-2 Drawing vertical and horizontal lines for the letter T

Draw Square



```python
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)
```

Image Processing

- Analog and Digital Information - range of values and discrete values
- Early recording and playback devices for images and sound were all analog devices
- Continuous analog information in a real visual scene must be mapped into a set of discrete values.
- This conversion process involves sampling

Sampling and Digitizing Images

- A visual scene projects an infinite set of color and intensity values onto a two-dimensional sensing medium, such as a human being’s retina or a scanner’s surface
- digital information can represent an image that is more or less indistinguishable to the human eye from the original scene
- Sampling devices measure discrete color values at distinct points on a two-dimensional grid. These values are pixels

Sampling and Digitizing Images

- the more pixels that are sampled, the more continuous and realistic the resulting image will appear
- the human eye cannot discern objects that are closer together than 0.1 mm, so a sampling of 10 pixels per linear millimeter (250 pixels per inch and 62,500 pixels per square inch) would be plenty accurate.
- Thus, a 3-inch by 5-inch image would need 3 *5 *62,500 pixels/inch 937,500 pixels

Image File Formats

- A raw image file saves all of the sampled information
- This has a cost and a benefit: The benefit is that the display of a raw image will be the most true to life, but the cost is that the file size of the image can be quite large
- Two of the most popular image file formats are JPEG (Joint Photographic Experts Group) and GIF (Graphic Interchange Format)

Image File Formats

- data-compression schemes are used to reduce the file size of a JPEG image
- If any color values are the same, their positions rather than their values are stored, thus potentially saving many bits of storage
- Before the image is displayed, the original color values are restored during the process of decompression
- This scheme is called lossless compression, meaning that no information is lost

Image File Formats

- another scheme analyzes larger regions of pixels and saves a color value that the pixels’ colors approximate
- This is called a lossy scheme, meaning that some of the original color information is lost
- human eye usually is not able to detect the difference between the new colors and the original ones

Image File Formats

- A GIF image relies on an entirely different compression scheme
- The compression algorithm consists of two phases
- first phase, the algorithm analyzes the color samples to build a table, or color palette, of up to 256 of the most prevalent colors
- The algorithm then visits each sample in the grid and replaces it with the key of the closest color in the color palette
- The resulting image file thus consists of at most 256 color values and the integer keys of the image’s colors in the palette

Image File Formats

- This strategy can potentially save a huge number of bits of storage
- The decompression algorithm uses the keys and the color palette to restore the grid of pixels for display
- Although GIF uses a lossy compression scheme, it works very well for images with broad, flat areas of the same color, such as cartoons, backgrounds, and banners

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Image-Manipulation Operations

- either transform the information in the pixels or alter the arrangement of the pixels in the image
  - Rotate an image
  - Convert an image from color to grayscale
  - Apply color 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 color inversion to an image
  - Morph an image into another image

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The Properties of Images

- When an image is loaded into a program such as a Web browser, the software maps the bits from the image file into a rectangular area of colored pixels for display
- The coordinates of the pixels in this two-dimensional grid range from (0, 0) at the upper-left corner of an image to (width – 1, height – 1) at the lower-right corner, where width and height are the image’s dimensions in pixels
- Thus, the screen coordinate system for the display of an image is somewhat different from the standard Cartesian coordinate system that we used with Turtle graphics, where the origin (0, 0) is at the center of the rectangular grid

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The images Module

- allows the programmer to load an image from a file, view the image in a window, examine and manipulate an image’s RGB values, and save the image to a file
- a non-standard, open-source Python tool
- includes a class named Image
- The Image class represents an image as a two-dimensional grid of RGB values

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The images Module

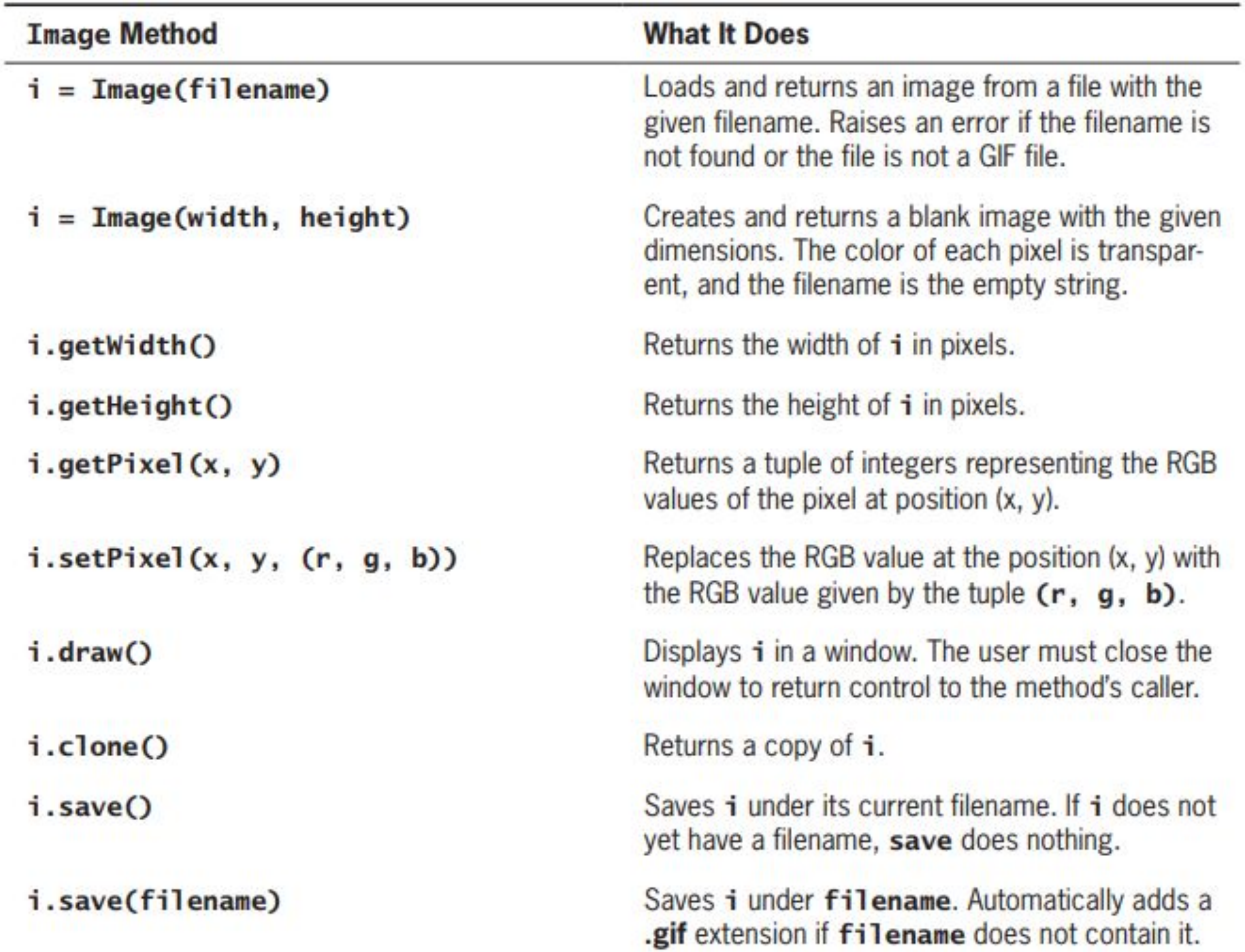
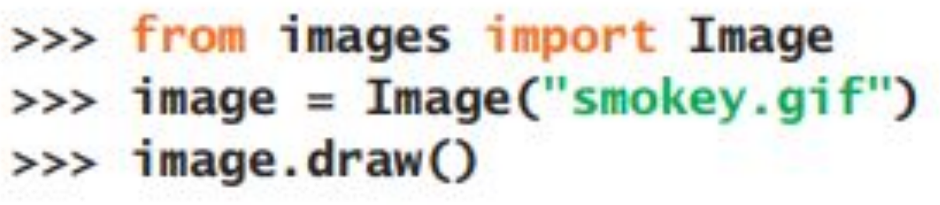


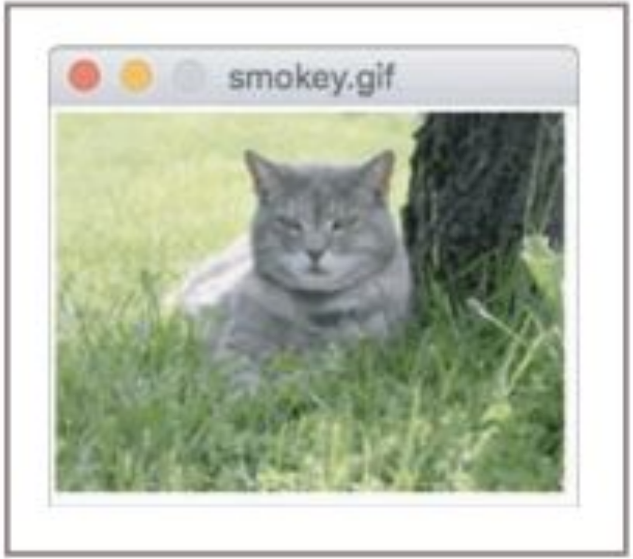
Image Method	What It Does
`i = Image(filename)`	Loads and returns an image from a file with the given filename. Raises an error if the filename is not found or the file is not a GIF file.
`i = Image(width, height)`	Creates and returns a blank image with the given dimensions. The color of each pixel is 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.

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The images Module

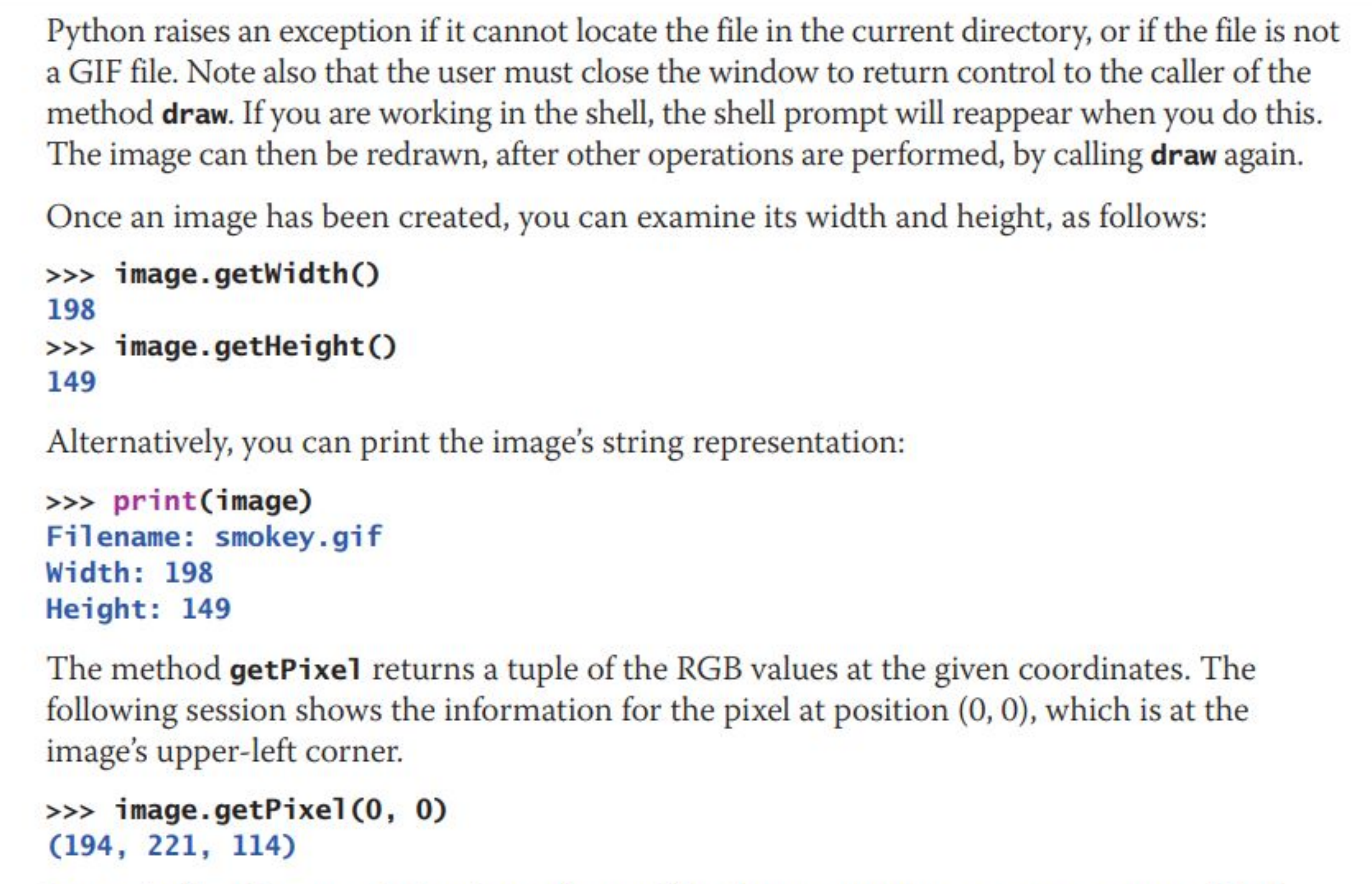


```
>>> from images import Image
>>> image = Image("smokey.gif")
>>> image.draw()
```



Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The images Module



Python raises an exception if it cannot locate the file in the current directory, or if the file is not a GIF file. Note also that the user must close the window to return control to the caller of the method **draw**. If you are working in the shell, the shell prompt will reappear when you do this. The image can then be redrawn, after other operations are performed, by calling **draw** again.

Once an image has been created, you can examine its width and height, as follows:

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

Alternatively, you can print the image's string representation:

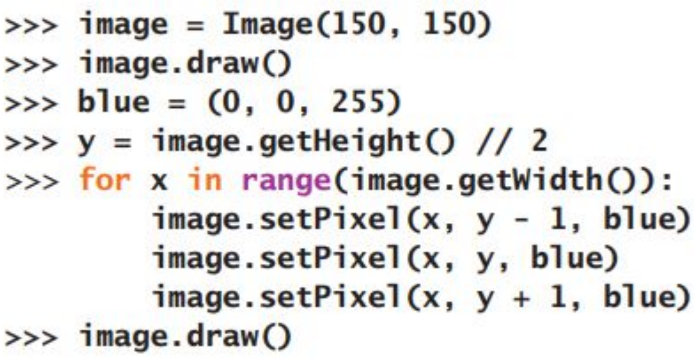
```
>>> print(image)
Filename: smokey.gif
Width: 198
Height: 149
```

The method **getPixel** returns a tuple of the RGB values at the given coordinates. The following session shows the information for the pixel at position (0, 0), which is at the image's upper-left corner.

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

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

The images Module



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

```
>>> image.save("horizontal.gif")
```

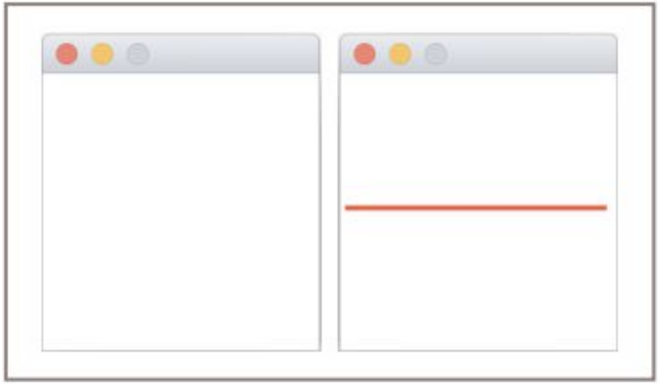
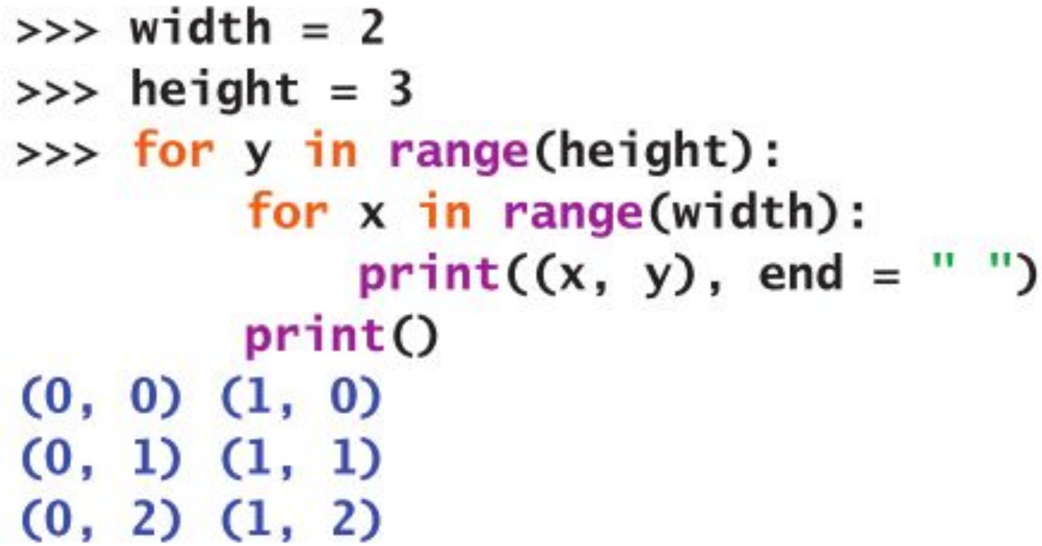


Figure 7-10 An image before and after replacing the pixels

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Image Grid

- Uses nested loop structure to traverse 2 dimensional grid



```
>>> 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)
```

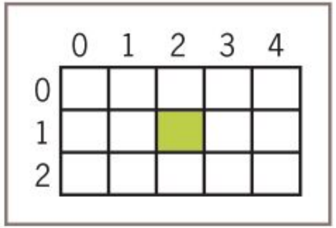
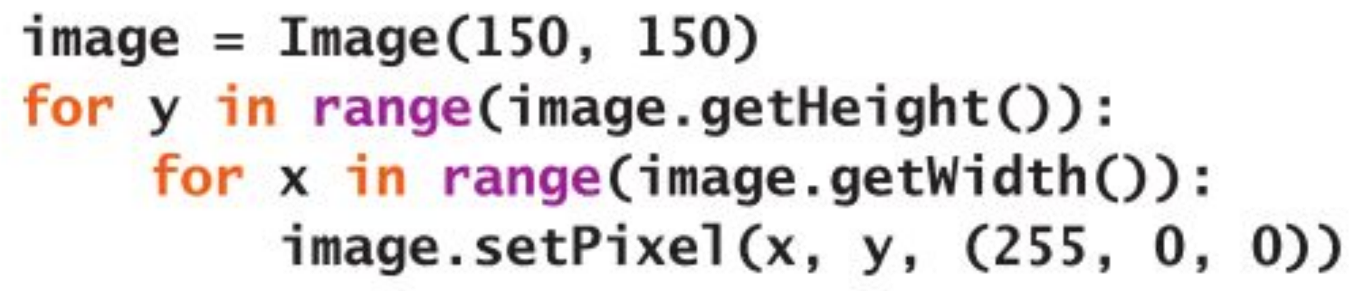


Figure 7-11 A grid with 3 rows and 5 columns

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Image Grid

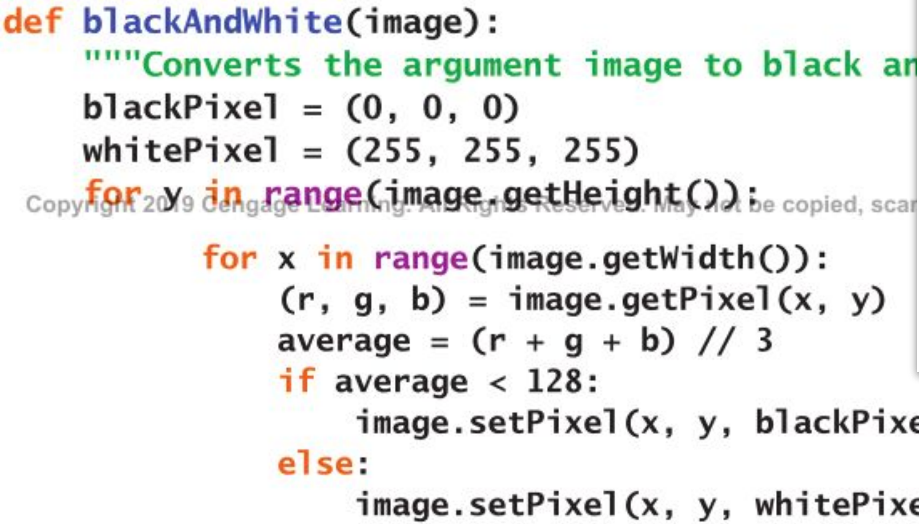
- Uses nested loop structure to fill a blank image with red



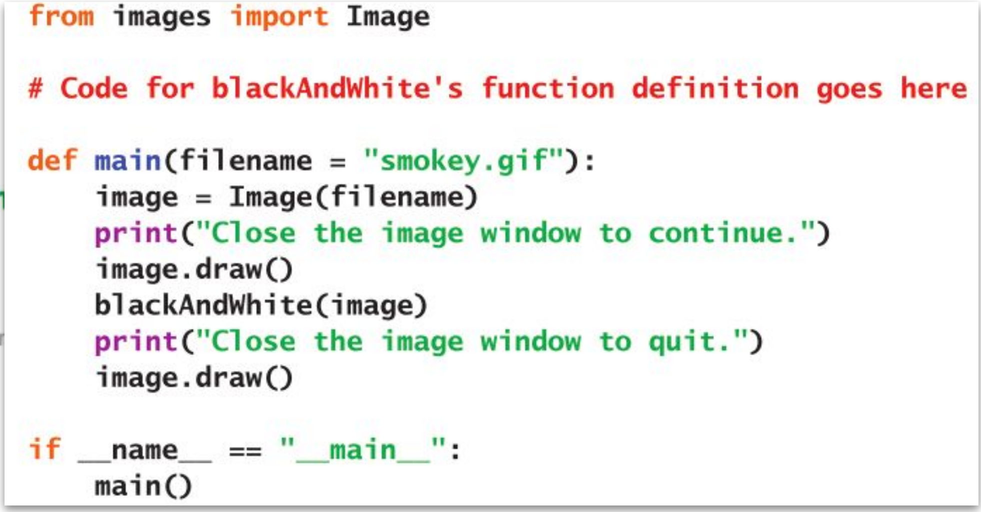
```
image = Image(150, 150)
for y in range(image.getHeight()):
    for x in range(image.getWidth()):
        image.setPixel(x, y, (255, 0, 0))
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

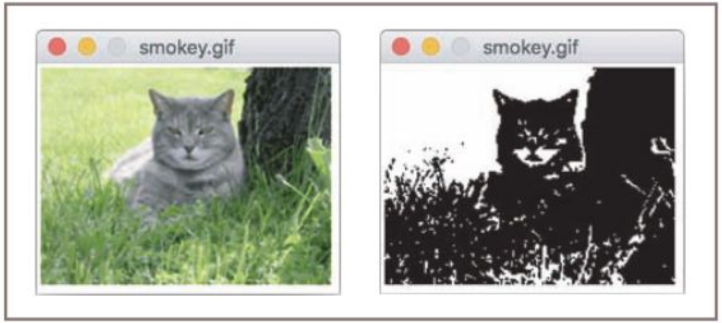
Color Image to black & white



```
def blackAndWhite(image):
    """Converts the argument image to black an
    blackPixel = (0, 0, 0)
    whitePixel = (255, 255, 255)
    for y in range(image.getHeight()):
        for x in range(image.getWidth()):
            (r, g, b) = image.getPixel(x, y)
            average = (r + g + b) // 3
            if average < 128:
                image.setPixel(x, y, blackPixel)
            else:
                image.setPixel(x, y, whitePixel)
```



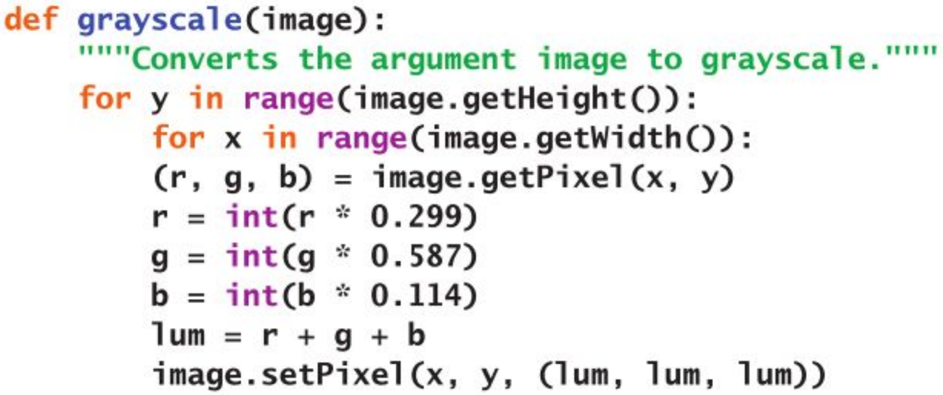
```
from images import Image

# Code for blackAndWhite's function definition goes here

def main(filename = "smokey.gif"):
    image = Image(filename)
    print("Close the image window to continue.")
    image.draw()
    blackAndWhite(image)
    print("Close the image window to quit.")
    image.draw()

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



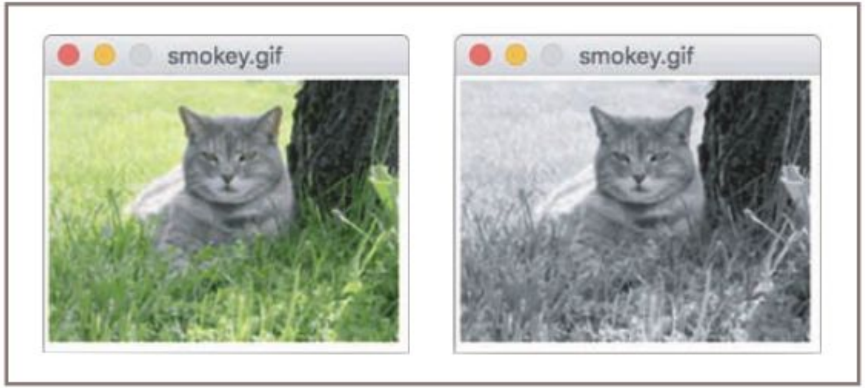
Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Color Image to Grayscale

- Human eye is more sensitive to red than blue
- Psychologists determined the relative propositions of RGB as 0.299, 0.587 and 0.114

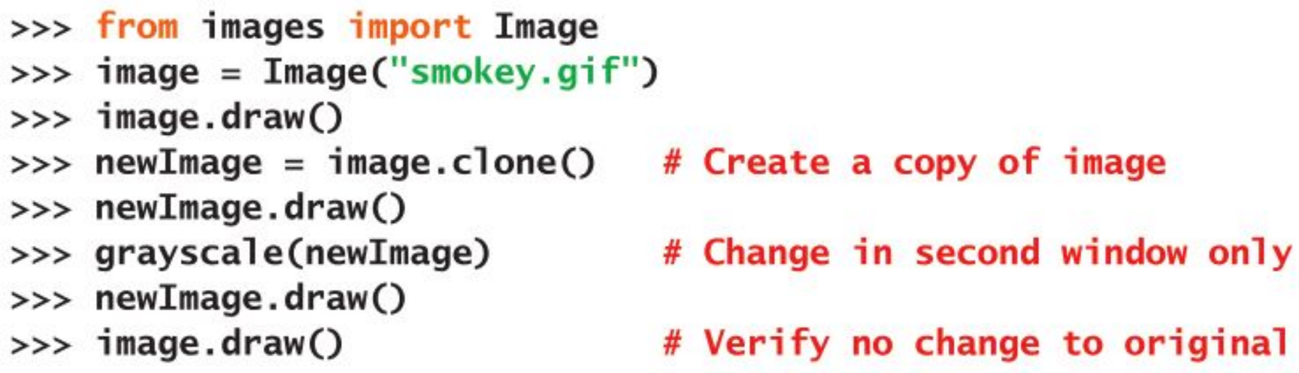


```python
def grayscale(image):
    """Converts the argument image to grayscale."""
    for y in range(image.getHeight()):
        for x in range(image.getWidth()):
        (r, g, b) = image.getPixel(x, y)
        r = int(r * 0.299)
        g = int(g * 0.587)
        b = int(b * 0.114)
        lum = r + g + b
        image.setPixel(x, y, (lum, lum, lum))
```



Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Copying an image



```
>>> 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
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

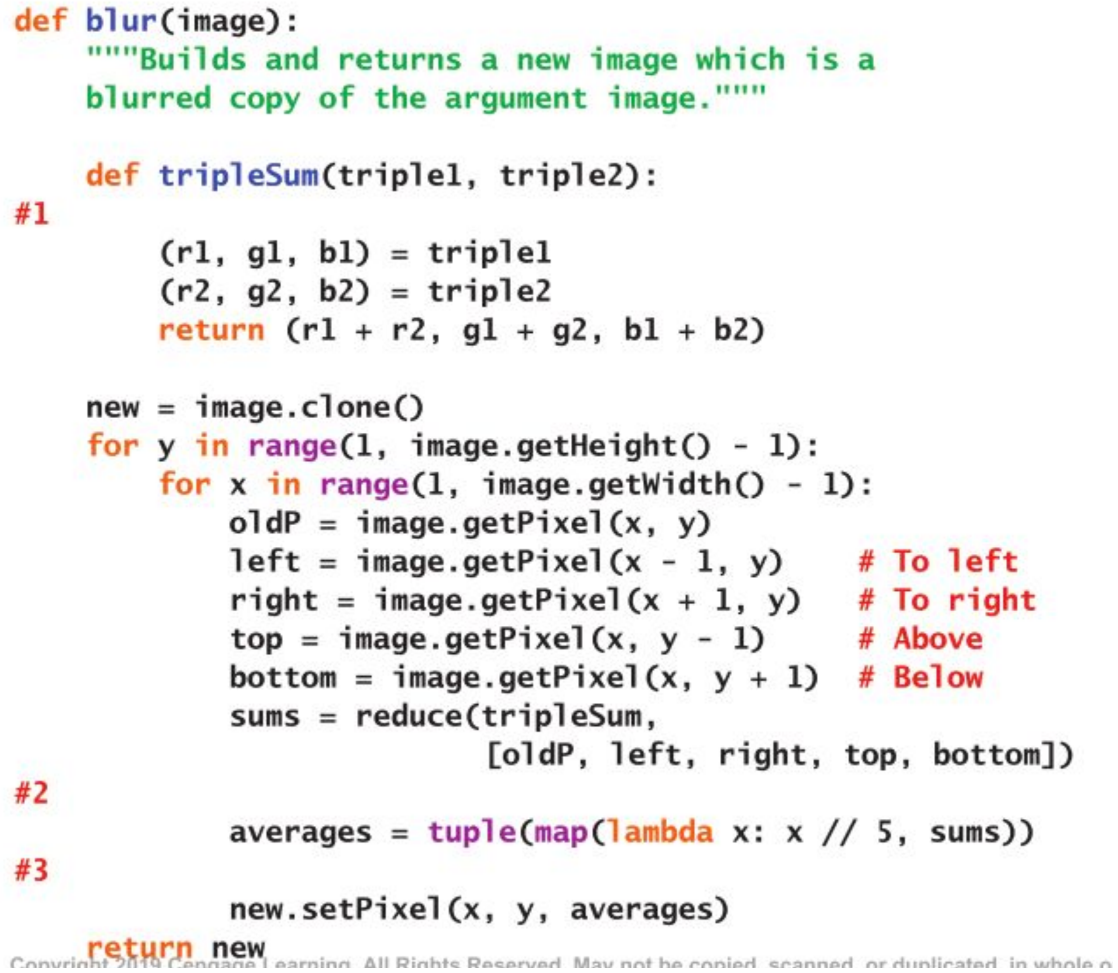
Image Blurring

- an image appears to contain rough, jagged edges, this condition, known as pixelation, can be mitigated by blurring the image
- Blurring makes these areas appear softer
- Resets each pixel’s color to the average of the four pixels surround it
- Traverse from (1,1) to (width-2,height-2)

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Blurring an image

(0,0)	(0,1)	(0,2)
(1,0)	(1,1) (x,y)	(1,2)
(2,0)	(2,1)	(2,2)

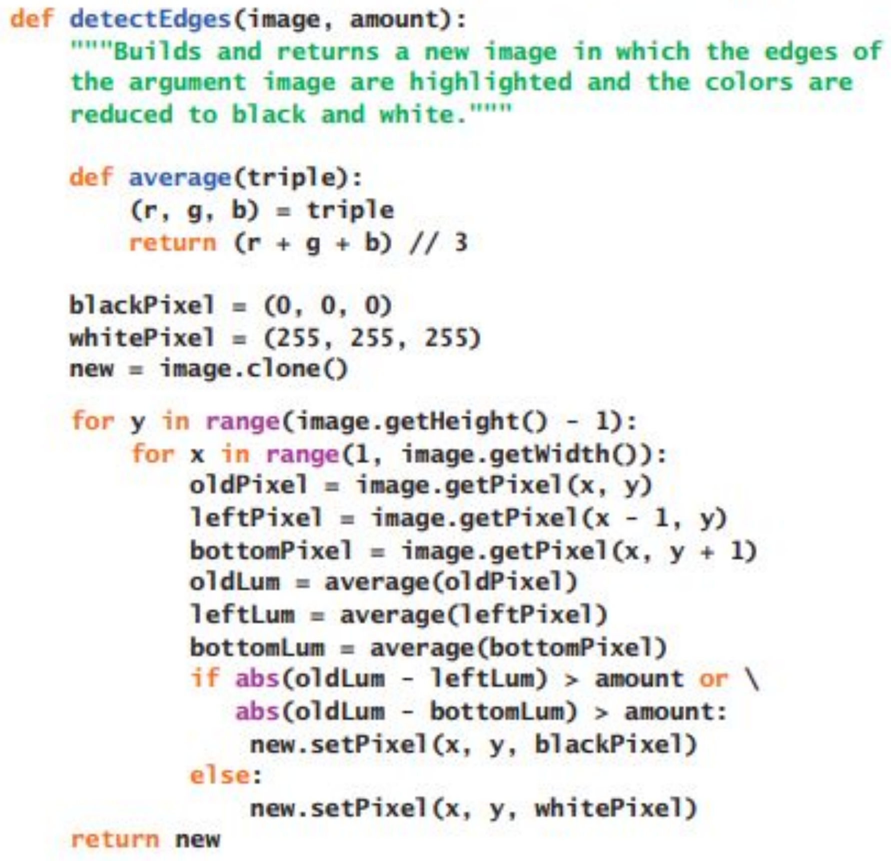


```
def blur(image):
    """Builds and returns a new image which is a
    blurred copy of the argument image."""

    def tripleSum(triple1, triple2):
#1
        (r1, g1, b1) = triple1
        (r2, g2, b2) = triple2
        return (r1 + r2, g1 + g2, b1 + b2)

    new = image.clone()
    for y in range(1, image.getHeight() - 1):
        for x in range(1, image.getWidth() - 1):
            oldP = image.getPixel(x, y)
            left = image.getPixel(x - 1, y)   # To left
            right = image.getPixel(x + 1, y)  # To right
            top = image.getPixel(x, y - 1)    # Above
            bottom = image.getPixel(x, y + 1) # Below
            sums = reduce(tripleSum,
                          [oldP, left, right, top, bottom])
#2
            averages = tuple(map(lambda x: x // 5, sums))
#3
            new.setPixel(x, y, averages)
    return new
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Edge Detection

- removes the full colors to uncover the outlines of the objects represented in the image
- simple edge-detection algorithm examines the neighbors below and to the left of each pixel in an image
- If the luminance of the pixel differs from that of either of these two neighbors by a significant amount, you have detected an edge, and you set that pixel’s color to black. Otherwise, you set the pixel’s color to white



```
def detectEdges(image, amount):
    """Builds and returns a new image in which the edges of
    the argument image are highlighted and the colors are
    reduced to black and white."""

    def average(triple):
        (r, g, b) = triple
        return (r + g + b) // 3

    blackPixel = (0, 0, 0)
    whitePixel = (255, 255, 255)
    new = image.clone()

    for y in range(image.getHeight() - 1):
        for x in range(1, image.getWidth()):
            oldPixel = image.getPixel(x, y)
            leftPixel = image.getPixel(x - 1, y)
            bottomPixel = image.getPixel(x, y + 1)
            oldLum = average(oldPixel)
            leftLum = average(leftPixel)
            bottomLum = average(bottomPixel)
            if abs(oldLum - leftLum) > amount or \
               abs(oldLum - bottomLum) > amount:
                new.setPixel(x, y, blackPixel)
            else:
                new.setPixel(x, y, whitePixel)
    return new
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Edge Detection

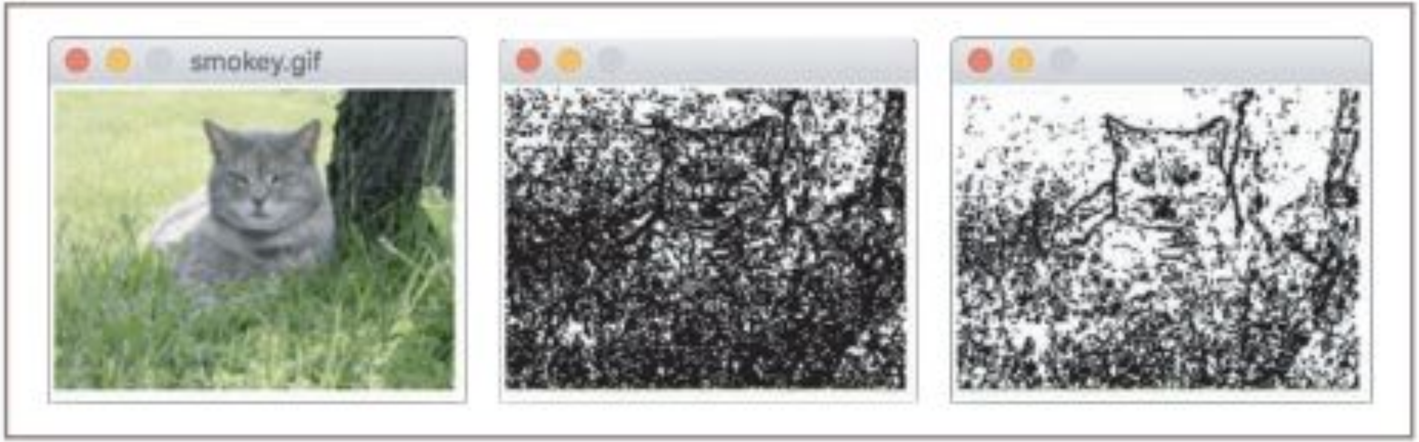
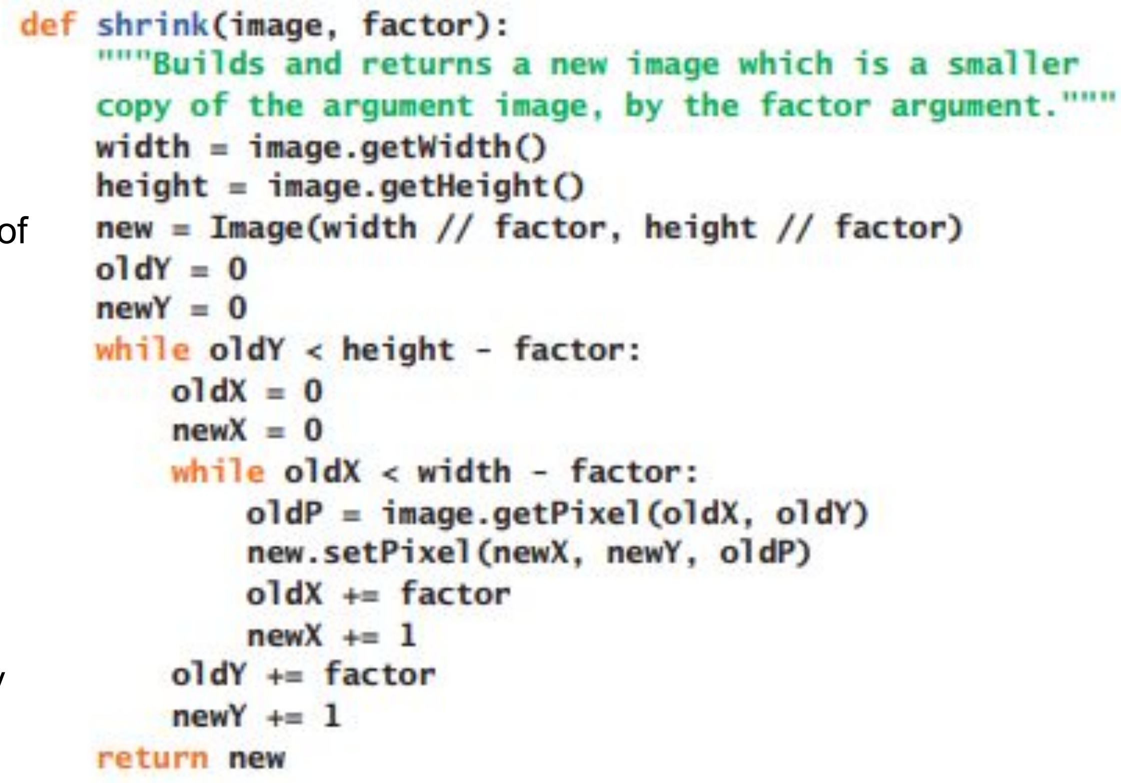


Figure 7-14 Edge detection: the original image, a luminance threshold of 10, and a luminance threshold of 20

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Reducing image size

- A size reduction usually preserves an image’s aspect ratio (that is, the ratio of its width to its height)
- simple way to shrink an image is to create a new image whose width and height are a constant fraction of the original image’s width and height
- The algorithm then copies the color values of just some of the original image’s pixels to the new image
- For example, to reduce the size of an image by a factor of 2, you could copy the color values from every other row and every other column of the original image to the new image



```python
def shrink(image, factor):
    """Builds and returns a new image which is a smaller
    copy of the argument image, by the factor argument."""
    width = image.getWidth()
    height = image.getHeight()
    new = Image(width // factor, height // factor)
    oldY = 0
    newY = 0
    while oldY < height - factor:
        oldX = 0
        newX = 0
        while oldX < width - factor:
            oldP = image.getPixel(oldX, oldY)
            new.setPixel(newX, newY, oldP)
            oldX += factor
            newX += 1
        oldY += factor
        newY += 1
    return new
```

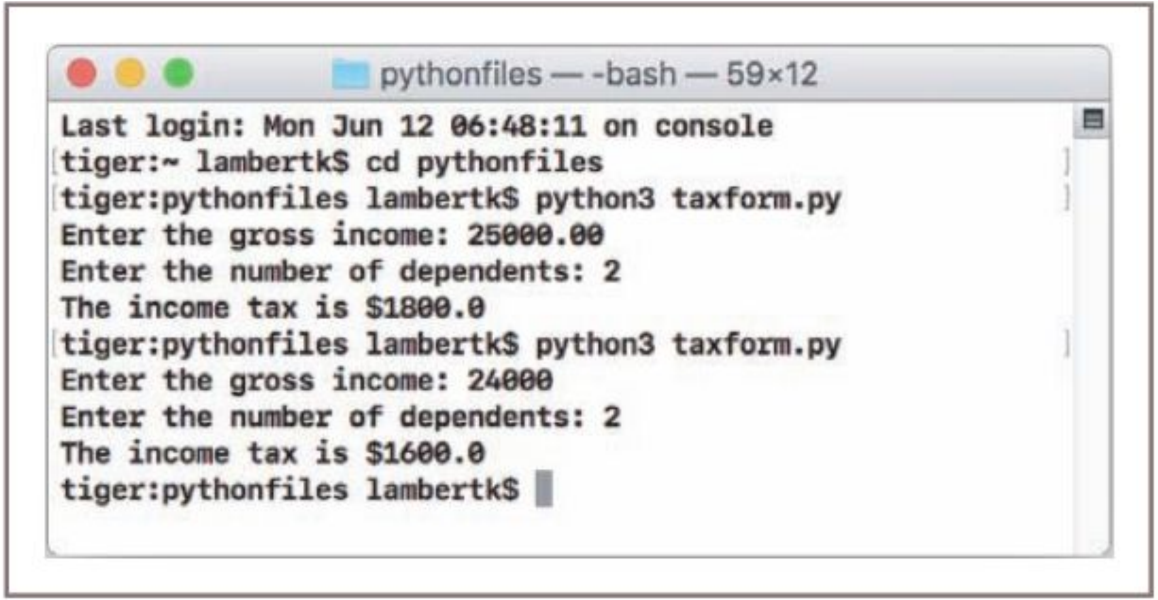
Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Graphical User Interfaces

- Event driven programs
- Inactive until user clicks a button or selects a menu option
- Terminal based program maintains a constant control over the interactions with the user
- terminal-based program prompts users to enter successive inputs, whereas a GUI program puts users in change, allowing them to enter inputs in any order and waiting for them to press a command button or select a menu option

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Graphical User Interfaces

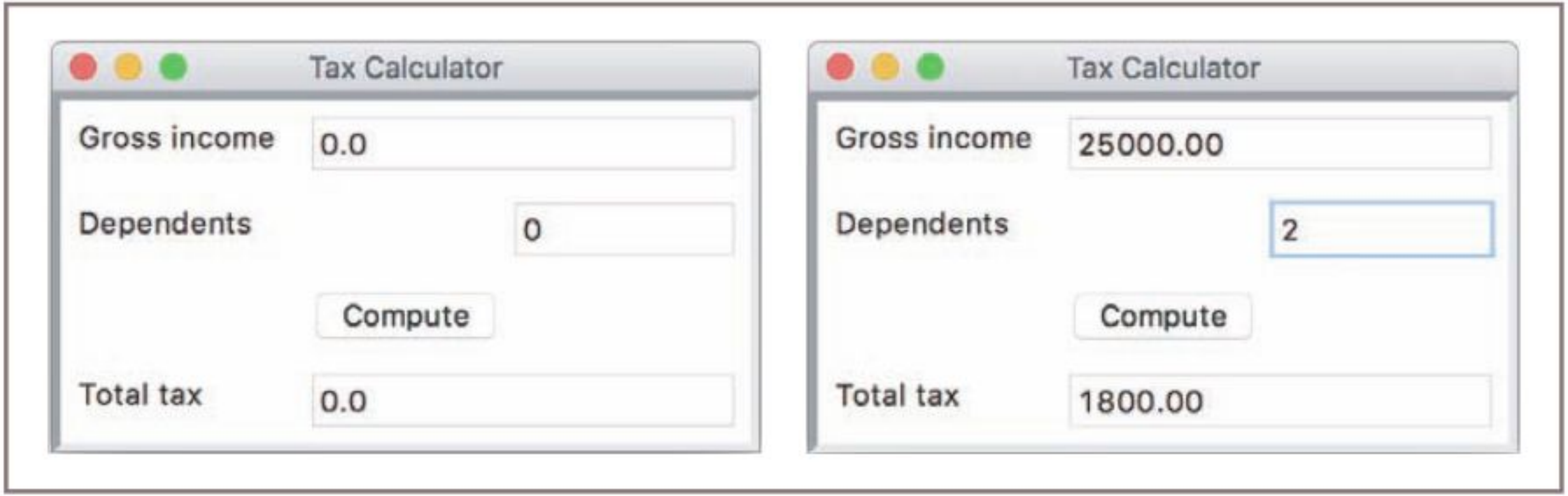


```
Last login: Mon Jun 12 06:48:11 on console
tiger:~ lambertk$ cd pythonfiles
tiger:pythonfiles lambertk$ python3 taxform.py
Enter the gross income: 25000.00
Enter the number of dependents: 2
The income tax is $1800.0
tiger:pythonfiles lambertk$ python3 taxform.py
Enter the gross income: 24000
Enter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

Figure 8-1 A session with the terminal-based tax calculator program

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Graphical User Interfaces



Gross income 0.0

Dependents 0

Total tax 0.0

Gross income 25000.00

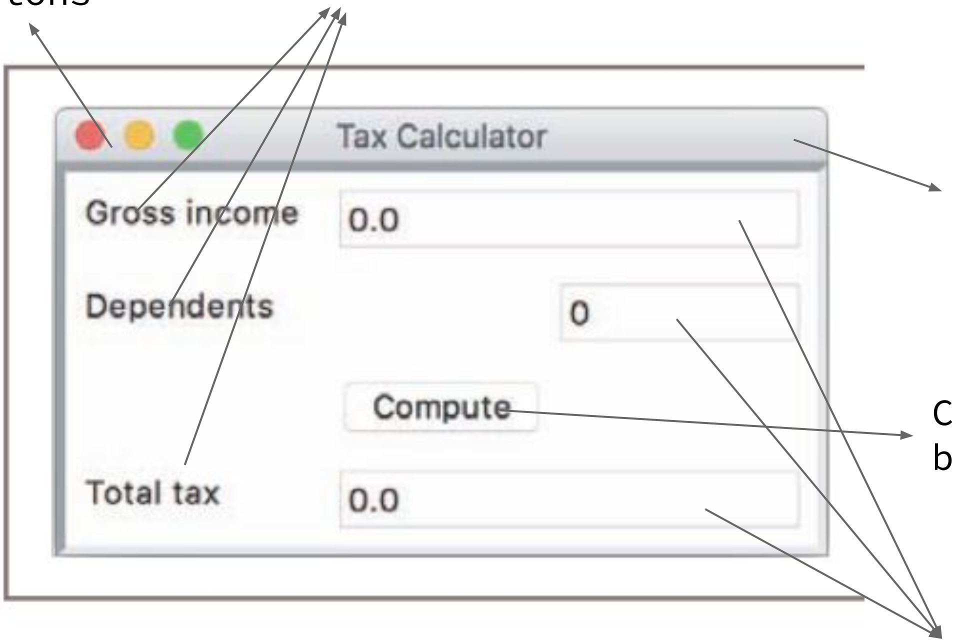
Dependents 2

Total tax 1800.00

Figure 8-2 A GUI-based tax calculator program

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Window Components



Tax Calculator

Gross income 0.0

Dependents 0

Compute

Total tax 0.0

Command buttons

Labels

Title bar

Command button

Entry fields

Figure 8-2 A GUI-based tax calculator program

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Graphical User Interfaces

- user is not constrained to enter inputs in a particular order
- Before pressing the Compute button, can edit any of the data in the two input fields
- Running different data sets does not require re-entering all of the data
- The user can edit just one value and press the Compute button to observe different results

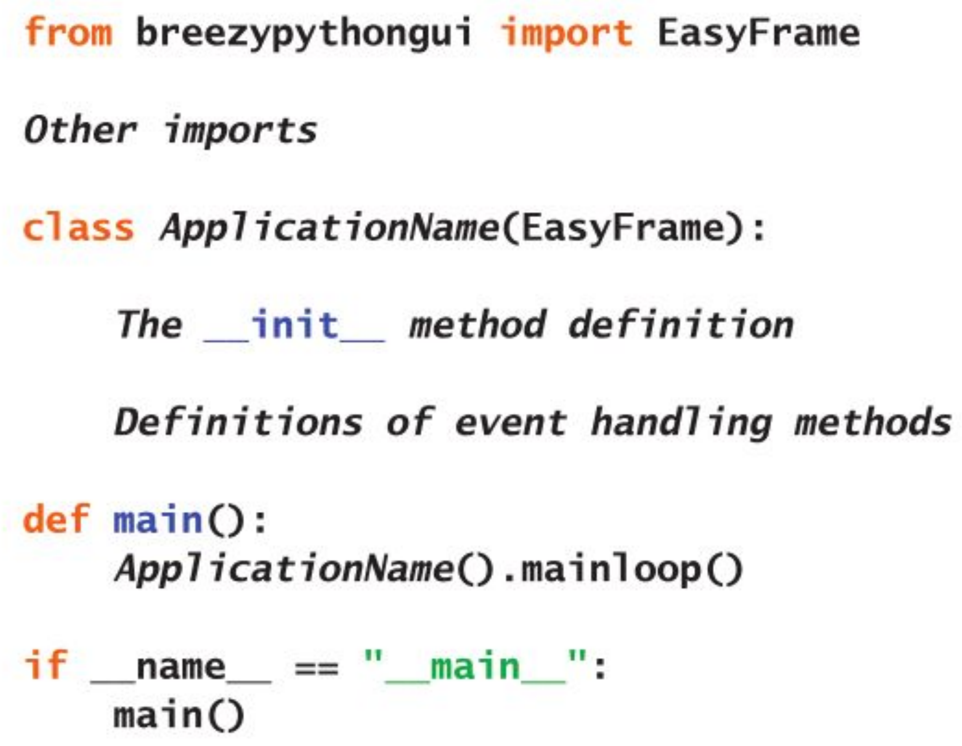
Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Event Driven Programming

- opens a window and waits for the user to manipulate window components with the mouse
- user-generated events, such as mouse clicks, trigger operations in the program to respond by pulling in inputs, processing them, and displaying results

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

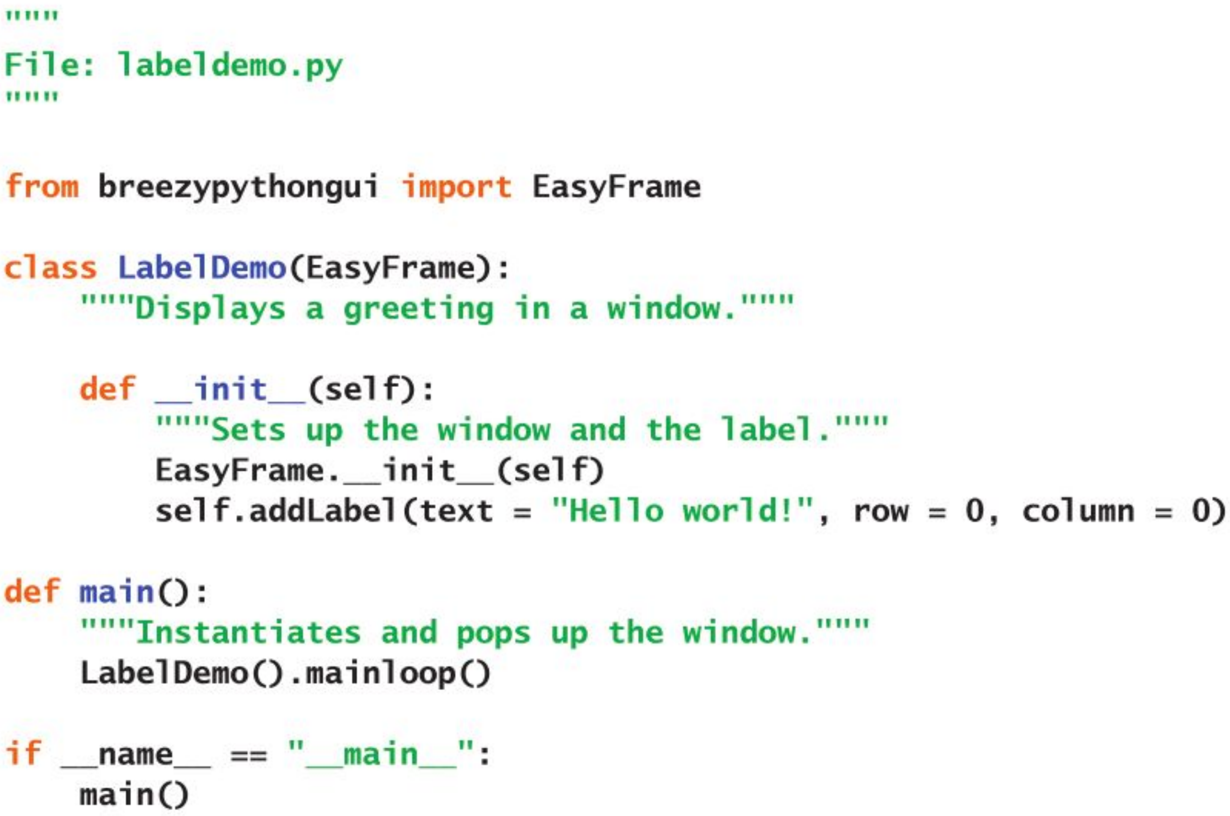
Template for GUI Programs



```
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()
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

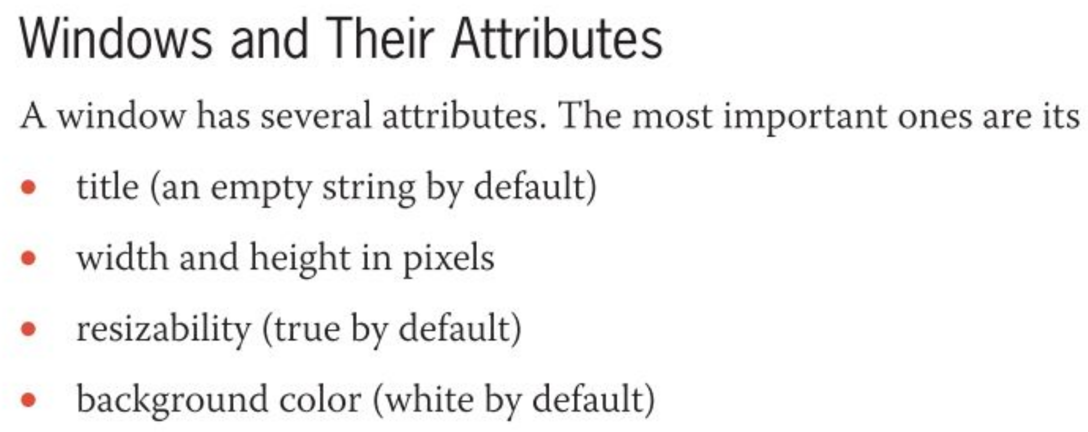
Template for GUI Programs



```
"""
File: labeldemo.py
"""

from breezypythongui import EasyFrame

class LabelDemo(EasyFrame):
    """Displays a greeting in a window."""

    def __init__(self):
        """Sets up the window and the label."""
        EasyFrame.__init__(self)
        self.addLabel(text = "Hello world!", row = 0, column = 0)

def main():
    """Instantiates and pops up the window."""
    LabelDemo().mainloop()

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

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Window



Windows and Their Attributes

A window has several attributes. The most important ones are its

- title (an empty string by default)
- width and height in pixels
- resizability (true by default)
- background color (white by default)

Another way to change a window's attributes is to reset them in the window's attribute dictionary

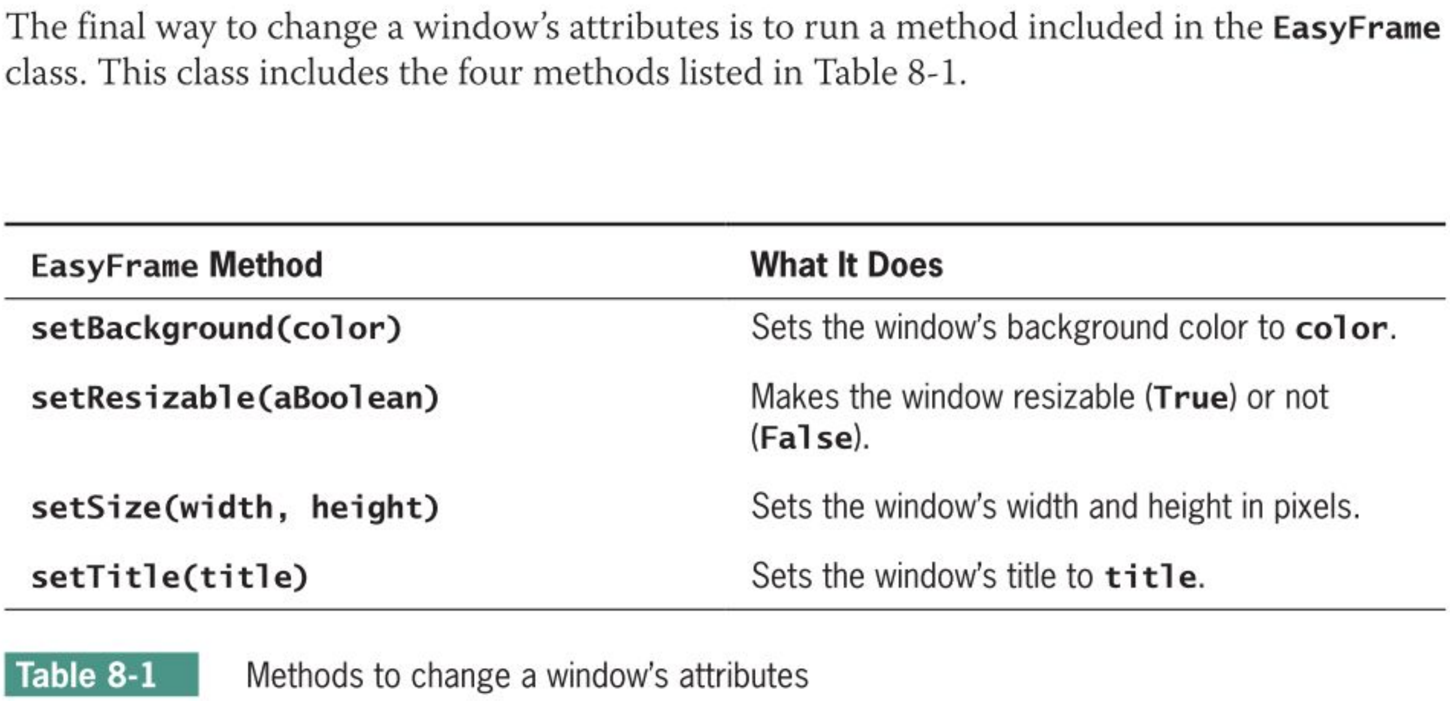
In the labeldemo's __init__ method,

self[“background”]=”yellow”

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

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

Window



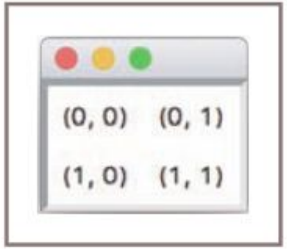
The final way to change a window's attributes is to run a method included in the **EasyFrame** class. This class includes the four methods listed in Table 8-1.

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**.

Table 8-1 Methods to change a window's attributes

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

```
class LayoutDemo(EasyFrame):
    """Displays labels in the quadrants."""

    def __init__(self):
        """Sets up the window and the labels."""
        EasyFrame.__init__(self)
        self.addLabel(text = "(0, 0)", row = 0, column = 0)
        self.addLabel(text = "(0, 1)", row = 0, column = 1)
        self.addLabel(text = "(1, 0)", row = 1, column = 0)
        self.addLabel(text = "(1, 1)", row = 1, column = 1)
```

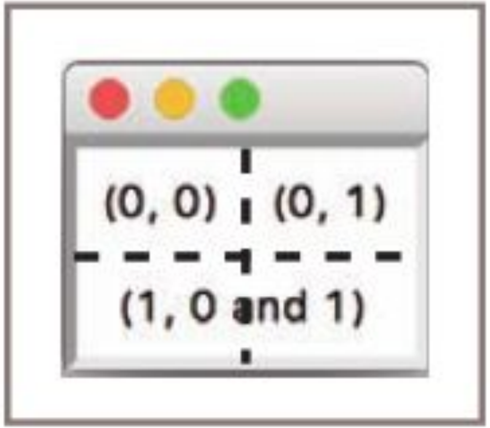


```
(0, 0)   (0, 1)

(1, 0)   (1, 1)
```

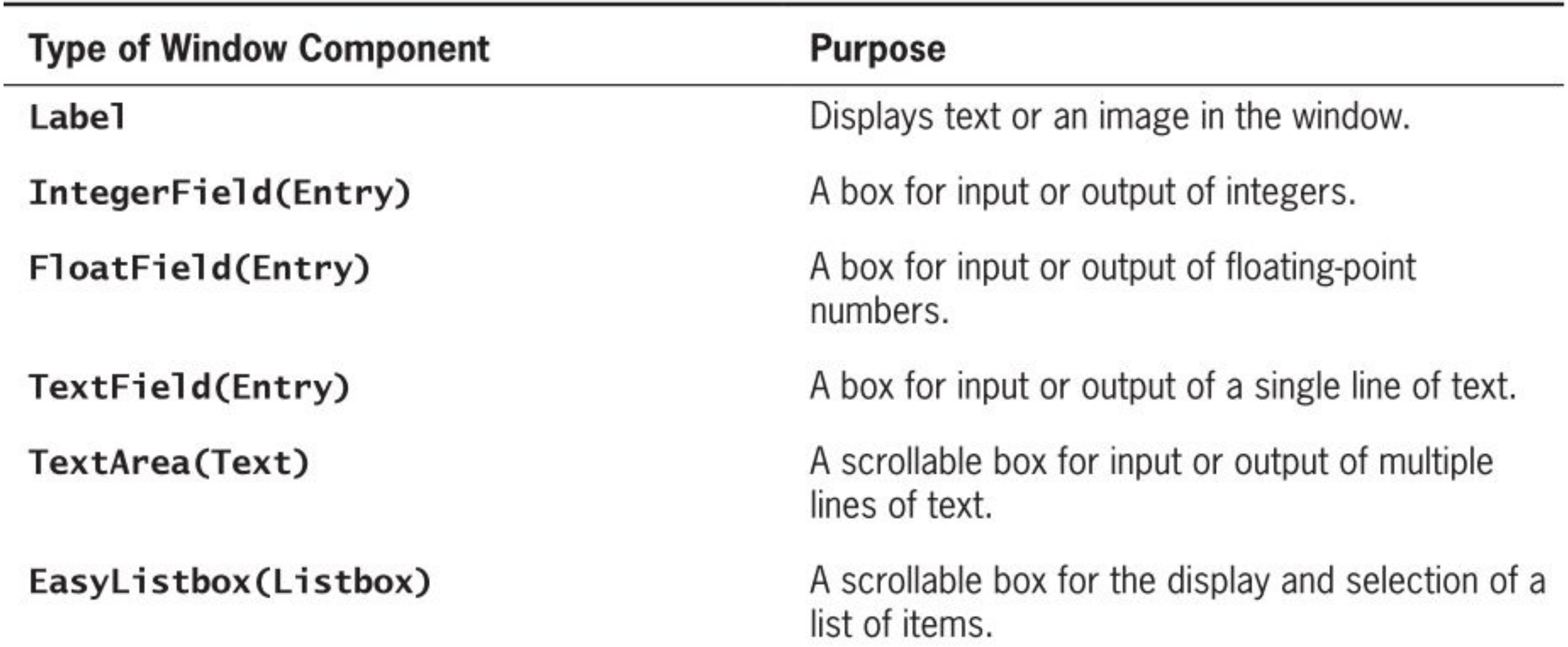
Figure 8-5 Laying out labels in the window's grid

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

```
self.addLabel(text = "(0, 0)", row = 0, column = 0,
              sticky = "NSEW")
self.addLabel(text = "(0, 1)", row = 0, column = 1,
              sticky = "NSEW")
self.addLabel(text = "(1, 0 and 1)", row = 1, column = 0,
              sticky = "NSEW", columnspan = 2)
```

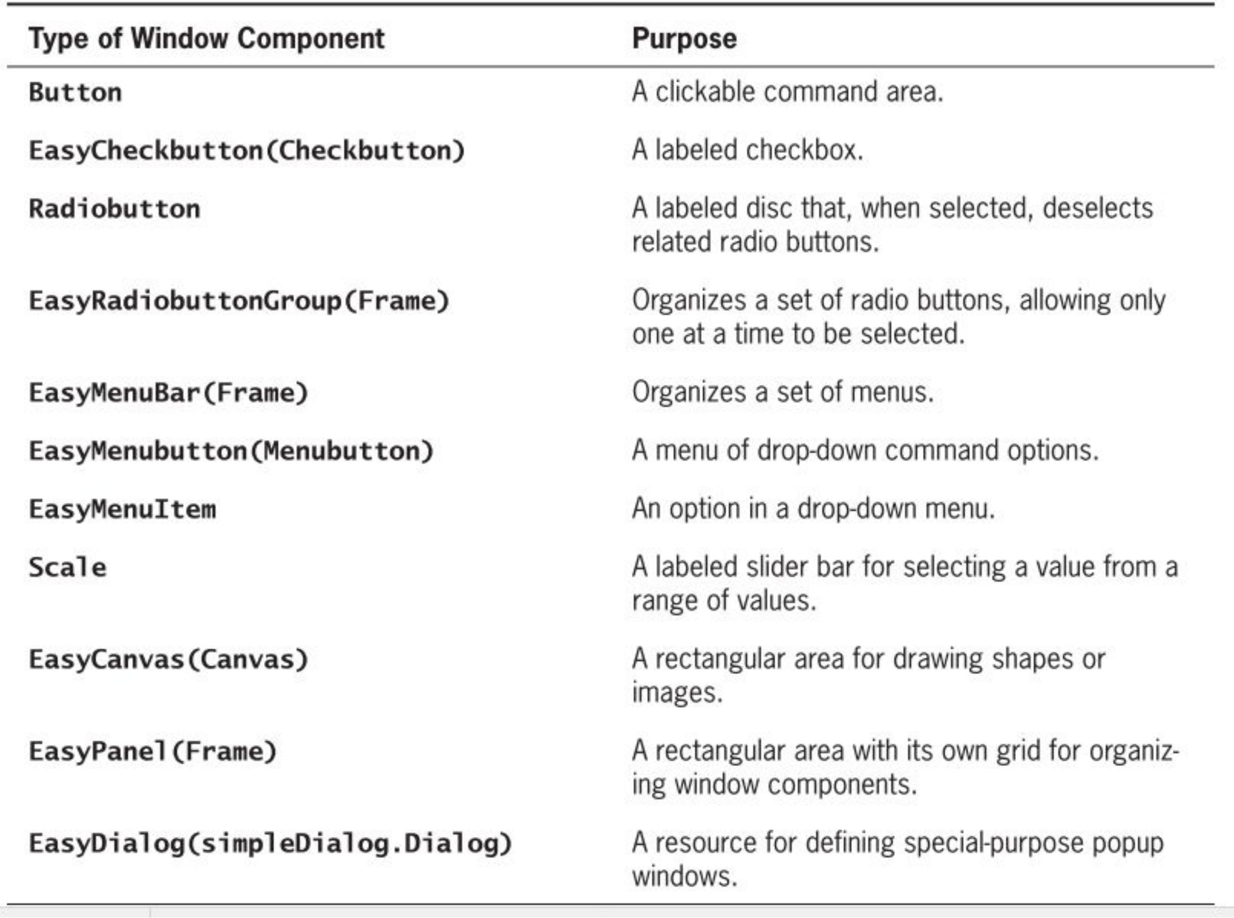


Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



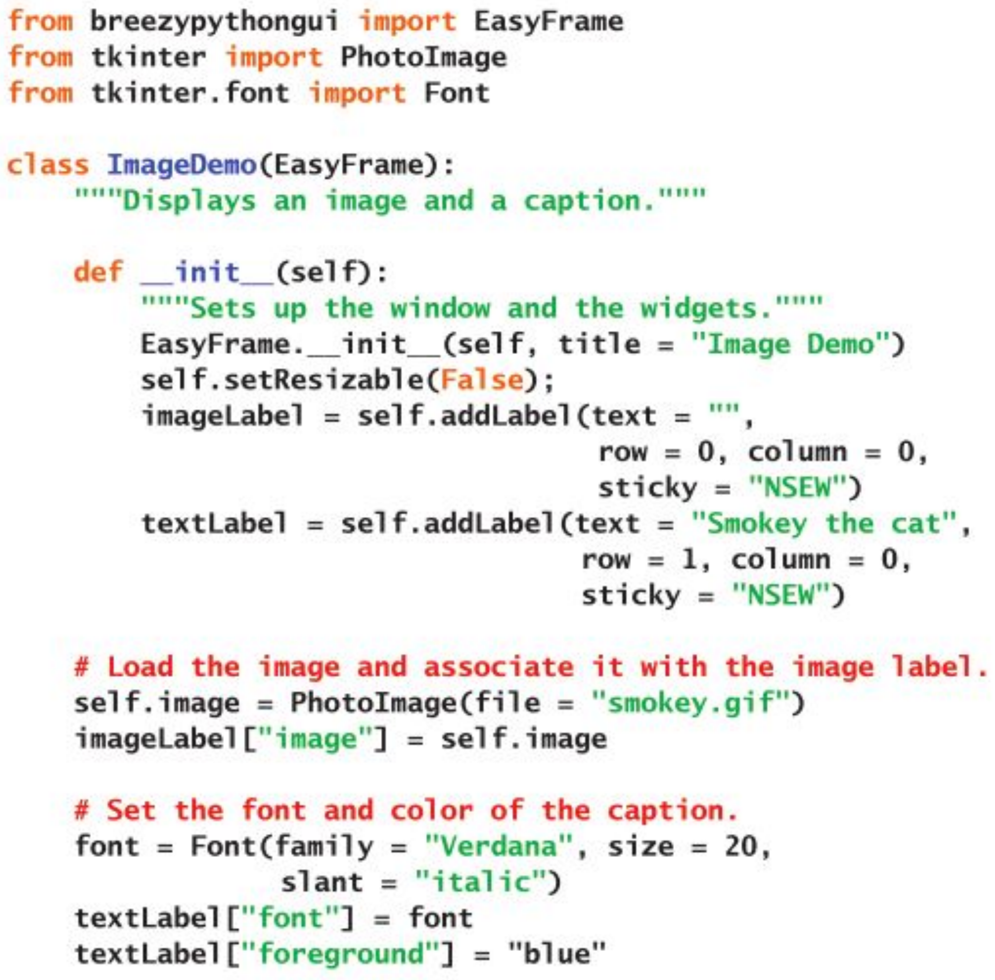
Type of Window Component	Purpose
`Label`	Displays text or an image in the window.
`IntegerField(Entry)`	A box for input or output of integers.
`FloatField(Entry)`	A box for input or output of floating-point numbers.
`TextField(Entry)`	A box for input or output of a single line of text.
`TextArea(Text)`	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.

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



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.

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



```python
from breezypythongui import EasyFrame
from tkinter import PhotoImage
from tkinter.font import Font

class ImageDemo(EasyFrame):
    """Displays an image and a caption."""

    def __init__(self):
        """Sets up the window and the widgets."""
        EasyFrame.__init__(self, title = "Image Demo")
        self.setResizable(False);
        imageLabel = self.addLabel(text = "",
                                   row = 0, column = 0,
                                   sticky = "NSEW")
        textLabel = self.addLabel(text = "Smokey the cat",
                                  row = 1, column = 0,
                                  sticky = "NSEW")

    # Load the image and associate it with the image label.
    self.image = PhotoImage(file = "smokey.gif")
    imageLabel["image"] = self.image

    # Set the font and color of the caption.
    font = Font(family = "Verdana", size = 20,
                slant = "italic")
    textLabel["font"] = font
    textLabel["foreground"] = "blue"
```

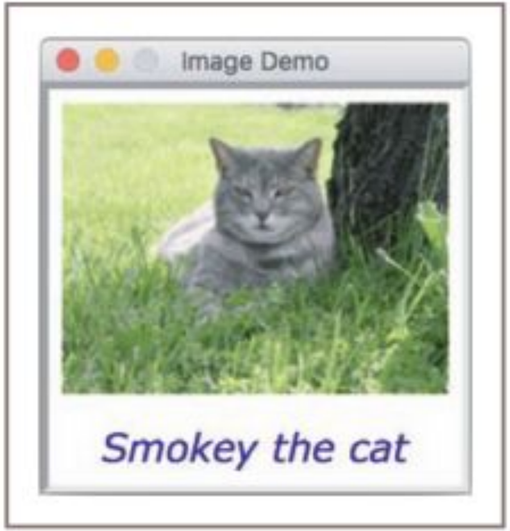
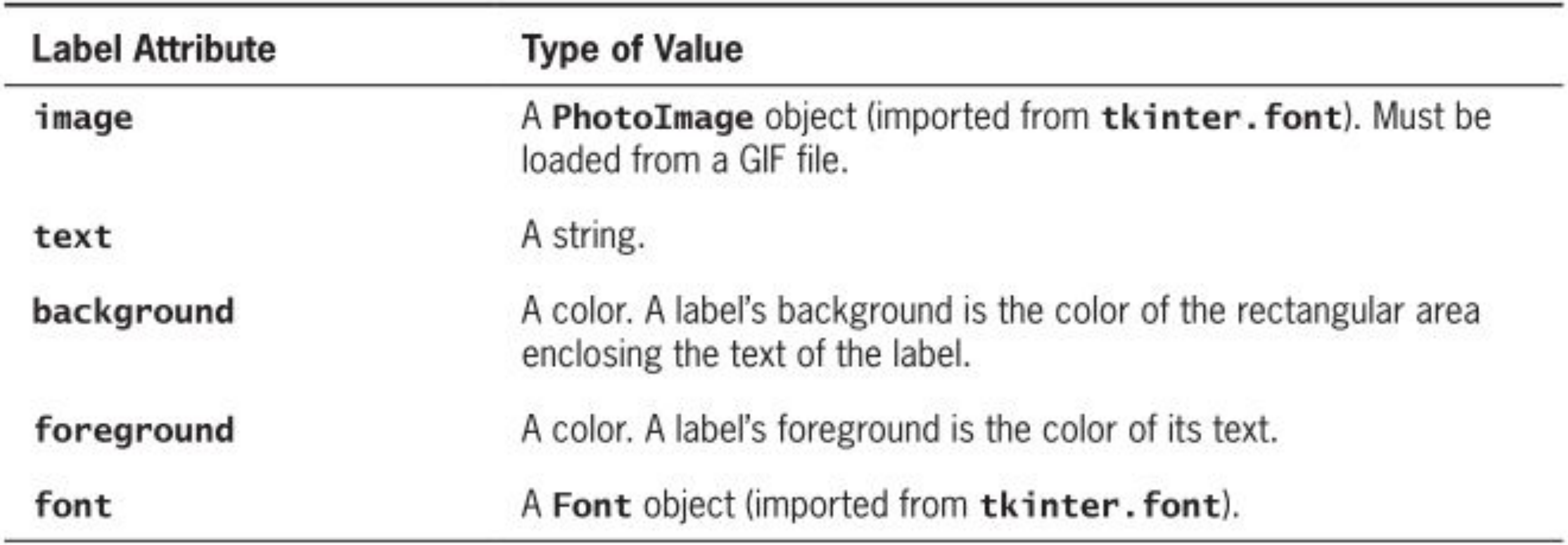


Figure 8-7 Displaying a captioned image

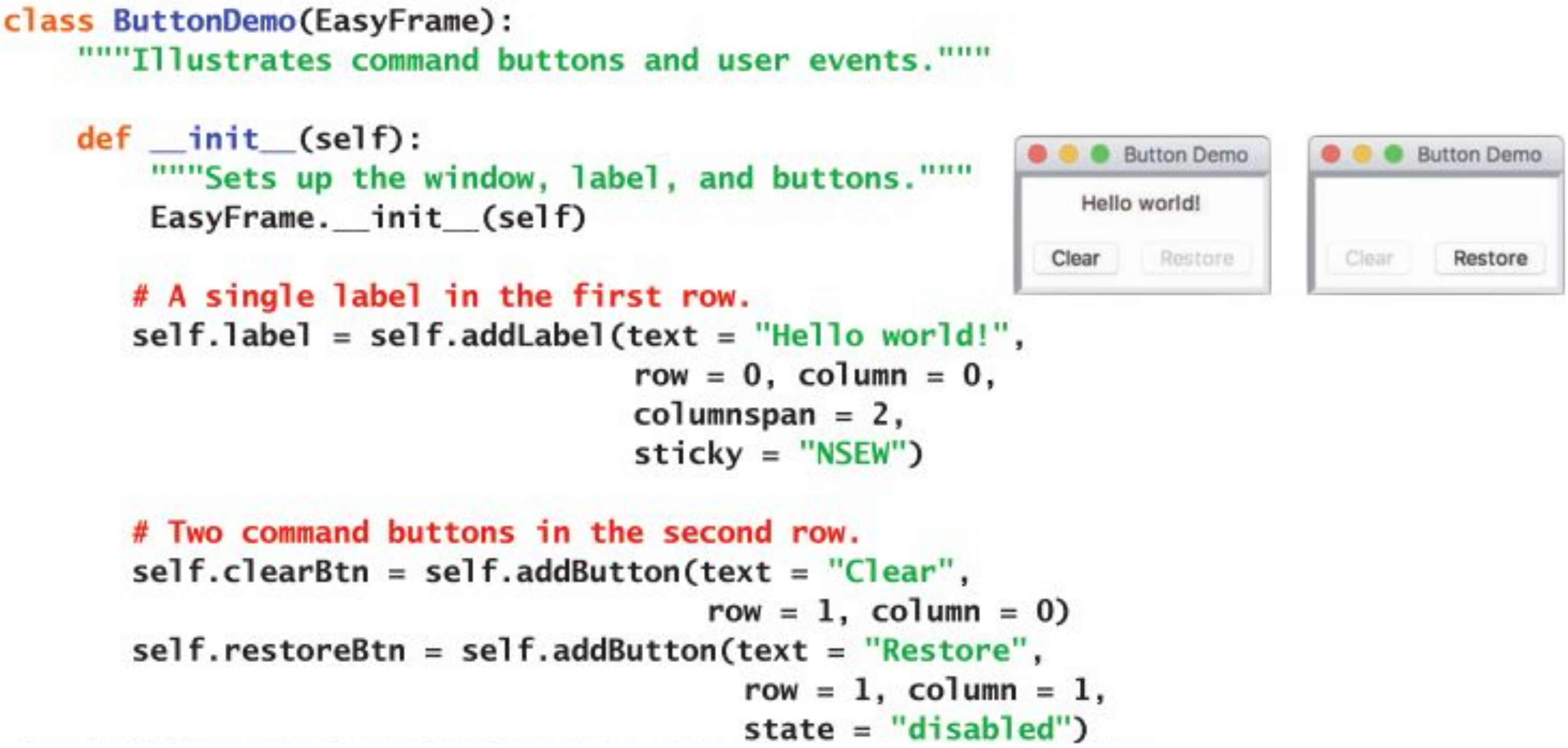
Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



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**).

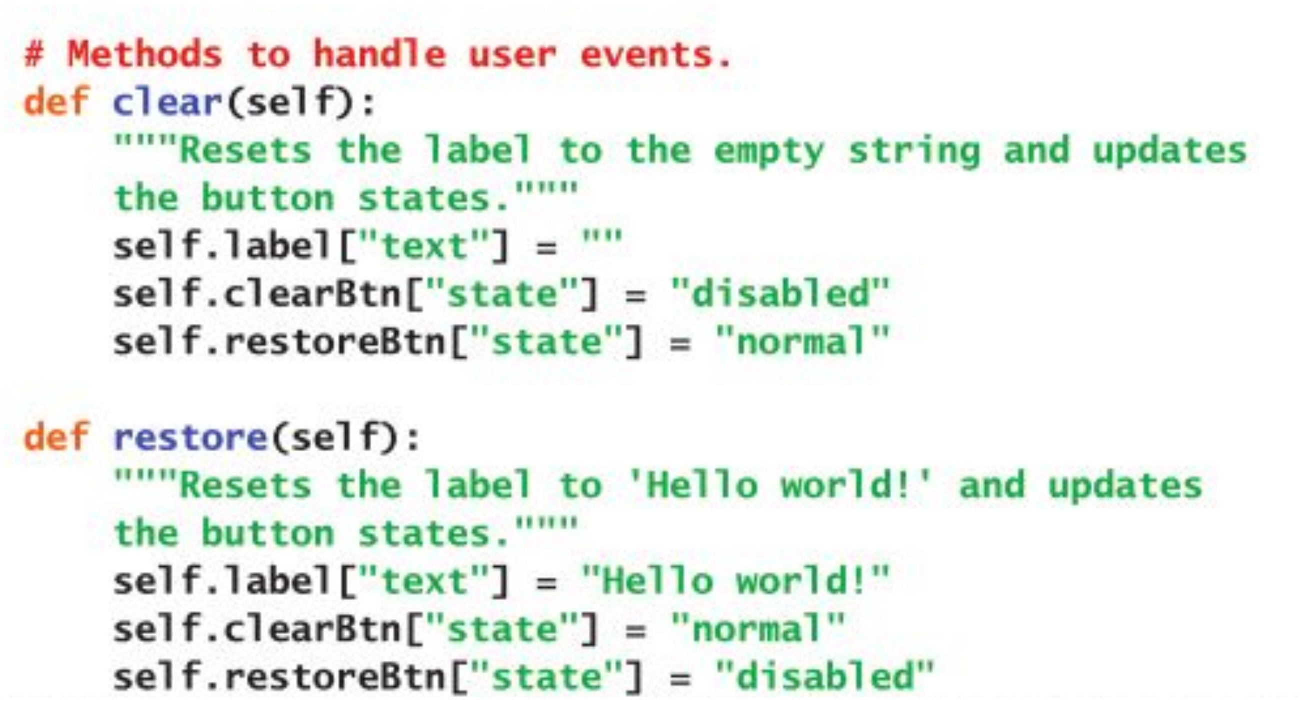
Table 8-3 The **tkinter.Label** attributes

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



```python
class ButtonDemo(EasyFrame):
    """Illustrates command buttons and user events."""

    def __init__(self):
        """Sets up the window, label, and buttons."""
        EasyFrame.__init__(self)

        # A single label in the first row.
        self.label = self.addLabel(text = "Hello world!",
                                   row = 0, column = 0,
                                   columnspan = 2,
                                   sticky = "NSEW")

        # Two command buttons in the second row.
        self.clearBtn = self.addButton(text = "Clear",
                                       row = 1, column = 0)
        self.restoreBtn = self.addButton(text = "Restore",
                                         row = 1, column = 1,
                                         state = "disabled")
```

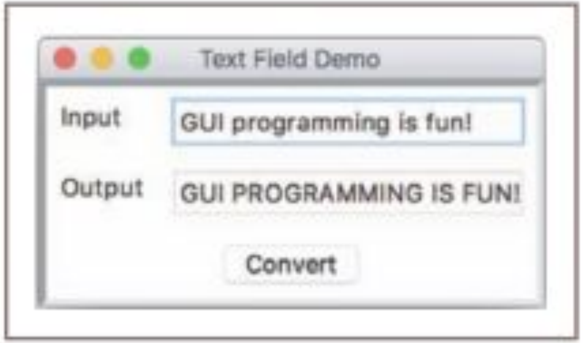
Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition



```python
# Methods to handle user events.
def clear(self):
    """Resets the label to the empty string and updates
    the button states."""
    self.label["text"] = ""
    self.clearBtn["state"] = "disabled"
    self.restoreBtn["state"] = "normal"

def restore(self):
    """Resets the label to 'Hello world!' and updates
    the button states."""
    self.label["text"] = "Hello world!"
    self.clearBtn["state"] = "normal"
    self.restoreBtn["state"] = "disabled"
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition

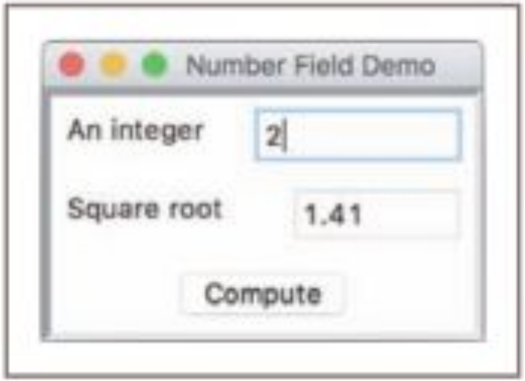
```
class TextFieldDemo(EasyFrame):
    """Converts an input string to uppercase and displays
    the result."""

    def __init__(self):
        """Sets up the window and widgets."""
        EasyFrame.__init__(self, title = "Text Field Demo")

        # Label and field for the input
        self.addLabel(text = "Input", row = 0, column = 0)
        self.inputField = self.addTextField(text = "",
                                            row = 0,
                                            column = 1)

       # Label and field for the output
       self.addLabel(text = "Output", row = 1, column = 0)
       self.outputField = self.addTextField(text = "",
                                            row = 1,
                                            column = 1,
                                            state = "readonly")

       # The command button
       self.addButton(text = "Convert", row = 2, column = 0,
                      columnspan = 2, command = self.convert)

    # The event handling method for the button
    def convert(self):
        """Inputs the string, converts it to uppercase,
        and outputs the result."""
        text = self.inputField.getText()
        result = text.upper()
        self.outputField.setText(result)
```

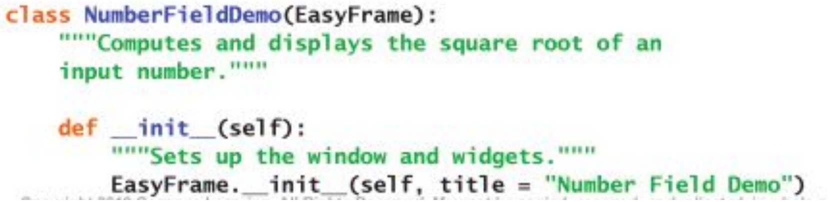


GUI programming is fun!

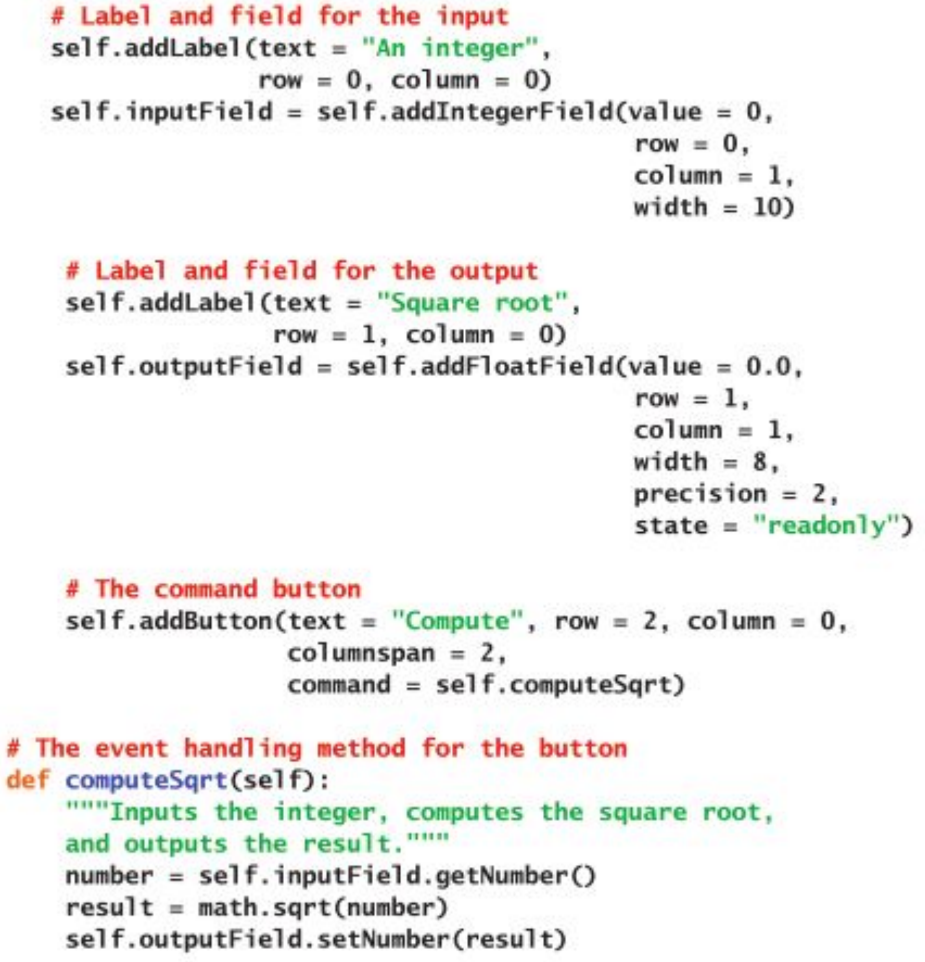
GUI PROGRAMMING IS FUN!

Ref: Kenneth A Lambert, Fundamentals of first python programs, 2nd edition





```
class NumberFieldDemo(EasyFrame):
    """Computes and displays the square root of an
    input number."""

    def __init__(self):
        """Sets up the window and widgets."""
        EasyFrame.__init__(self, title = "Number Field Demo")
```



```
# Label and field for the input
self.addLabel(text = "An integer",
              row = 0, column = 0)
self.inputField = self.addIntegerField(value = 0,
                                       row = 0,
                                       column = 1,
                                       width = 10)

 # Label and field for the output
 self.addLabel(text = "Square root",
               row = 1, column = 0)
 self.outputField = self.addFloatField(value = 0.0,
                                       row = 1,
                                       column = 1,
                                       width = 8,
                                       precision = 2,
                                       state = "readonly")

 # The command button
 self.addButton(text = "Compute", row = 2, column = 0,
                columnspan = 2,
                command = self.computeSqrt)

# The event handling method for the button
def computeSqrt(self):
    """Inputs the integer, computes the square root,
    and outputs the result."""
    number = self.inputField.getNumber()
    result = math.sqrt(number)
    self.outputField.setNumber(result)
```

Ref. Kenneth A Lambert, Fundamentals of first python programs, 2nd edition